



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

SCHOOL OF COMPUTING

COMMON TO CSE AND IT

UNIT – I – Customer Interface Design and Development – SITA1502

UNIT 1 HTML, XML, CSS AND RWD

Introduction To HTML- DHTML, XML – Structuring XML document using DTD – Schemas – XML parsers – DOM – SAX presentation technologies – XSL – XFORMS – XHTML – Transformations – XSLT – XLINK – XPATH – XQuery. Responsive Web Design-Intro-Fluid Grid-Viewport-Media Queries Images. Introduction To CSS-Syntax, Selectors-Types of style sheets

INTRODUCTION TO HTML

- HTML is the standard markup language for Web pages.
- With HTML you can create your own Website.
- HTML is easy to learn - You will enjoy it!



What is HTML?

Fig. 1.1 HTML code

- HTML stands for Hyper Text Markup Language
- HTML is the standard markup language for creating web pages
- HTML describes the structure of a web page
- HTML consists of a series of elements
- HTML elements tell the browser how to display the content
- HTML elements label pieces of content such as "this is a heading", "this is a paragraph", "this is a link", etc.

What is an HTML Element?

- An HTML element is defined by a start tag, some content, and an end tag:
- `<tagname>Content goes here...</tagname>`

The HTML element is everything from the start tag to the end tag:

`<h1>My First Heading</h1>`

`<p>My first paragraph.</p>`

“Normal text” surrounded by bracketed tags that tell browsers how to display web pages

Basic Requirements

- 1. Editor (text)

- 2. web browser



Fig. 1.2 Web Browsers

Web Browsers

- The purpose of a web browser (Chrome, Edge, Firefox, Safari) is to read HTML documents and display them correctly.
- A browser does not display the HTML tags, but uses them to determine how to display the document.

Web Browsers

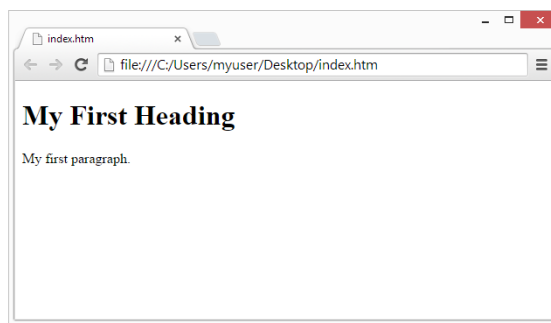


Fig. 1.3 output page

HTML Page Structure

Below is a visualization of an HTML page structure:

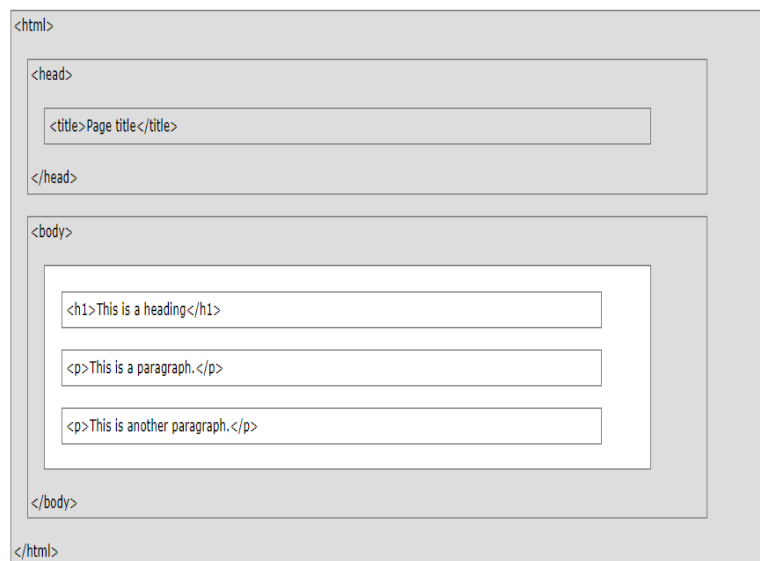


Fig. 1.4 HTML Page Structure

Note: The content inside the <body> section will be displayed in a browser. The content inside the <title> element will be shown in the browser's title bar or in the page's tab.

HTML History

HTML History

Since the early days of the World Wide Web, there have been many versions of HTML:

Year	Version
1989	Tim Berners-Lee invented www
1991	Tim Berners-Lee invented HTML
1993	Dave Raggett drafted HTML+
1995	HTML Working Group defined HTML 2.0
1997	W3C Recommendation: HTML 3.2
1999	W3C Recommendation: HTML 4.01
2000	W3C Recommendation: XHTML 1.0
2008	WHATWG HTML5 First Public Draft
2012	WHATWG HTML5 Living Standard
2014	W3C Recommendation: HTML5
2016	W3C Candidate Recommendation: HTML 5.1
2017	W3C Recommendation: HTML5.1 2nd Edition
2017	W3C Recommendation: HTML5.2

Fig. 1.5 HTML History

FIRST SAMPLE PROGRAM

- <html>
- <head>
- <title>Sample program-2</title>
- </head>
- <body>
- WELCOME TO CIDD COURSE- DEAR STUDENTS
- </body>
- </html>

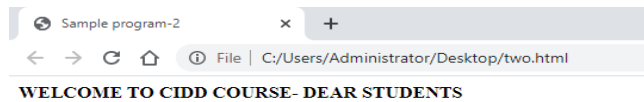


Fig. 1.6 SAMPLE PROGRAM- HEADING TAGS

- <html>
- <head>
- < title>Sample program-3</title>
- </head>
- <body>
- <h1>WELCOME TO CIDD COURSE-DEAR STUDENTS</h1>
- <h2>WELCOME TO CIDD COURSE-DEAR STUDENTS</h2>
- <h3>WELCOME TO CIDD COURSE-DEAR STUDENTS</h3>
- <h4>WELCOME TO CIDD COURSE-DEAR STUDENTS</h4>
- <h5>WELCOME TO CIDD COURSE-DEAR STUDENTS</h5>
- <h6>WELCOME TO CIDD COURSE-DEAR STUDENTS</h6>
- </html>

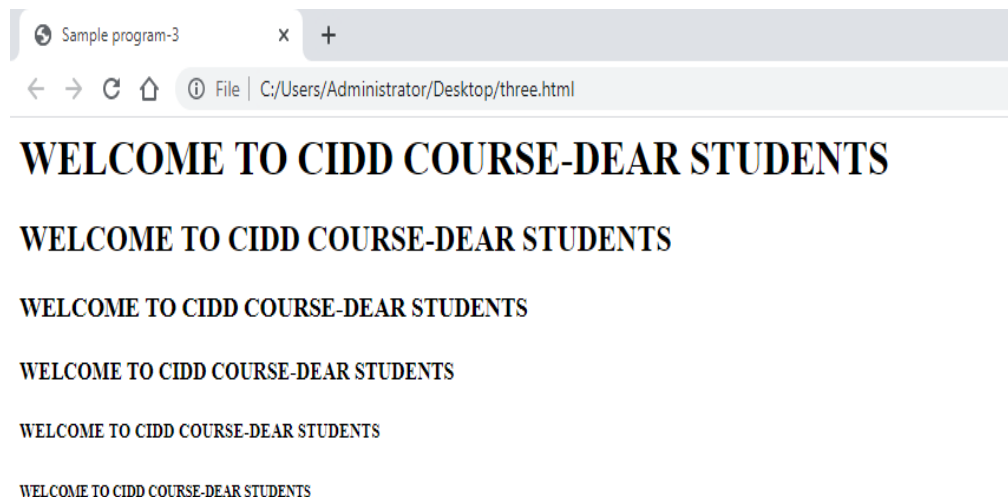


Fig. 1.7 Output -Heading tags

Sample program -text written outside the font tag

- `<html>`
- `<head>`
- `<title>Sample program-4 </title>`
- `</head>`
- `<body>`
- ``
- WELCOME TO CIDD COURSE-DEAR STUDENTS
- ``
- WELCOME TO SATHYABAMA INSTITUTE OF TECHNOLOGY
- `</html>`



Fig. 1.8 output -font tag

SAMPLE PROGRAM – BREAK TAG

- `<html>`
- `<head>`
- `<title>Sample program-5 </title>`
- `</head>`
- `<body>`
- ``
- WELCOME TO CIDD COURSE-DEAR STUDENTS
- ``
- `
`WELCOME TO SATHYABAMA INSTITUTE OF TECHNOLOGY

- `</html>`



Fig. 1.9 Break tag

HTML FORMATTING ELEMENTS

Formatting elements were designed to display special types of text:

- `` - Bold text
- `` - Important text
- `<i>` - Italic text
- `` - Emphasized text
- `<mark>` - Marked text
- `<small>` - Smaller text
- `` - Deleted text
- `<ins>` - Inserted text
- `<sub>` - Subscript text
- `<sup>` - Superscript text

SAMPLE PROGRAM 6- WITH BOLD TAG AND WITH OUT BOLD TAG

- `<html>`
- `<head>`
- `<title>Sample program 6 </title>`
- `</head>`
- `<body>`
- ``
- `WELCOME TO CIDD COURSE-DEAR STUDENTS`
- `
`

- WELCOME TO CIDD COURSE-DEAR STUDENTS
-
- </html>



Fig. 1.10 Bold Tag

SAMPLE PROGRAM - UNDERLINE AND ITALIC

- <html>
- <head>
- <title>Sample program 7</title>
- </head>
- <body>
-
- WELCOME TO CIDD COURSE-<u>DEAR STUDENTS</u>
-

- <i>WELCOME TO SATHYABAMA INSTITUTE OF TECHNOLOGY</i>
- </html>



Fig:1.11 output -Italic Tag

SAMPLE PROGRAM 8- STRIKE TAG

- <html>
- <head>
- <title>SAMPLE PROGRAM 8 </title>
- </head>
- <body>

- ``
- `WELCOME TO CIDD COURSE-<u><q>DEAR STUDENTS</q></u>`
- `
`
- `<i>WELCOME TO SATHYABAMA INSTITUTE OF TECHNOLOGY</i>`
- `<strike> BREAK THE CHAIN</strike>`
- `</html>`



Fig: 1.12 Strike Tag

SAMPLE PROGRAM 10- FOR QUOTATION TAG

- `<html>`
- `<head>`
- `<title>sample program 10 </title>`
- `</head>`
- `<body>`
- ``
- `WELCOME TO CIDD COURSE-<u><q>DEAR STUDENTS</q></u>`
- ``
- `</html>`

WELCOME TO CIDD COURSE-"DEAR STUDENTS"

Fig: 1.13 output- Quotation tags

HTML TABLES

- HTML tables allow web developers to arrange data into rows and columns
- The `<table>` tag defines an HTML table.

- Each table row is defined with a <tr> tag.
- Each table header is defined with a <th> tag.
- Each table data/cell is defined with a <td> tag.
- By default, the text in <th> elements are bold and centered.
- By default, the text in <td> elements are regular and left-aligned.

Sample program 11 - HTML Table -& Border

- <html>
- <head>
- <style>
- table, th, td {
- border: 1px solid black;
- }
- </style>
- </head>
- <body>
- <h2>Table With Border</h2>
- <p>Use the CSS border property to add a border to the table.</p>
- <table style="width:100%">
- <tr>
- <th>Firstname</th>
- <th>Lastname</th>
- <th>Age</th>
- </tr>
- <tr>
- <td>Jill</td>
- <td>Smith</td>
- <td>50</td>
- </tr>
- <tr>
- <td>Eve</td>
- <td>Jackson</td>

- `<td>94</td>`
- `</tr>`
- `<tr>`
- `<td>John</td>`
- `<td>Doe</td>`
- `<td>80</td>`
- `</tr>`
- `</table>`
- `</body>`
- `</html>`

Table With Border

Use the CSS border property to add a border to the table.

Firstname	Lastname	Age
Jill	Smith	50
Eve	Jackson	94
John	Doe	80

Fig. 1.14 output table

HTML FRAMES

- HTML frames are used to divide your browser window into multiple sections.
- Each section can load a separate HTML document.
- A collection of frames in the browser window is known as a frameset.
- The window is divided into frames in a similar way the tables are organized: into rows and columns.

CREATING FRAMES

- To use frames on a page we use `<frameset>` tag instead of `<body>` tag.
- The `<frameset>` tag defines, how to divide the window into frames.
- The rows attribute of `<frameset>` tag defines horizontal frames and cols attribute defines vertical frames.
- Each frame is indicated by `<frame>` tag and it defines which HTML document shall open into the frame.

FRAMES

- A framed page is actually made up of multiple HTML pages.

- There is one HTML document that describes how to break up the single browser window into multiple windowpanes.
- Each windowpane is filled with an HTML document.
- For example to make a framed page with a windowpane on the left and one on the right requires three html pages.
- Doc1.Html and Doc2.Html are the pages that contain content.
- Frames.Html is the page that describes the division of the single browser window into two windowpanes.



Fig. 1.15 frames



Fig. 1.16 frames

Forms

- An HTML form is used to collect user input. The user input is most often sent to a server for processing.
- The HTML `<form>` element is used to create an HTML form for user input
- `<form>`
- .
- form elements
- .
- `</form>`

- The HTML `<input>` element is the most used form element.
- An `<input>` element can be displayed in many ways, depending on the type attribute.

Type	Description
<code><input type="text"></code> single-line text input field	Displays a
1. <code><input type="radio"></code> button (for selecting one of many choices)	Displays a radio
2. <code><input type="checkbox"></code> (for selecting zero or more of many choices)	Displays a checkbox
3. <code><input type="submit"></code> button (for submitting the form)	Displays a submit
4. <code><input type="button"></code> button	Displays a clickable

The `<input>` Element

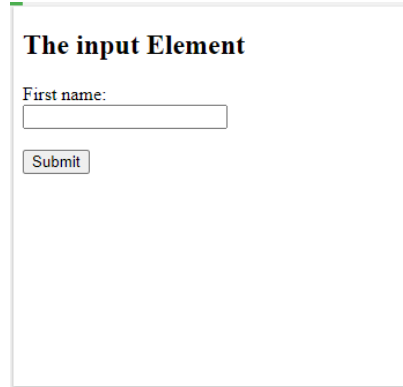
- `<Html>`
- `<Body>`
- `<H2>The input Element</h2>`
- `<Form action="/action_page.php">`
- `<Label for="fname">first name:</label>
`
- `<Input type="text" id="fname" name="fname">

`
- `<Input type="submit" value="submit">`
- `</Form>`

- `</Body>`

`</Html>`

Note: 'for' attribute of the `<label>` tag should be equal to the id attribute of the `<input>` element to bind them together



The screenshot shows a web browser window with a form titled "The input Element". Inside the form, there is a label "First name:" followed by a text input field. Below the input field is a button labeled "Submit".

Fig:1.17 -output -form

The `<label>` Element

- The `<label>` element defines a label for several form elements.
- The `<label>` element is useful for screen-reader users, because the screen-reader will read out loud the label when the user focus on the input element.
- The `<label>` element also help users who have difficulty clicking on very small regions (such as radio buttons or checkboxes) - because when the user clicks the text within the `<label>` element, it toggles the radio button/checkbox.
- The for attribute of the `<label>` tag should be equal to the id attribute of the `<input>` element to bind them together.

The `<select>` Element

The `<select>` element defines a drop-down list

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h2>The select Element</h2>
```

```
<p>The select element defines a drop-down list:</p>
```

```
<form action="/action_page.php">
```

```
<label for="cars">Choose a car:</label>

<select id="cars" name="cars">

  <option value="volvo">Volvo</option>

  <option value="saab">Saab</option>

  <option value="fiat">Fiat</option>

  <option value="audi">Audi</option>

</select>

<input type="submit">

</form></body></html>
```

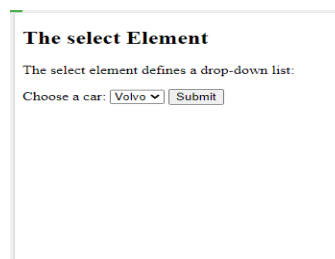


Fig: 1.18 -output-form

XML

- XML stands for Extensible Markup Language. It is a text-based markup language derived from Standard Generalized Markup Language (SGML).
- XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags, which are used to display the data. XML is not going to replace HTML in the near future, but it introduces new possibilities by adopting many successful features of HTML.
- There are three important characteristics of XML that make it useful in a variety of systems and solutions –
- **XML is extensible** – XML allows you to create your own self-descriptive tags, or language, that suits your application.

- **XML carries the data, does not present it** – XML allows you to store the data irrespective of how it will be presented.
- **XML is a public standard** – XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard.

XML Usage

A short list of XML usage says it all –

- XML can work behind the scene to simplify the creation of HTML documents for large web sites.
- XML can be used to exchange the information between organizations and systems.
- XML can be used for offloading and reloading of databases.
- XML can be used to store and arrange the data, which can customize your data handling needs.
- XML can easily be merged with style sheets to create almost any desired output.
- Virtually, any type of data can be expressed as an XML document.

```
<?xml version = "1.0"?>
<contact-info>
  <name>Tanmay Patil</name>
  <company>TutorialsPoint</company>
  <phone>(011) 123-4567</phone>
</contact-info>
```

- You can notice there are two kinds of information in the above example –
- Markup, like <contact-info>
- The text, or the character data, Tutorials Point and (040) 123-4567.
- The following diagram depicts the syntax rules to write different types of markup and text in an XML document.

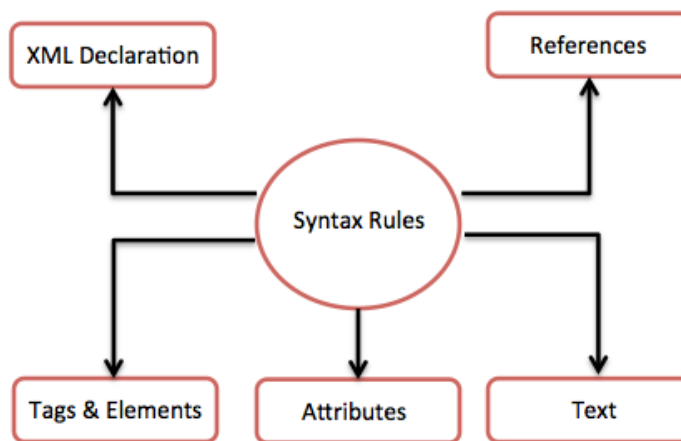


Fig. 1.19 Syntax Rules

Syntax Rules for XML Declaration

- The XML declaration is case sensitive and must begin with "<?xml>" where "xml" is written in lower-case.
- If document contains XML declaration, then it strictly needs to be the first statement of the XML document.
- The XML declaration strictly needs be the first statement in the XML document.
- An HTTP protocol can override the value of *encoding* that you put in the XML declaration.

Tags and Elements

- An XML file is structured by several XML-elements, also called XML-nodes or XML-tags. The names of XML-elements are enclosed in triangular brackets < > as shown below –

```
<element>
```

Syntax Rules for Tags and Elements

Element Syntax – Each XML-element needs to be closed either with start or with end elements as shown below –

```
<element>....</element>
```

or in simple-cases, just this way –

```
<element/>
```

DOM

The **Document Object Model (DOM)** is the data representation of the objects that comprise the structure and content of a document on the web. This guide will introduce the DOM, look at how the DOM represents an HTML document in memory and how to use APIs to create web content and applications.

What is the DOM?

The Document Object Model (DOM) is a programming interface for web documents. It represents the page so that programs can change the document structure, style, and content. The DOM represents the document as nodes and objects; that way, programming languages can interact with the page.

A web page is a document that can be either displayed in the browser window or as the HTML source. In both cases, it is the same document but the Document Object Model (DOM) representation allows it to be manipulated. As an object-oriented representation of the web page, it can be modified with a scripting language such as JavaScript.

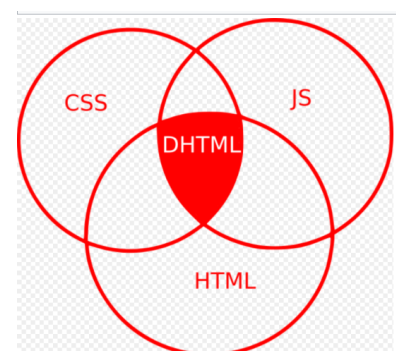
For example, the DOM specifies that the query SelectorAll method in this code snippet must return a list of all the <p> elements in the document:

DHTML

- Dynamic HTML, or DHTML, is a collection of technologies used together to create interactive and animated websites by using a combination of a static markup language (such as HTML), a client-side scripting language (such as JavaScript), a presentation definition language (such as CSS), and the Document Object Model (DOM).

What is DHTML?

- It is considered to be made up of
 - HTML
 - Cascading Style Sheets (CSS)
 - Scripting language
- All three of these components are linked via Document Object Model (DOM)
- DOM is the interface that allows scripting languages to access the content, style, and structure of the web documents and change them dynamically



Define what is DHTML?

- DHTML stands for “dynamic hypertext transfer markup language”.
- DHTML is not a language.
- DHTML is a term describing the art of making dynamic & interactive WebPages.
- Designed to enhance a web user’s experience.

☐ Define DHTML cont....

- *DHTML is a combination of web development technologies used to create dynamically changing websites.*
- *WebPages may include animation, dynamic menus & text effects.*
- *The technologies used include a combination of HTML, JavaScript, or VBScript, CSS & the document object model(DOM).*

Advantages of DHTML

1. **DHTML** is more efficient for content management purposes.
2. **Results in faster and fresher** content and visual appeal.
3. **DHTML** for web design is that it is easier to create good internal linking and cross linking.
4. **Small file size:-** DHTML files are small compared to other interactive media such as flash or shockwave. Therefore they have a shorter download time & take up less bandwidth.
5. **Supported by both major browser manufactures:-** both Microsoft & Netscape currently support DHTML in some shape or form.

- XSL (Extensible Stylesheet Language), formerly called Extensible Style Language, is a language for creating a style sheet that describes how data sent over the Web using the Extensible Markup Language (XML) is to be presented to the user.

XML parsers

- An XML parser is a software library or package that provides interfaces for client applications to work with an XML document. The XML Parser is designed to read the XML and create a way for programs to use XML. XML parser validates the document and check that the document is well formatted.

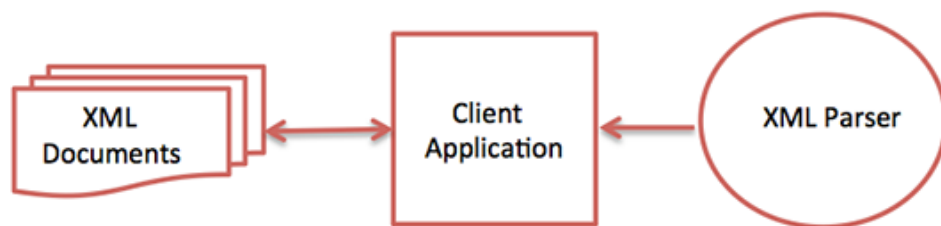


Fig. 1.20 XML parser

The goal of a parser is to transform XML into a readable code.

To ease the process of parsing, some commercial products are available that facilitate the breakdown of XML document and yield more reliable results.

Some commonly used parsers are listed below –

- **MSXML (Microsoft Core XML Services)** – This is a standard set of XML tools from Microsoft that includes a parser.
- **System.Xml.XmlDocument** – This class is part of .NET library, which contains a number of different classes related to working with XML.
- **Java built-in parser** – The Java library has its own parser. The library is designed such that you can replace the built-in parser with an external implementation such as Xerces from Apache or Saxon.
- **Saxon** – Saxon offers tools for parsing, transforming, and querying XML.
- **Xerces** – Xerces is implemented in Java and is developed by the famous open source Apache Software Foundata

What is XForms

XForms provides a lot of new capabilities for web developers by providing a rich interactive experience. It also maintains many of the familiar aspects of creating forms in HTML. Following are the main points specifying what XForms is:

- XForms is the next generation of HTML forms.
- XForms is richer and more flexible than HTML forms.
- XForms will be the forms standard in XHTML 2.0
- XForms is platform independent.
- XForms is device independent.
- XForms separates data and logic from presentation.
- XForms uses XML to define form data.
- XForms stores and transports data in XML documents.
- XForms contains features like calculations and validations of forms.
- XForms reduces or eliminates the need for scripting.
- XForms is a W3C Recommendation.

WHAT SHOULD YOU KNOW BEFORE XFORMS

You must have basic knowledge of HTML, HTML Forms, XHTML, and XML before learn XForm.

XForms Features

- XForms is a new generation of HTML forms. It inherits some properties of HTML forms and extends some other properties. In other word, XForms can be called the successor of HTML forms.
- XForms uses XML for data definition and HTML or XHTML to display data.
- XForms separates the data logic of the form from its presentation.
- XForms uses XML to define form data.
- XForms uses XML to store and transport data. The data displayed in a form are stored in an XML document, and the data submitted from the form, are transported over the internet using XML.
- XForms is device independent because data is separated from presentation and the data model can be used for all devices.
- XForms is a W3C recommendation.

XForms Model

The XForms input data can be described in two parts:

- 1) The XForm model (to describe the data and the logic): The XForms model is used to define what the form is, what it should do and what the data it should contain.

2) The XForm user interface (to display and input the data): The XForms user interface is used to define the input field and how they should be displayed.

XHTML

XHTML stands for EXtensible HyperText Markup Language. It is the next step in the evolution of the internet. The XHTML 1.0 is the first document type in the XHTML family. ... XHTML was developed by World Wide Web Consortium (W3C) to help web developers make the transition from HTML to XML.

What is XHTML?

XHTML stands for EXtensible HyperText Markup Language

XHTML is a stricter, more XML-based version of HTML

XHTML is HTML defined as an XML application

XHTML is supported by all major browsers

Why XHTML?

XML is a markup language where all documents must be marked up correctly (be "well-formed").

XHTML was developed to make HTML more extensible and flexible to work with other data formats (such as XML). In addition, browsers ignore errors in HTML pages, and try to display the website even if it has some errors in the markup. So XHTML comes with a much stricter error handling.

XHTML - <!DOCTYPE> Is Mandatory

An XHTML document must have an XHTML <!DOCTYPE> declaration.

The <html>, <head>, <title>, and <body> elements must also be present, and the xmlns attribute in <html> must specify the xml namespace for the document.

Example

Here is an XHTML document with a minimum of required tags:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

```

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Title of document</title>
</head>
<body>

  some content here...

</body>
</html>

```

TRANSFORMATIONS:

Before learning XSLT, we should first understand XSL which stands for **Extensible Style sheet Language**. It is similar to XML as CSS is to HTML.

In case of HTML document, tags are predefined such as table ,div ,and span ;and the browser knows how to add style to them and display those using CSS styles. But in case of XML documents, tags are not predefined .In order to understand and style an XML document, An XSL document specifies how a browser should render an XML document.

Following are main parts of XSL:

XSLT—used to transform XML document in to various other types of document.

XPath —used to navigate XML document

XSLT:

An XSLT style sheet is used to define the transformation rules to be applied on the target XML document. XSLT style sheet is written in XML format. XSLT Processor takes the XSLT style sheet and applies the transformation rules on the target XML document and then it generates a formatted document in the form of XML ,HTML ,or text format .This formatted document is then utilized by XSLT formatter to generate the actual output which is to be displayed to the end-user.

Correct Style Sheet Declaration

The root element that declares the document to be an XSL style sheet is <xsl:stylesheet> or <xsl:transform>.

We want to **transform** the following XML document ("cdcatalog.xml") into XHTML:

```

<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>

```

```

    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
.
.
</catalog>

```

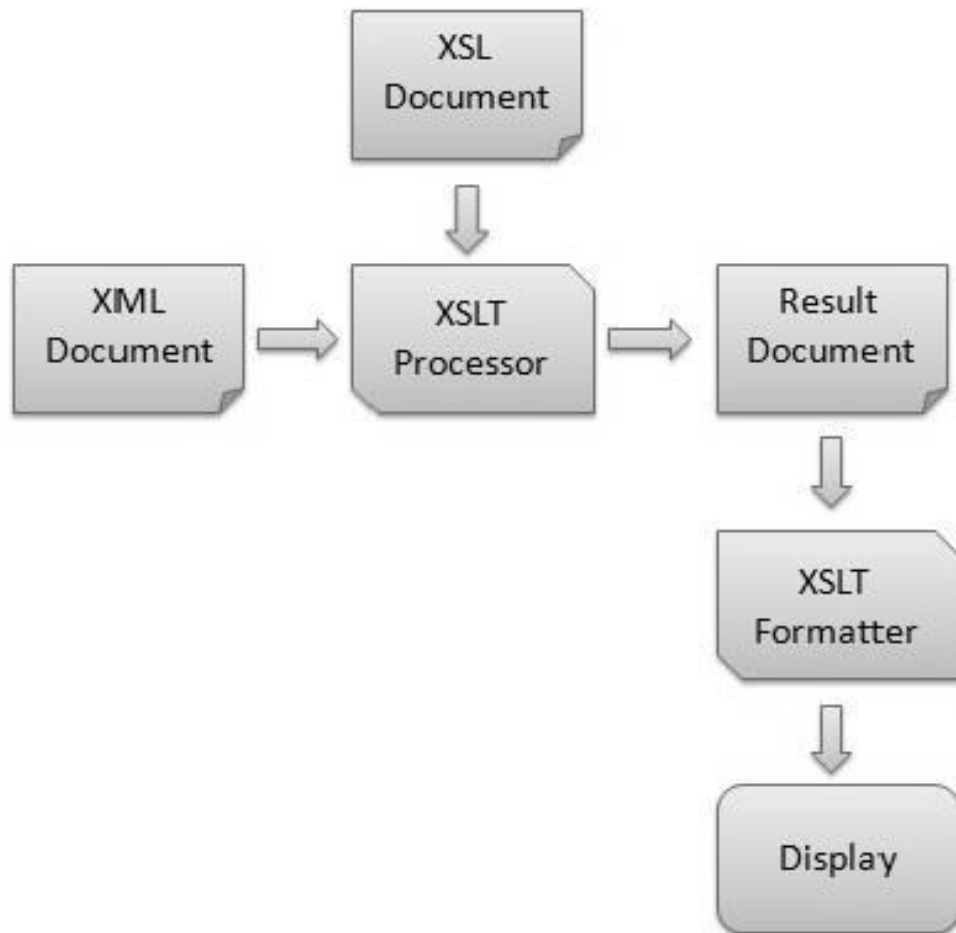


Fig. 1.21 XSLT

Create an XSL Style Sheet

Then you create an XSL Style Sheet ("cdcatalog.xml") with a transformation template:

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>

```



```

<table border="1">
  <tr bgcolor="#9acd32">
    <th>Title</th>
    <th>Artist</th>
  </tr>
  <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
  </xsl:for-each>
</table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>

```

Link the XSL Style Sheet to the XML Document

Add the XSL style sheet reference to your XML document ("cdcatalog.xml"):

```

<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="cdcatalog.xsl"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
  .
  .
</catalog>

```

The <xsl:template> Element

The <xsl:template> element is used to build templates.

The **match** attribute is used to associate a template with an XML element. The match attribute can also be used to define a template for the entire XML document. The value of the match attribute is an XPath expression (i.e. match="/" defines the whole document).

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">

```

```

<html>
<body>
<h2>My CD Collection</h2>
<table border="1">
  <tr bgcolor="#9acd32">
    <th>Title</th>
    <th>Artist</th>
  </tr>
  <tr>
    <td>.</td>
    <td>.</td>
  </tr>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>

```

Example Explained

Since an XSL style sheet is an XML document, it always begins with the XML declaration: **<?xml version="1.0" encoding="UTF-8"?>**.

The next element, **<xsl:stylesheet>**, defines that this document is an XSLT style sheet document (along with the version number and XSLT namespace attributes).

The **<xsl:template>** element defines a template. The **match="/"** attribute associates the template with the root of the XML source document.

The content inside the **<xsl:template>** element defines some HTML to write to the output.

The last two lines define the end of the template and the end of the style sheet.

The result from this example was a little disappointing, because no data was copied from the XML document to the output. In the next chapter you will learn how to use the **<xsl:value-of>** element to select values from the XML elements.

The <xsl:value-of> Element

The **<xsl:value-of>** element can be used to extract the value of an XML element and add it to the output stream of the transformation:

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>

```

```

<table border="1">
  <tr bgcolor="#9acd32">
    <th>Title</th>
    <th>Artist</th>
  </tr>
  <tr>
    <td><xsl:value-of select="catalog/cd/title"/></td>
    <td><xsl:value-of select="catalog/cd/artist"/></td>
  </tr>
</table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>

```

Example Explained

Note: The **select** attribute, in the example above, contains an XPath expression. An XPath expression works like navigating a file system; a forward slash (/) selects subdirectories.

The result from the example above was a little disappointing; only one line of data was copied from the XML document to the output. In the next chapter you will learn how to use the **<xsl:for-each>** element to loop through the XML elements, and display all of the records.

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <html>
    <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/cd">
        <tr>
          <td><xsl:value-of select="title"/></td>
          <td><xsl:value-of select="artist"/></td>
        </tr>
      </xsl:for-each>
    </table>
    </body>
    </html>
  </xsl:template>

</xsl:stylesheet>

```

The value of the **select** attribute is an XPath expression. An XPath expression works like navigating a file system; where a forward slash (/) selects subdirectories.

Where to put the Sort Information

To sort the output, simply add an `<xsl:sort>` element inside the `<xsl:for-each>` element in the XSL file:

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
      <xsl:sort select="artist"/>
      <tr>
        <td><xsl:value-of select="title"/></td>
        <td><xsl:value-of select="artist"/></td>
      </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

The **select** attribute indicates what XML element to sort on.

The `<xsl:if>` Element

To put a conditional if test against the content of the XML file, add an `<xsl:if>` element to the XSL document.

Syntax

```
<xsl:if test="expression">
  ...some output if the expression is true...
</xsl:if>
```

Where to Put the `<xsl:if>` Element

To add a conditional test, add the `<xsl:if>` element inside the `<xsl:for-each>` element in the XSL file:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
      <th>Price</th>
    </tr>
    <xsl:for-each select="catalog/cd">
      <xsl:if test="price > 10">
        <tr>
          <td><xsl:value-of select="title"/></td>
          <td><xsl:value-of select="artist"/></td>
          <td><xsl:value-of select="price"/></td>
        </tr>
      </xsl:if>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

The value of the required **test** attribute contains the expression to be evaluated.

The code above will only output the title and artist elements of the CDs that has a price that is higher than 10.

The `<xsl:choose>` Element

Syntax

```
<xsl:choose>
  <xsl:when test="expression">
    ... some output ...
  </xsl:when>
  <xsl:otherwise>
    ... some output ....
  </xsl:otherwise>
</xsl:choose>
```

The advantages of using XSLT:

Independent of programming. Transformations are
is again an XML document.

written in a separate xsl file which

Output can be altered by simply modifying the transformations in xsl file.
change any code. So Web designers can edit the stylesheet and can see the change in the output quickly.

Noneed to

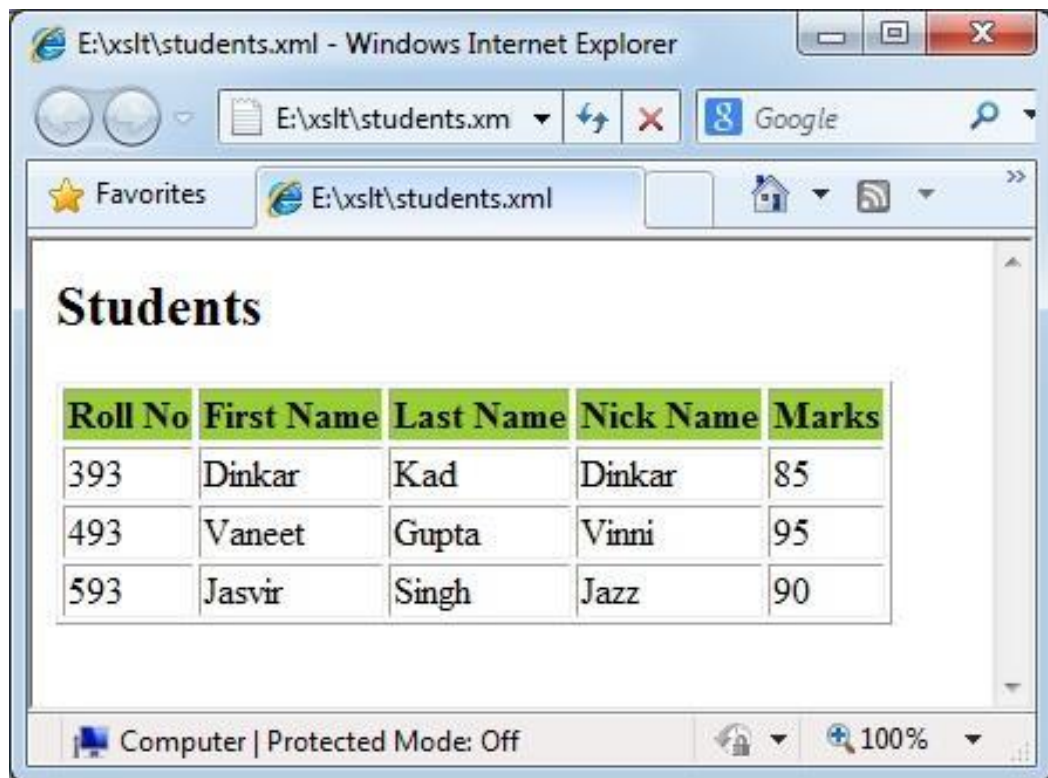
STUDENTS.XML

```
<?xml version="1.0"?>
<class>
  <student rollno="393">
    <firstname>Dinkar</firstname>
    <lastname>Kad</lastname>
    <nickname>Dinkar</nickname>
    <marks>85</marks>
  </student>
  <student rollno="493">
    <firstname>Vaneet</firstname>
    <lastname>Gupta</lastname>
    <nickname>Vinni</nickname>
    <marks>95</marks>
  </student>
  <student rollno="593">
    <firstname>Jasvir</firstname>
    <lastname>Singh</lastname>
    <nickname>Jazz</nickname>
    <marks>90</marks>
  </student>
</class>
```

We need to define an XSLT stylesheet document for the above XML document to meet the following criteria:

- Pages should have a title **Students**.

- Pageshouldhaveatableofstudentdetails.
- Columnsshouldhavefollowingheaders:RollNo,First Name,Last Name,NickName, Marks
- Tablemustcontaindetailsofthestudentsaccordingly



Roll No	First Name	Last Name	Nick Name	Marks
393	Dinkar	Kad	Dinkar	85
493	Vaneet	Gupta	Vinni	95
593	Jasvir	Singh	Jazz	90

Fig: 1.22 -Output

XPath

- Simple language consisting of path expressions
- XPath is used to address (select) parts of documents using path expressions
- A path expression is a sequence of steps separated by “/”
- Think of file names in a directory hierarchy
- Result of path expression: set of values that along with their containing elements/attributes match the specified path

XPath Standard Functions

XPath includes over 200 built-in functions.

There are functions for string values, numeric values, booleans, date and time comparison, node manipulation, sequence manipulation, and much more.

XPath expressions can also be used in JavaScript, Java, XML Schema, PHP, Python, C and C++, and lots of other languages.

XPath is Used in XSLT

- XPath is a major element in the XSLT standard.
- With XPath knowledge you will be able to take great advantage of your XSLT knowledge.

XPath Terminology

In XPath, XML documents are treated as trees of nodes. The topmost element of the tree is called the root element.

Look at the following XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<bookstore>
```

```
<book>
```

```
<title lang="en">Harry Potter</title>
```

```
<author>J K. Rowling</author>
```

```
<year>2005</year>
```

```
<price>29.99</price>
```

```
</book>
```

```
</bookstore>
```

Example of nodes in the XML document above:

<bookstore> (root element node)

<author>J K. Rowling</author> (element node)

lang="en" (attribute node)

Atomic values

Atomic values are nodes with no children or parent.

Example of atomic values:

J K. Rowling

"en"

Items

Items are atomic values or nodes.

Relationship of Nodes

Parent

Each element and attribute has one parent.

In the following example; the book element is the parent of the title, author, year, and price:

```
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

Children

Element nodes may have zero, one or more children.

In the following example; the title, author, year, and price elements are all children of the book element:

```
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

Siblings

Nodes that have the same parent.

In the following example; the title, author, year, and price elements are all siblings:

```
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

Ancestors

A node's parent, parent's parent, etc.

In the following example; the ancestors of the title element are the book element and the bookstore element:

```
<bookstore>

  <book>
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>

</bookstore>
```

Descendants

A node's children, children's children, etc.

In the following example; descendants of the bookstore element are the book, title, author, year, and price elements:

```
<bookstore>

  <book>
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
```

```
</book>
</bookstore>
```

Selecting Nodes

XPath uses path expressions to select nodes in an XML document. The node is selected by following a path or steps. The most useful path expressions are listed below:

Expression	Description
<i>nodename</i>	Selects all nodes with the name " <i>nodename</i> "
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

Table: 1.1- path expressions

Path Expression	Result
bookstore	Selects all nodes with the name "bookstore"
/bookstore	Selects the root element bookstore Note: If the path starts with a slash (/) it always represents an absolute path to an element!
bookstore/book	Selects all book elements that are children of bookstore
//book	Selects all book elements no matter where they are in the document
bookstore//book	Selects all book elements that are descendant of the bookstore element, no matter where they are under the bookstore element
//@lang	Selects all attributes that are named lang

Table: 1.2- path expressions

Selecting Unknown Nodes

XPath wildcards can be used to select unknown XML nodes.

Wildcard	Description
*	Matches any element node
@*	Matches any attribute node
node()	Matches any node of any kind

Table: 1.3 wildcards

Predicates

Predicates are used to find a specific node or a node that contains a specific value.

Predicates are always embedded in square brackets.

In the table below we have listed some path expressions with predicates and the result of the expressions:

Path Expression	Result
/bookstore/book[1]	Selects the first book element that is the child of the bookstore element. Note: In IE 5,6,7,8,9 first node is [0], but according to W3C, it is [1]. To solve this problem in IE, set the SelectionLanguage to XPath: <i>In JavaScript:</i> <code>xml.setProperty("SelectionLanguage","XPath");</code>
/bookstore/book[last()]	Selects the last book element that is the child of the bookstore element
/bookstore/book[last()-1]	Selects the last but one book element that is the child of the bookstore element
/bookstore/book[position()<3]	Selects the first two book elements that are children of the bookstore element
//title[@lang]	Selects all the title elements that have an attribute named lang
//title[@lang='en']	Selects all the title elements that have a "lang" attribute with a value of "en"
/bookstore/book[price>35.00]	Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00
/bookstore/book[price>35.00]/title	Selects all the title elements of the book elements of the bookstore element that have a price element with a value greater than 35.00

Table: 1.4 -Path Expression

XPath Axes

The XML Example Document

We will use the following XML document in the examples below.

```
<?xml version="1.0" encoding="UTF-8"?>

<bookstore>

  <book>
    <title lang="en">Harry Potter</title>
    <price>29.99</price>
  </book>

  <book>
    <title lang="en">Learning XML</title>
    <price>39.95</price>
  </book>

</bookstore>
```

Location Path Expression

A location path can be absolute or relative.

An absolute location path starts with a slash (/) and a relative location path does not. In both cases the location path consists of one or more steps, each separated by a slash:

An absolute location path:

```
/step/step/...
```

A relative location path:

```
step/step/...
```

Each step is evaluated against the nodes in the current node-set.

A step consists of:

- an axis (defines the tree-relationship between the selected nodes and the current node)
- a node-test (identifies a node within an axis)
- zero or more predicates (to further refine the selected node-set)

The syntax for a location step is:

```
axisname::nodetest[predicate]
```

Loading the XML Document

Using an XMLHttpRequest object to load XML documents is supported in all modern browsers.

```
var xmlhttp = new XMLHttpRequest();
```

Selecting Nodes

Unfortunately, there are different ways of dealing with XPath in different browsers.

Chrome, Firefox, Edge, Opera, and Safari use the evaluate() method to select nodes:

```
xmlDoc.evaluate(xpath, xmlDoc, null, XPathResult.ANY_TYPE,null);
```

Internet Explorer uses the selectNodes() method to select node:

```
xmlDoc.selectNodes(xpath);
```

In our examples we have included code that should work with most major browsers.

Select all the titles

The following example selects all the title nodes:

Example:

```
/bookstore/book/title
```

Select the title of the first book

The following example selects the title of the first book node under the bookstore element:

Example:

```
/bookstore/book[1]/title
```

Select all the prices

The following example selects the text from all the price nodes:

Example:

```
/bookstore/book/price[text()]
```

Select price nodes with price>35

The following example selects all the price nodes with a price higher than 35:

Example:

```
/bookstore/book[price>35]/price
```

Select title nodes with price>35

The following example selects all the title nodes with a price higher than 35:

Example:

```
/bookstore/book[price>35]/title
```

XQuery

- Standard language for querying XML data
- Modeled after SQL (but significantly different)
- Incorporates XPath expressions
- XQuery is supported by all major databases

XQuery can be used to:

- Extract information to use in a Web Service
- Generate summary reports
- Transform XML data to XHTML
- Search Web documents for relevant information

Xquery basic syntax rules:

- XQuery is case-sensitive
- XQuery elements, attributes, and variables must be valid XML names
- An XQuery string value can be in single or double quotes
- An XQuery variable is defined with a \$ followed by a name, e.g. \$bookstore
- XQuery comments are delimited by (: and :), e.g. (: XQuery Comment :)

XQuery is a functional, expression-oriented programming language with a simple summed up by Kilpeläinen:^[8]

All XQuery expressions operate on sequences, and evaluate to sequences. *Sequences* are ordered lists of items. *Items* can be either *nodes*, which represent components of XML documents, or *atomic values*, which are instances of XML Schema base types like xs:integer or xs:string. Sequences can also be empty, or consist of a single item only. No distinction is made between a single item and a

singleton sequence. (...) XQuery/XPath sequences differ from lists in languages by excluding nested sequences. Designers of XQuery may have considered nested sequences unnecessary for the manipulation of document contents. Nesting, or hierarchy of document structures is instead represented by nodes and their child-parent relationships

XQuery provides the means to extract and manipulate data from XML documents or any data source that can be viewed as XML, such as or office documents.

XQuery contains a superset of expression syntax to address specific parts of an XML document. It supplements this with an expression" for performing joins. A FLWOR expression is constructed from the five clauses after which it is named: FOR, LET, WHERE, ORDER BY, RETURN.

The language also provides syntax allowing new XML documents to be constructed. Where the element and attribute names are known in advance, an XML-like syntax can be used; in other cases, expressions referred to as dynamic node constructors are available. All these constructs are defined as expressions within the language, and can be arbitrarily nested.

The language is based on which uses a tree-structured model of the information content of an XML document, containing seven kinds of nodes: document nodes, elements, attributes, text nodes, comments, processing instructions, and namespaces. XDM also models all values as sequences (a singleton value is considered to be a sequence of length one). The items in a sequence can either be XML nodes or atomic values. Atomic values may be integers, strings, booleans, and so on: the full list of types is based on the primitive types

Features for updating XML documents or databases, and capability, are not part of the core language, but are defined in add-on extension standards: XQuery Update Facility supports update feature and XQuery and XPath supports full text search in XML documents.

XQuery adds support for full functional programming, in that functions are values that can be manipulated (stored in variables, passed to higher-order functions, and dynamically called).

The sample XQuery code below lists the unique speakers in each act of Shakespeare's play Hamlet, encoded in [hamlet.xml](#)

```
<html><body>
{
for $act in doc("hamlet.xml")//ACT
let $speakers := distinct-values($act//SPEAKER)
return
```

```

<div>
<h1>{ string($act/TITLE) }</h1>
<ul>
  {
  for $speaker in $speakers
  return<li>{ $speaker }</li>
  }
</ul>
</div>
}
</body></html>

```

XQuery FLWOR Expressions

FLWOR (pronounced "flower") is an acronym for "For, Let, Where, Order by, Return".

- **For** - selects a sequence of nodes
- **Let** - binds a sequence to a variable
- **Where** - filters the nodes
- **Order by** - sorts the nodes
- **Return** - what to return (gets evaluated once for every node)

How to Select Nodes From "books.xml" With FLWOR

Look at the following path expression:

```
doc("books.xml")/bookstore/book[price>30]/title
```

The expression above will select all the title elements under the book elements that are under the bookstore element that have a price element with a value that is higher than 30.

The following FLWOR expression will select exactly the same as the path expression above:

```

for $x in doc("books.xml")/bookstore/book
where $x/price>30
return $x/title

```

The result will be:

```

<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>

```

With FLWOR you can sort the result:

```

for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title

```

The **for** clause selects all book elements under the bookstore element into a variable called \$x.

The **where** clause selects only book elements with a price element with a value greater than 30.

The **order by** clause defines the sort-order. Will be sort by the title element.

The **return** clause specifies what should be returned. Here it returns the title elements.

The result of the XQuery expression above will be:

```
<title lang="en">Learning XML</title>
<title lang="en">XQuery Kick Start</title>
```

XQuery FLWOR + HTML

Present the Result In an HTML List

Look at the following XQuery FLWOR expression:

```
for $x in doc("books.xml")/bookstore/book/title
order by $x
return $x
```

The expression above will select all the title elements under the book elements that are under the bookstore element, and return the title elements in alphabetical order.

Now we want to list all the book-titles in our bookstore in an HTML list. We add and tags to the FLWOR expression:

The result of the above will be:

```
<ul>
{
for $x in doc("books.xml")/bookstore/book/title
order by $x
return <li>{data($x)}</li>
}
</ul>
```

The result will be (an HTML list):

```
<ul>
<li>Everyday Italian</li>
<li>Harry Potter</li>
<li>Learning XML</li>
<li>XQuery Kick Start</li>
</ul>
```

XQuery Terminology

Nodes

In XQuery, there are seven kinds of nodes: element, attribute, text, namespace, processing-instruction, comment, and document (root) nodes.

XML documents are treated as trees of nodes. The root of the tree is called the document node (or root node).

Look at the following XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<bookstore>
```

```
<book>
```

```
<title lang="en">Harry Potter</title>
```

```
<author>J K. Rowling</author>
```

```
<year>2005</year>
```

```
<price>29.99</price>
```

```
</book>
```

```
</bookstore>
```

Example of nodes in the XML document above:

<bookstore> (root node)

<author>J K. Rowling</author> (element node)

lang="en" (attribute node)

Atomic values

Atomic values are nodes with no children or parent.

Example of atomic values:

J K. Rowling

"en"

Items

Items are atomic values or nodes.

Relationship of Nodes

Parent

Each element and attribute has one parent.

In the following example; the book element is the parent of the title, author, year, and price:

```
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

Children

Element nodes may have zero, one or more children.

In the following example; the title, author, year, and price elements are all children of the book element:

```
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

Siblings

Nodes that have the same parent.

In the following example; the title, author, year, and price elements are all siblings:

```
<book>

<title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

Ancestors

A node's parent, parent's parent, etc.

In the following example; the ancestors of the title element are the book element and the bookstore element:

```
<bookstore>
```

```
<book>
```

```
<title>Harry Potter</title>
```

```
<author>J K. Rowling</author>
```

```
<year>2005</year>
```

```
<price>29.99</price>
```

```
</book>
```

```
</bookstore>
```

Descendants

A node's children, children's children, etc.

In the following example; descendants of the bookstore element are the book, title, author, year, and price elements:

```
<bookstore>
```

```
<book>
```

```
<title>Harry Potter</title>
```

```
<author>J K. Rowling</author>
```

```
<year>2005</year>
```

```
<price>29.99</price>
```

```
</book>
```

```
</bookstore>
```

XQuery Basic Syntax Rules

Some basic syntax rules:

- XQuery is case-sensitive
- XQuery elements, attributes, and variables must be valid XML names
- An XQuery string value can be in single or double quotes
- An XQuery variable is defined with a \$ followed by a name, e.g. \$bookstore
- XQuery comments are delimited by (: and :), e.g. (: XQuery Comment :)

XQuery Conditional Expressions

"If-Then-Else" expressions are allowed in XQuery.

Look at the following example:

```
for $x in doc("books.xml")/bookstore/book
return if ($x/@category="children")
```

```
then <child>{ data($x/title) }</child>
else <adult>{ data($x/title) }</adult>
```

Notes on the "if-then-else" syntax: parentheses around the if expression are required. else is required, but it can be just else ().

The result of the example above will be:

```
<adult>Everyday Italian</adult>
<child>Harry Potter</child>
<adult>XQuery Kick Start</adult>
<adult>Learning XML</adult>
```

XQuery Comparisons

In XQuery there are two ways of comparing values.

1. General comparisons: =, !=, <, <=, >, >=
2. Value comparisons: eq, ne, lt, le, gt, ge

The difference between the two comparison methods are shown below.

The following expression returns true if any q attributes have a value greater than 10:

```
$bookstore//book/@q > 10
```

The following expression returns true if there is only one q attribute returned by the expression, and its value is greater than 10. If more than one q is returned, an error occurs:

```
$bookstore//book/@q gt 10
```

Adding Elements and Attributes to the Result

As we have seen in a previous chapter, we may include elements and attributes from the input document ("books.xml") in the result:

```
for $x in doc("books.xml")/bookstore/book/title
order by $x
return $x
```

The XQuery expression above will include both the title element and the lang attribute in the result, like this:

```
<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">Learning XML</title>
<title lang="en">XQuery Kick Start</title>
```

The XQuery expression above returns the title elements the exact same way as they are described in the input document.

We now want to add our own elements and attributes to the result!

Add HTML Elements and Text

Now, we want to add some HTML elements to the result. We will put the result in an HTML list - together with some text:

```
<html>
<body>

<h1>Bookstore</h1>

<ul>
{
for $x in doc("books.xml")/bookstore/book
order by $x/title
return <li>{data($x/title)}. Category: {data($x/@category)}</li>
}
</ul>

</body>
</html>
```

The XQuery expression above will generate the following result:

```
<html>
<body>
<h1>Bookstore</h1>
<ul>
<li>Everyday Italian. Category: COOKING</li>
<li>Harry Potter. Category: CHILDREN</li>
<li>Learning XML. Category: WEB</li>
<li>XQuery Kick Start. Category: WEB</li>
</ul>
</body>
</html>
```

Add Attributes to HTML Elements

Next, we want to use the category attribute as a class attribute in the HTML list:


```

<html>
<body>
<h1>Bookstore</h1>
<ul>
{
for $x in doc("books.xml")/bookstore/book
order by $x/title
return <li class="{ data($x/@category)}">{ data($x/title)}</li>
}
</ul>
</body>
</html>

```

The XQuery expression above will generate the following result:

```

<html>
<body>
<h1>Bookstore</h1>
<ul>
<li class="COOKING">Everyday Italian</li>
<li class="CHILDREN">Harry Potter</li>
<li class="WEB">Learning XML</li>
<li class="WEB">XQuery Kick Start</li>
</ul>
</body>
</html>

```

Selecting and Filtering Elements

As we have seen in the previous chapters, we are selecting and filtering elements with either a Path expression or with a FLWOR expression.

Look at the following FLWOR expression:

```

for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title

```

- for - (optional) binds a variable to each item returned by the in expression
- let - (optional)
- where - (optional) specifies a criteria
- order by - (optional) specifies the sort-order of the result
- return - specifies what to return in the result

The for Clause

The for clause binds a variable to each item returned by the in expression. The for clause results in iteration. There can be multiple for clauses in the same FLWOR expression.

To loop a specific number of times in a for clause, you may use the **to** keyword:

```
for $x in (1 to 5)
return <test>{$x}</test>
```

Result:

```
<test>1</test>
<test>2</test>
<test>3</test>
<test>4</test>
<test>5</test>
```

The **at** keyword can be used to count the iteration:

```
for $x at $i in doc("books.xml")/bookstore/book/title
return <book>{$i}. {data($x)}</book>
```

Result:

```
<book>1. Everyday Italian</book>
<book>2. Harry Potter</book>
<book>3. XQuery Kick Start</book>
<book>4. Learning XML</book>
```

It is also allowed with more than one in expression in the for clause. Use comma to separate each in expression:

```
for $x in (10,20), $y in (100,200)
return <test>x={$x} and y={$y}</test>
```

Result:

```
<test>x=10 and y=100</test>
<test>x=10 and y=200</test>
<test>x=20 and y=100</test>
<test>x=20 and y=200</test>
```

The let Clause

The let clause allows variable assignments and it avoids repeating the same expression many times. The let clause does not result in iteration.

```
let $x := (1 to 5)
return <test>{$x}</test>
```

Result:

```
<test>1 2 3 4 5</test>
```

The where Clause

The where clause is used to specify one or more criteria for the result:

```
where $x/price>30 and $x/price<100
```

The order by Clause

The order by clause is used to specify the sort order of the result. Here we want to order the result by category and title:

```
for $x in doc("books.xml")/bookstore/book
order by $x/@category, $x/title
return $x/title
```

Result:

```
<title lang="en">Harry Potter</title>
<title lang="en">Everyday Italian</title>
<title lang="en">Learning XML</title>
<title lang="en">XQuery Kick Start</title>
```

The return Clause

The return clause specifies what is to be returned.

```
for $x in doc("books.xml")/bookstore/book
return $x/title
```

Result:

```
<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```

XQuery Functions

XQuery is built on XPath expressions. XQuery 1.0 and XPath 2.0 share the same data model and support the same functions and operators.

[XPath Operators](#)

[XPath Functions](#)

You can also define your own functions in XQuery.

XQuery Data Types

XQuery shares the same data types as XML Schema 1.0 (XSD).

[XSD String](#)

[XSD Date](#)

[XSD Numeric](#)

[XSD Misc](#)

Examples of Function Calls

A call to a function can appear where an expression may appear. Look at the examples below:

Example 1: In an element

```
<name>{upper-case($booktitle)}</name>
```

Example 2: In the predicate of a path expression

```
doc("books.xml")/bookstore/book[substring(title,1,5)='Harry']
```

Example 3: In a let clause

```
let $name := (substring($booktitle,1,4))
```

XQuery User-Defined Functions

If you cannot find the XQuery function you need, you can write your own.

User-defined functions can be defined in the query or in a separate library.

Syntax

```
declare function prefix:function_name($parameter as datatype)  
as returnDatatype  
{  
  ...function code here...  
};
```

Notes on user-defined functions:

- Use the declare function keyword
- The name of the function must be prefixed

- The data type of the parameters are mostly the same as the data types defined in XML Schema
- The body of the function must be surrounded by curly braces

Example of a User-defined Function Declared in the Query

```
declare function local:minPrice($p as xs:decimal?, $d as xs:decimal?)
as xs:decimal?
{
let $disc := ($p * $d) div 100
return ($p - $disc)
};
```

Below is an example of how to call the function above:

```
<minPrice>{local:minPrice($book/price,$book/discount)}</minPrice>
```

XLink

- XLink is used to create hyperlinks within XML documents
- Any element in an XML document can behave as a link
- With XLink, the links can be defined outside the linked files

XLink Syntax

- In HTML, the <a> element defines a hyperlink. However, this is not how it works in XML. In XML documents, you can use whatever element names you want - therefore it is impossible for browsers to predict what link elements will be called in XML documents.
- An example of how to use XLink to create links in an XML document:
- ```
<?xml version="1.0" encoding="UTF-8"?>
<homepages xmlns:xlink="http://www.google.com">
 <homepage xlink:type="simple" xlink:href="https://www.yahoo.com">Visit yahoo home
page</homepage>
 <homepage xlink:type="simple" xlink:href="http://www.sathyabamauniversity.com">Vis
it sathyabama university</homepage>
</homepages>
```
- To get access to the XLink features we must declare the XLink namespace. The XLink namespace is: "http://www.google.com".
- The xlink:type and the xlink:href attributes in the <homepage> elements come from the XLink namespace.
- The xlink:type="simple" creates a simple "HTML-like" link (means "click here to go there").
- The xlink:href attribute specifies the URL to link to.

## **XLink Example**

The following XML document contains XLink features:

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore xmlns:xlink="http://www.w3.org/1999/xlink">
<book title="Harry Potter">
 <description
 xlink:type="simple"
 xlink:href="/images/HPotter.gif"
 xlink:show="new">
 As his fifth year at Hogwarts School of Witchcraft and
 Wizardry approaches, 15-year-old Harry Potter is.....
 </description>
</book>
<book title="XQuery Kick Start">
 <description
 xlink:type="simple"
 xlink:href="/images/XQuery.gif"
 xlink:show="new">
 XQuery Kick Start delivers a concise introduction
 to the XQuery standard.....
 </description>
</book>
</bookstore>
```

### **Example explained:**

- The XLink namespace is declared at the top of the document (xmlns:xlink="http://www.w3.org/1999/xlink")
- The xlink:type="simple" creates a simple "HTML-like" link
- The xlink:href attribute specifies the URL to link to (in this case - an image)
- The xlink:show="new" specifies that the link should open in a new window

## **XLink - Going Further**

In the example above we have demonstrated simple XLinks. XLink is getting more interesting when accessing remote locations as resources, instead of standalone pages.

If we set the value of the xlink:show attribute to "embed", the linked resource should be processed inline within the page. When you consider that this could be another XML document you could, for example, build a hierarchy of XML documents.

## **ResponsiveWebDesign(RWD)**

Responsive web design makes your web page look good on all devices.

Responsive web design uses only HTML and CSS. Web pages can be viewed using many different devices: desktops, tablets, and phones. Your web page should look good, and be

easy to use, regardless of the device. Web pages should not leave out information to fit smaller devices, but rather adapt its content to fit any device

## Setting The Viewport

To create a responsive website, add the following <meta> tag to all your web pages:

Example:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

This will set the viewport of your page, which will give the browser instructions on how to control the page's dimensions and scaling.

## Responsive Images

Responsive images are images that scale nicely to fit any browser size.

Using the width Property

If the CSS width property is set to 100%, the image will be responsive and scale up and down:

Example

```

```

In the example above, the image can be scaled up to be larger than its original size. A better solution, in many cases, will be to use the max-width property instead.

Using the max-width Property

If the max-width property is set to 100%, the image will scale down if it has to, but never scale up to be larger than its original size:

Example

```

```

Show Different Images Depending on Browser Width

The HTML <picture> element allows you to define different images for different browser window sizes.

Resize the browser window to see how the image below change depending on the width:

```
<picture>
```

```

<source srcset="img_smallflower.jpg" media="(max-width: 600px)">
<source srcset="img_flowers.jpg" media="(max-width: 1500px)">
<source srcset="flowers.jpg">

</picture>

```

## Responsive Text Size

The text size can be set with a "vw" unit, which means the "viewport width".

That way the text size will follow the size of the browser window:

Example

```
<h1 style="font-size:10vw">Hello World</h1>
```

## Media Queries

In addition to resize text and images, it is also common to use media queries in responsive web pages.

With media queries you can define completely different styles for different browser sizes.

Example: resize the browser window to see that the three div elements below will display horizontally on large screens and stacked vertically on small screens:

Example

```

<style>
.left, .right {
 float: left;
 width: 20%; /* The width is 20%, by default */
}
.main {
 float: left;
 width: 60%; /* The width is 60%, by default */
}

/* Use a media query to add a breakpoint at 800px: */
@media screen and (max-width: 800px) {
 .left, .main, .right {
 width: 100%; /* The width is 100%, when the viewport is 800px or smaller */
 }
}

```



```
}
}
</style>
```

### **Responsive Web Design - The Viewport**

The viewport is the user's visible area of a web page. The viewport varies with the device, and will be smaller on a mobile phone than on a computer screen. Before tablets and mobile phones, web pages were designed only for computer screens, and it was common for web pages to have a static design and a fixed size. Then, when we started surfing the internet using tablets and mobile phones, fixed size web pages were too large to fit the viewport. To fix this, browsers on those devices scaled down the entire web page to fit the screen

### **Responsive Web Design - Grid-View**

Many web pages are based on a grid-view, which means that the page is divided into columns. Using a grid-view is very helpful when designing web pages. It makes it easier to place elements on the page. A responsive grid-view often has 12 columns, and has a total width of 100%, and will shrink and expand as you resize the browser window.

### **Responsive Web Design - Media Queries**

Media query is a CSS technique introduced in CSS3. If the browser window is 600px or smaller, the background color will be lightblue.

```
@media only screen and (max-width: 600px)
{
 body
{
 background-color: lightblue;
 }
}
```



**Fig :1.24 -output**

```
img {
 width: 100%;
 height: auto;
}
```

Notice that in the example above, the image can be scaled up to be larger than its original size.

## **CSS Introduction**

Cascading Style Sheets, fondly referred to as CSS, is a simple design language intended to simplify the process of making web pages presentable. CSS handles the look and feel part of a web page. Using CSS, you can control the color of the text, the style of fonts, the spacing between paragraphs, how columns are sized and laid out, what background images or colors are used, as well as a variety of other effects. CSS is easy to learn and understand but it provides a powerful control over the presentation of an HTML document. Most commonly, CSS is combined with the markup languages HTML or XHTML.

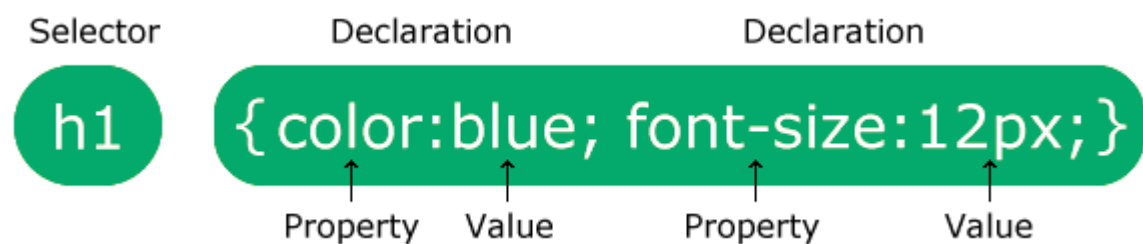
## **Advantages of CSS**

- CSS saves time - You can write CSS once and then reuse the same sheet in multiple HTML pages. You can define a style for each HTML element and apply it to as many web pages as you want.
- Pages load faster - If you are using CSS, you do not need to write HTML tag attributes

every time. Just write one CSS rule of a tag and apply it to all the occurrences of that tag. So, less code means faster download times.

- Easy maintenance - To make a global change, simply change the style, and all the elements in all the web pages will be updated automatically.
- Superior styles to HTML - CSS has a much wider array of attributes than HTML, so you can give a far better look to your HTML page in comparison to HTML attributes.
- Multiple Device Compatibility - Style sheets allow content to be optimized for more than one type of device. By using the same HTML document, different versions of a website can be presented for handheld devices such as PDAs and cellphones or for printing.
- Global web standards – Now HTML attributes are being deprecated and it is being recommended to use CSS. So it's a good idea to start using CSS in all the HTML pages to make them compatible with future browsers.

### CSS Syntax:



The declaration block contains one or more declarations separated by semicolons. Each declaration includes a CSS property name and a value, separated by a colon. Multiple CSS declarations are separated with semicolons, and declaration blocks are surrounded by curly braces.

A CSS comprises of style rules that are interpreted by the browser and then applied to the corresponding elements in your document. A style rule is made of three parts:

- **Selector:** A selector is an HTML tag at which a style will be applied. This could be any tag like etc.
- **Property:** A property is a type of attribute of HTML tag. Put simply, all the HTML attributes are converted into CSS properties. They could be color, border, etc.
- **Value:** Values are assigned to properties.

### SELECTORS:

The **class selector** selects HTML elements with a specific class attribute. To select elements with a specific class, write a period (.) character, followed by the class name.

### Example:

```
center
{
text-align: center;
color: red;
}
```

The **grouping selector** selects all the HTML elements with the same style definitions. Look at the following CSS code (the h1, h2, and p elements have the same style definitions):

### Example

```
h1 {
text-align: center;
color: red;
}
```

```
h2 {
text-align: center;
color: red;
}
```

```
p {
text-align: center;
color: red;
}
```

It will be better to group the selectors, to minimize the code. To group selectors, separate each selector with a comma.

```
h1,h2,p
{
text-align: center;
color: red;
}
```

### The Type Selectors

This is the same selector we have seen above. Again, one more example to give a color to all level 1 headings:

```
h1
{
color: #36CFFF;
}
```

### The Universal Selectors

Rather than selecting elements of a specific type, the universal selector quite simply matches the name of any element type:

```
* {
```

```
color: #000000;
}
```

This rule renders the content of every element in our document in black.

### The Descendant Selectors

Suppose you want to apply a style rule to a particular element only when it lies inside a particular element. As given in the following example, the style rule will apply to *element only when it lies inside the tag*.

```
ul em {
color: #000000;
}
```

### The Class Selectors

You can define style rules based on the class attribute of the elements. All the elements having that class will be formatted according to the defined rule

```
.black {
color: #000000;
}
```

This rule renders the content in black for every element with class attribute set to black in our document. You can make it a bit more particular. For example:

```
h1.black {
color: #000000;
}
```

### The ID Selectors

You can define style rules based on the id attribute of the elements. All the elements having that id will be formatted according to the defined rule.

```
#black {
color: #000000;
}
```

### The ID Selectors

You can define style rules based on the id attribute of the elements. All the elements having that id will be formatted according to the defined rule.

```
#black {
color: #000000;
}
```

This rule renders the content in black for every element with id attribute set to black in our document. You can make it a bit more particular. For example:

```
h1#black {
```

```
color: #000000;
}
```

This rule renders the content in black for only elements with id attribute set to black. The true power of id selectors is when they are used as the foundation for descendant selectors.

For example

```
#black h2 {
 color: #000000;
}
```

In this example, all level 2 headings will be displayed in black color when those headings will lie within tags having id attribute set to black.

### The Child Selectors

You have seen the descendant selectors. There is one more type of selector, which is very similar to descendants but have different functionality. Consider the following example:


```
body > p {
 color: #000000;
}
```

This rule will render all the paragraphs in black if they are a direct child of the element. Other paragraphs put inside other elements like or would not have any effect of this rule.

### The Attribute Selectors

You can also apply styles to HTML elements with particular attributes. The style rule below will match all the input elements having a type attribute with a value of text:

```
input[type="text"]{
 color: #000000;
}
```

The advantage to this method is that the  element is unaffected, and the color applied only to the desired text fields. There are following rules applied to attribute selector.

- `p[lang]` - Selects all paragraph elements with a lang attribute.
- `p[lang="fr"]` - Selects all paragraph elements whose lang attribute has a value of exactly "fr".
- `p[lang~="fr"]` - Selects all paragraph elements whose lang attribute contains the word "fr".
- `p[lang|="en"]` - Selects all paragraph elements whose lang attribute contains values that are exactly "en", or begin with "en-".

### Multiple Style Rules

You may need to define multiple style rules for a single element. You can define these rules to combine multiple properties and corresponding values into a single block as defined in the following example:

```
h1 {
```

```

color: #36C;
font-weight: normal;
letter-spacing: .4em;
margin-bottom: 1em;
text-transform: lowercase;
}

```

Here all the property and value pairs are separated by a semicolon (;). You can keep them in a single line or multiple lines. For better readability, we keep them in separate lines. For a while, don't bother about the properties mentioned in the above block. These properties will be explained in the coming chapters and you can find the complete detail about properties in CSS References.

### Grouping Selectors

You can apply a style to many selectors if you like. Just separate the selectors with a comma, as given in the following example:

```

h1, h2, h3 {
color: #36C;
font-weight: normal;
letter-spacing: .4em;
margin-bottom: 1em;
text-transform: lowercase;
}

```

This define style rule will be applicable to h1, h2 and h3 element as well. The order of the list is irrelevant. All the elements in the selector will have the corresponding declarations applied to them.

You can combine the various class selectors together as shown below:

```

#content,
#footer,
#supplement
{
position: absolute;
left: 510px;
width: 200px;
}

```

Basic CSS rule syntax

```

selector {
property: value;
property: value;
...
property: value;
}

```

```

p {
font-family: sans-serif;
color: red;
}

```

}

a CSS file consists of one or more rules . Each rule starts with a selector that specifies an HTML element(s) and then applies style properties to them a selector of \* selects all elements

### **Types of style sheet:**

There are three types of style sheet:

- External CSS
- Internal CSS
- Inline CSS

### **External CSS**

a CSS file consists of one or more rules each rule starts with a selector that specifies an HTML element(s) and then applies style properties to them a selector of \* selects all element

a page can link to multiple style sheet files in case of a conflict (two sheets define a style for the same HTML element), the latter sheet's properties will be used.

CSS code can be embedded within the head of an HTML page this is bad style and should be avoided when possible.

### **Example**

External styles are defined within the <link> element, inside the <head> section of an HTML page:

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" href="mystyle.css">
</head>
<body>
<h1>Thisisaheading</h1>
<p>Thisisaparagraph</p>
</body>
</html>
```

An external style sheet can be written in any text editor, and must be saved with a .css extension.

### **External CSS**

An external style sheet can be written in any text editor, and must be saved with a .css extension.

### **Example**



External styles are defined within the <link> element, inside the <head> section of an HTML page

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" href="mystyle.css">
</head>
<body>
<h1>Thisisaheading</h1>
<p>Thisisaparagraph</p>
</body>
</html>
```

### **Internal CSS**

An internal style sheet may be used if one single HTML page has a unique style.

Example

Internal styles are defined within the <style> element, inside the <head> section of an HTML page

```
<!DOCTYPE html>
<html>
<head>
<style>
body
{
background-color: linen;
}

h1 {
color: maroon;
margin-left: 40px;
}
</style>
</head>
<body>
<h1>Thisisaheading</h1>
<p>Thisisaparagraph.</p>
</body>
</html>
```

### **Inline CSS**

To use inline styles, add the style attribute to the relevant element. The style attribute can contain

any CSS property.

Example

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h1 style="color:blue;text-align:center;">Thisisaheading</h1>
```

```
<p style="color:red;">Thisisaparagraph.</p>
```

```
</body>
```

```
</html>
```

### QUESTION BANK

S.No	Questions (2 marks)	Competence	BTLevel
1.	Write the HTMLcoding to print “WELCOME”.	Create	BTL6
2.	Classify <LINK>tag? Give an example.	Analyze	BTL4
3.	Write some advantages of using XML	Create	BTL6
4.	List out the <FORM >tag attributes.	Knowledge	BTL1
5.	Differentiate the HTML and XML.	Analyze	BTL4
6.	DefineXML schemas.	Knowledge	BTL1
7.	Define XML parsers.	Knowledge	BTL1
8.	Write short note on DOM.	Create	BTL6
9.	Define SAX.	Knowledge	BTL1
10.	Write the syntax to create an XSL.	Create	BTL6
11.	Define Xforms and Xhtml.	Knowledge	BTL1
12.	Define XSLT.	Knowledge	BTL1
13.	Write short note on XLINK.	Create	BTL6
14.	Define XPATH.	Knowledge	BTL1
15.	Write short note on XQuery.	Create	BTL6

S.No	Questions(16 marks)	Competence	BTLevel
1.	Define XML? Explain how to write an XML document? What are the goals of XML? Clearly explain the XML Schema and XML parsing in detail.	Understand	BTL2
2.	Explain the following HTML tags with all attributes. (1) <a>(2)<body> (3)<img>(4)<table> (5) <p>	Understand	BTL2
3.	Write and explain tags to create following HTML form elements with their attributes. (i). textbox (ii). Dropdown list (iii). Password field (iv). Checkbox (v). radio button	Create	BTL6
4.	Define Cascading Style Sheet? Explain various types of Style Sheets with example.	Understand	BTL2
5.	Create an HTML code which creates form to collect user information like name, age, email, phone no. And address.	Create	BTL6
6.	Write the CSS code to insert the image of the butterfly with name butterfly.gif as a background image for a web page.	Create	BTL6
7.	Create the CSS code to display the text of td tag in centre.	Create	BTL6
8.	Summarize the following CSS code: h1 {font-family: arial, comic sans-serif, "Times New Roman";} Explain font-family.	Evaluate	BTL5
9.	Explain the importance of DOM component of DHTML.	Understand	BTL2
10.	Explain the Responsive web design (RWD).	Understand	BTL2



**SATHYABAMA**

INSTITUTE OF SCIENCE AND TECHNOLOGY  
(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

[www.sathyabama.ac.in](http://www.sathyabama.ac.in)

---

## UNIT – II – Customer Interface Design and Development – SITA1502

## UNIT 2 CLIENT-SIDE SCRIPTING

Java Script – Advantages – Data types – Variables – Operators – Control statements – Functions – Objects and arrays – Windows and frames – Forms. AJAX – XMLHttpRequest (XHR) – Create Object – Request – Response – Ready state.

### What is JavaScript?

Javascript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

JavaScript was first known as LiveScript, but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name LiveScript. The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers.

The ECMA-262 Specification defined a standard version of the core JavaScript language.

- JavaScript is a lightweight, interpreted programming language.
- Designed for creating network-centric applications.
- Complementary to and integrated with Java.
- Complementary to and integrated with HTML.
- Open and cross-platform.

### Client-Side JavaScript

- Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.
- It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.
- The JavaScript client-side mechanism provides many advantages over traditional CGI server-side scripts. For example, you might use JavaScript to check if the user has entered a valid e-mail address in a form field.
- The JavaScript code is executed when the user submits the form, and only if all the entries are valid, they would be submitted to the Web Server.
- JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly.

### Advantages of JavaScript

The merits of using JavaScript are:

- **Less server interaction:** You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.

- **Immediate feedback to the visitors:** They don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity:** You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- **Richer interfaces:** You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

## Limitations of JavaScript

We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features:

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
  - JavaScript doesn't have any multithreading or multiprocessor capabilities.

Once again, JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTML pages.

## JavaScript Development Tools

One of major strengths of JavaScript is that it does not require expensive development tools. You can start with a simple text editor such as Notepad. Since it is an interpreted language inside the context of a web browser, you don't even need to buy a compiler.

To make our life simpler, various vendors have come up with very nice JavaScript editing tools. Some of them are listed here:

- **Microsoft FrontPage:** Microsoft has developed a popular HTML editor called FrontPage. FrontPage also provides web developers with a number of JavaScript tools to assist in the creation of interactive websites.
- **Macromedia Dreamweaver MX:** Macromedia Dreamweaver MX is a very popular HTML and JavaScript editor in the professional web development crowd. It provides several handy prebuilt JavaScript components, integrates well with databases, and conforms to new standards such as XHTML and XML.
- **Macromedia HomeSite 5:** HomeSite 5 is a well-liked HTML and JavaScript editor from Macromedia that can be used to manage personal websites effectively.

## SYNTAX

- JavaScript can be implemented using JavaScript statements that are placed within the `<script>...</script>` HTML tags in a web page.
- You can place the `<script>` tags, containing your JavaScript, anywhere within your web page, but it is normally recommended that you should keep it within the `<head>` tags.
- The `<script>` tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

```
<script ...>
 JavaScript
code
</script>
```

The script tag takes two important attributes:

- **Language:** This attribute specifies what scripting language you are using. Typically, its value will be javascript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.
- **Type:** This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".

So your JavaScript syntax will look as follows.

```
<script language="javascript"
type="text/javascript">
 JavaScript code
</script>
```

## First JavaScriptCode

Let us take a sample example to print out "Hello World". We added an optional HTML comment that surrounds our JavaScript code. This is to save our code from a browser that does not support JavaScript. The comment ends with a "`!-->`". Here "`/*`" signifies a comment in JavaScript, so we add that to prevent a browser from reading the end of the HTML comment as a piece of JavaScript code. Next, we call a function `document.write` which writes a string into our HTML document.

This function can be used to write text, HTML, or both. Take a look at the following code.

```
<html>
<body>
 <script language="javascript"
 type="text/javascript">
 <!--
 document.write ("Hello World!")
 //-->
 </script>
</body>
</html>
```

This code will produce the following result: **Hello World!**

## Whitespace and Line Breaks

JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs. You can use spaces, tabs, and newlines freely in your program and you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

## Semicolons are Optional

Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java. JavaScript, however, allows you to omit this semicolon if each of your statements are placed on a separate line. For example, the following code could be written without semicolons.

```
<script language="javascript" type="text/javascript">
<!--
 var1 = 10
 var2 = 20
//-->
</script>
```

But when formatted in a single line as follows, you must use semicolons:

```
<script language="javascript" type="text/javascript">
<!--
 var1 = 10; var2 = 20;
//-->
</script>
```

## Case Sensitivity

JavaScript is a case-sensitive language. This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters. So the identifiers `Time` and `TIME` will convey different meanings in JavaScript.

## Comments in JavaScript

JavaScript supports both C-style and C++-style comments. Thus:

- Any text between a `//` and the end of a line is treated as a comment and is ignored by JavaScript.
- Any text between the characters `/*` and `*/` is treated as a comment. This may span multiple lines.



- JavaScript also recognizes the HTML comment opening sequence is not recognized by JavaScript so it should be written as `!-->`.
- The HTML comment closing sequence `-->` is not recognized by JavaScript so it should be written as `!-->`.

The following example shows how to use comments in JavaScript.

```
<script language="javascript"
type="text/javascript">

<!--

// This is a comment. It is similar to comments in
C++

/*

* This is a multiline comment in JavaScript

* It is very similar to comments in C
Programming

*/

!-->

</script
```

## **PLACEMENT**

There is a flexibility given to include JavaScript code anywhere in an HTML document. However the most preferred ways to include JavaScript in an HTML file are as follows:

- Script in `<head>...</head>` section.
- Script in `<body>...</body>` section.
- Script in `<body>...</body>` and `<head>...</head>` sections.
- Script in an external file and then include in `<head>...</head>` section.

In the following section, we will see how we can place JavaScript in an HTML file in different ways.

### **JavaScript in `<head>...</head>` Section**

If you want to have a script run on some event, such as when a user clicks somewhere, then you will place that script in the head as follows.

```
html>
<head>
<script type="text/javascript">
<!--
function sayHello() {
 alert("Hello World")
}
//-->
</script>
</head>
<body>
Click here for the result
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```

Click here for the result

Say Hello

### JavaScript in <body>...</body> Section

If you need a script to run as the page loads so that the script generates content in the page, then the script goes in the <body> portion of the document. In this case, you would not have any function defined using JavaScript. Take a look at the following code.

```
<html>
<head>
</head>
<body>
<script type="text/javascript">
<!--
document.write("Hello
World")
//-->
</script>
<p>This is web page body </p>
</body>
</html>
Hello World
This is web page body
```

### JavaScript in <body> and <head> Sections

You can put your JavaScript code in <head> and <body> section altogether as Follows.

```
<html>
<head>
<script type="text/javascript">
<!--
function sayHello()
{
 alert("Hello World")
}
//-->
</script>
</head>
<body>
<script type="text/javascript">
<!--
document.write("Hello World")
//-->
</script>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
Click here for the result
Say Hello
```

## JavaScript in External File

As you begin to work more extensively with JavaScript, you will be likely to find that there are cases where you are reusing identical JavaScript code on multiple pages of a site.

You are not restricted to be maintaining identical code in multiple HTML files. The script tag provides a mechanism to allow you to store JavaScript in an external file and then include it into your HTML files.

Here is an example to show how you can include an external JavaScript file in your HTML code using script tag and its src attribute.

```
<html>
<head>
<script type="text/javascript" src="filename.js" ></script>
</head>
<body>
.....
</body>
```

**</html>**

To use JavaScript from an external file source, you need to write all your JavaScript source code in a simple text file with the extension ".js" and then include that file as shown above.

For example, you can keep the following content in **filename.js** file and then you can use **sayHello** function in your HTML file after including the filename.js file.

```
function sayHello()
{

 alert("Hello World")

}
```

## **JavaScript Datatypes**

One of the most fundamental characteristics of a programming language is the set of data types it supports. These are the type of values that can be represented and manipulated in a programming language.

JavaScript allows you to work with three primitive data types:

- Numbers, e.g., 123, 120.50 etc.
- Strings of text, e.g. "This text string" etc.
- Boolean, e.g. true or false.

JavaScript also defines two trivial data types, null and undefined, each of which defines only a single value. In addition to these primitive data types, JavaScript supports a composite data type known as object. We will cover objects in detail in a separate chapter.

## **JavaScript Variables**

Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.

Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the var keyword as follows.

```
<script type="text/javascript">

<!--

var money;

var name;

//-->

</script>
```

You can also declare multiple variables with the same var keyword as follows:

```
<script type="text/javascript">
<!--
var money, name;
//-->
</script>
```

Storing a value in a variable is called variable initialization. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.

For instance, you might create a variable named money and assign the value 2000.50 to it later. For another variable, you can assign a value at the time of initialization as follows.

```
<script type="text/javascript">
<!--
var name = "Ali";
var money;
money = 2000.50;
//-->
</script>
```

JavaScript is untyped language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

### JavaScript Variable Scope

The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.

- **Global Variables:** A global variable has global scope which means it can be defined anywhere in your JavaScript code.
- **Local Variables:** A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

Within the body of a function, a local variable takes precedence over a global variable with the same name. If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable. Take a look into the following example.

```
<script type="text/javascript">
```

```

<!--
var myVar = "global"; // Declare a global variable

function checkscope() {

 var myVar = "local"; // Declare a local variable

 document.write(myVar);

}

//-->
</script>

```

Result: Local

## JavaScript Variable Names

While naming your variables in JavaScript, keep the following rules in mind.

- You should not use any of the JavaScript reserved keywords as a variable name. These keywords are mentioned in the next section. For example, break or boolean variable names are not valid.
- JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For example, 123test is an invalid variable name but \_123test is a valid one.
- JavaScript variable names are case-sensitive. For example, Name and name are two different variables.

## JavaScript Reserved Words

A list of all the reserved words in JavaScript are given in the following table. They cannot be used as JavaScript variables, functions, methods, loop labels, or any object names.

**Fig. 2.1 Keywords**

Reserved Words in JavaScript			
abstract	else	instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	false	native	throws
catch	final	new	transient
char	finally	null	true
class	float	package	try
const	for	private	typeof
continue	function	protected	var
debugger	goto	public	void
default	if	return	volatile
delete	implements	short	while
do	import	static	with
double	in	super	

## OPERATORS

### What is an Operator?

Let us take a simple expression  $4 + 5$  is equal to 9. Here 4 and 5 are called operands and '+' is called the operator. JavaScript supports the following types of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

### Arithmetic Operators

```
<html>

<body>

<script type="text/javascript">

<!--

var a = 33;

var b = 10;

var c = "Test";

var linebreak = "
";

document.write("a + b = ");

result = a + b;

document.write(result);

document.write(linebreak);

document.write("a - b = ");

result = a - b;

document.write(result);

document.write(linebreak);

document.write("a / b = ");

result = a / b;

document.write(result);

document.write(linebreak);

document.write("a % b = ");
```

```
result = a % b;
document.write(result);
document.write(linebreak);
document.write("a + b + c = ");
result = a + b + c;
document.write(result);
document.write(linebreak);
a = a++;
document.write("a++ = ");
result = a++;
document.write(result);
document.write(linebreak);
b = b--;
document.write("b-- = ");
result = b--;
document.write(result);
document.write(linebreak);
//-->
</script>
<p>Set the variables to different values and then try...</p>
</body>
</html>
```

## Output

```
a + b = 43
a - b = 23
a / b = 3.3
a % b = 3
a + b + c = 43Test
a++ = 33
b-- = 10
```



## Comparison Operators

```
<html>

<body>

<script type="text/javascript">

<!--

var a = 10;

var b = 20;

var linebreak = "
";

document.write("(a == b) => ");

result = (a == b);

document.write(result);

document.write(linebreak);

document.write("(a < b) => ");

result = (a < b);

document.write(result);

document.write(linebreak);

document.write("(a > b) => ");

result = (a > b);

document.write(result);

document.write(linebreak);

document.write("(a != b) => ");

result = (a != b);

document.write(result);

document.write(linebreak);

document.write("(a >= b) => ");

result = (a >= b);

document.write(result);

document.write(linebreak);

document.write("(a <= b) => ");
```

```
result = (a <= b);
document.write(result);
document.write(linebreak);
//-->
</script>
<p>Set the variables to different values and different operators and
then try...</p>
</body>
</html>
```

## Output

```
(a == b) => false
(a < b) => true
(a > b) => false
(a != b) => true
(a >= b) => false
(a <= b) => true
```

## Logical Operators

```
<html>
<body>
<script type="text/javascript">
<!--
var a = true;
var b = false;
var linebreak = "
";
document.write("(a && b) => ");
result = (a && b);
document.write(result);
document.write(linebreak);
document.write("(a || b) => ");
```

```

result = (a || b);
document.write(result);
document.write(linebreak);
document.write("!(a && b) => ");
result = !(a && b);
document.write(result);
document.write(linebreak);
//-->
</script>
<p>Set the variables to different values and different operators and
then try...</p>
</body>
</html>

```

### Output

```

(a && b) => false
(a || b) => true
!(a && b) => true

```

### Bitwise Operators

```

<html>
<body>
<script type="text/javascript">
<!--
var a = 2; // Bit presentation 10
var b = 3; // Bit presentation 11
var linebreak = "
";
document.write("(a & b) => ");
result = (a & b);
document.write(result);
document.write(linebreak);
document.write("(a | b) => ");

```

```
result = (a | b);
document.write(result);
document.write(linebreak);
document.write("(a ^ b) => ");
result = (a ^ b);
document.write(result);
document.write(linebreak);
document.write("(~b) => ");
result = (~b);
document.write(result);
document.write(linebreak);
document.write("(a << b) => ");
result = (a << b);
document.write(result);
document.write(linebreak);
document.write("(a >> b) => ");
result = (a >> b);
document.write(result);
document.write(linebreak);
//-->
</script>
```

<p>Set the variables to different values and different operators and then try...</p>

</body>

</html>

### **Output**

(a & b) => 2  
(a | b) => 3  
(a ^ b) => 1  
(~b) => -4

(a << b) => 16

(a >> b) => 0

### **Assignment Operators**

<html>

<body>

<script type="text/javascript">

<!--

var a = 33;

var b = 10;

var linebreak = "<br />";

document.write("Value of a => (a = b) => ");

result = (a = b);

document.write(result);

document.write(linebreak);

document.write("Value of a => (a += b) => ");

result = (a += b);

document.write(result);

document.write(linebreak);

document.write("Value of a => (a -= b) => ");

result = (a -= b);

document.write(result);

document.write(linebreak);

document.write("Value of a => (a \*= b) => ");

result = (a \*= b);

document.write(result);

document.write(linebreak);

document.write("Value of a => (a /= b) => ");

result = (a /= b);

document.write(result);

document.write(linebreak);

```

document.write("Value of a => (a %= b) => ");
result = (a %= b);
document.write(result);
document.write(linebreak);
//-->
</script>
<p>Set the variables to different values and different operators and
then try...</p>
</body>
</html>

```

## Output

```

Value of a => (a = b) => 10
Value of a => (a += b) => 20
Value of a => (a -= b) => 10
Value of a => (a *= b) => 100
Value of a => (a /= b) => 10
Value of a => (a %= b) => 0

```

## Miscellaneous Operators

conditional operator (? :) and the typeof operator

### Conditional Operator ( ? : )

The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

#### ? : (Conditional )

**If Condition is true? Then value X : Otherwise value Y**

```

<html>

<body>

<script type="text/javascript">

<!--

var a = 10;

```

```

var b = 20;

var linebreak = "
";

document.write ("((a > b) ? 100 : 200) => ");

result = (a > b) ? 100 : 200;

document.write(result);

document.write(linebreak);

document.write ("((a < b) ? 100 : 200) => ");

result = (a < b) ? 100 : 200;

document.write(result);

document.write(linebreak);

//-->

</script>

<p>Set the variables to different values and different operators and
then try...</p>

</body>

</html>

```

## Output

```

((a > b) ? 100 : 200) => 200

((a < b) ? 100 : 200) => 100

```

## typeof Operator

The typeof operator is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.

The typeof operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

Here is a list of the return values for the typeof Operator.

**Table 2.1 typeof Operator**

| Type    | String Returned by<br>typeof |
|---------|------------------------------|
| Number  | "number"                     |
| String  | String                       |
| Boolean | "boolean"                    |

|           |             |
|-----------|-------------|
| Object    | "object"    |
| Function  | "function"  |
| Undefined | "undefined" |
| Null      | "object"    |

### Example

```

<html>
<body>
<script type="text/javascript">
<!--
var a = 10;
var b = "String";
var linebreak = "
";
result = (typeof b == "string" ? "B is String" : "B is Numeric");
document.write("Result => ");
document.write(result);
document.write(linebreak);
result = (typeof a == "string" ? "A is String" : "A is Numeric");
document.write("Result => ");
document.write(result);
document.write(linebreak);
//-->
</script>
<p>Set the variables to different values and different operators and
then try...</p>
</body>
</html>

```

### Output

Result => B is String

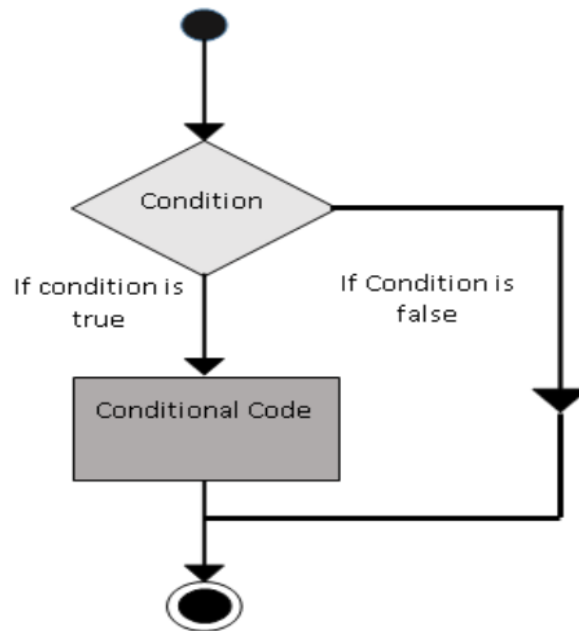
Result => A is Numeric



## CONTROL STATEMENTS

JavaScript supports conditional statements which are used to perform different actions based on different conditions.

### Flow Chart of if-else



**Fig. 2.2 if-else statement**

**JavaScript supports the following forms of if..else statement:**

- if statement
- if...else statement
- if...else if... statement

### if Statement

The 'if' statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

### Syntax

The syntax for a basic if statement is as follows:

```
if (expression)
{
 Statement(s) to be executed if expression is true
}
```

Here a JavaScript expression is evaluated. If the resulting value is true, the given statement(s) are executed. If the expression is false, then no statement would be not executed. Most of the times, you will use comparison operators while making decisions.

### Example

```
<html>

<body>

<script type="text/javascript">

<!--
var age = 20;
if(age > 18){
 document.write("Qualifies for driving");
}
//-->

</script>

<p>Set the variable to different value and then try...</p>

</body>

</html>
```

### Output

Qualifies for driving

Set the variable to different value and then try...

### if...else Statement

The ‘if...else’ statement is the next form of control statement that allows JavaScript to execute statements in a more controlled way.

### Syntax

```
if (expression){
 Statement(s) to be executed if expression is true
}else{
 Statement(s) to be executed if expression is false
}
```

Here JavaScript expression is evaluated. If the resulting value is true, the given statement(s) in the ‘if’ block, are executed. If the expression is false, then the given statement(s) in the else block are executed.

### Example

```

<html>
<body>
<script type="text/javascript">
<!--
var age = 15;
if(age > 18){
 document.write("Qualifies for driving");
}else{
 document.write("Does not qualify for driving");
}!-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>

```

## Output

Does not qualify for driving

Set the variable to different value and then try...

## if...else if... Statement

The 'if...else if...' statement is an advanced form of if...else that allows JavaScript to make a correct decision out of several conditions.

## Syntax

```

if (expression 1){
 Statement(s) to be executed if expression 1 is true
}else if (expression 2){
 Statement(s) to be executed if expression 2 is true
}else if (expression 3){
 Statement(s) to be executed if expression 3 is true
}else{
 Statement(s) to be executed if no expression is true
}

```

```
}
```

There is nothing special about this code. It is just a series of if statements, where each if is a part of the else clause of the previous statement. Statement(s) are executed based on the true condition, if none of the conditions is true, then the else block is executed.

### Example

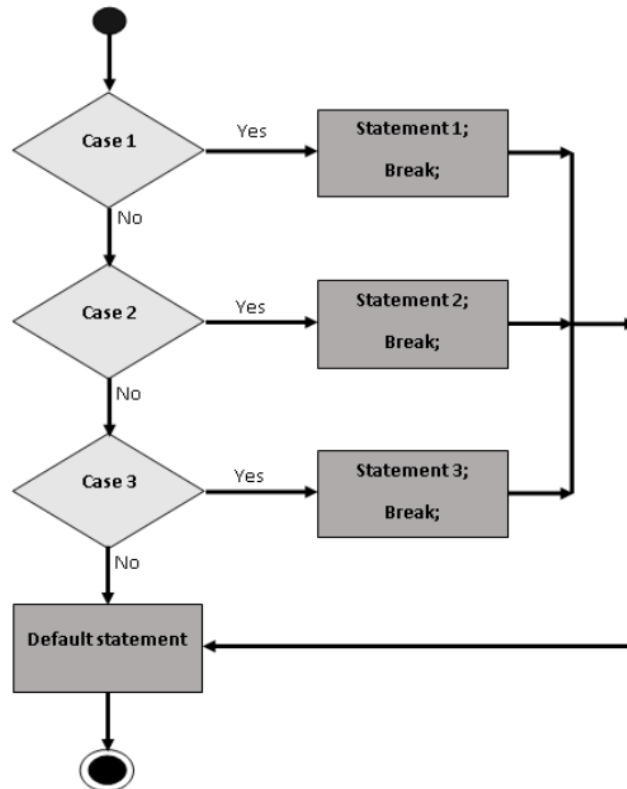
```
<html>
<body>
<script type="text/javascript">
<!--
var book = "maths";
if(book == "history"){
 document.write("History Book");
}else if(book == "maths"){
 document.write("Maths Book");
}else if(book == "economics"){
 document.write("Economics Book");
}else{
 document.write("Unknown Book");
}
//-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

### Output

Maths Book

Set the variable to different value and then try...

## SWITCH-CASE



**Fig. 2.3 Switch statement**

### Syntax

The objective of a switch statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.

switch (expression)

{

case condition 1: statement(s)

break;

case condition 2: statement(s)

break;

...

case condition n: statement(s)

break;

```
default: statement(s)
```

```
}
```

The break statements indicate the end of a particular case. If they were omitted, the interpreter would continue executing each statement in each of the following cases.

### Example

```
<html>
```

```
<body>
```

```
<script type="text/javascript">
```

```
<!--
```

```
var grade='A';
```

```
document.write("Entering switch block
");
```

```
switch (grade)
```

```
{
```

```
case 'A': document.write("Good job
");
```

```
break;
```

```
case 'B': document.write("Pretty good
");
```

```
break;
```

```
case 'C': document.write("Passed
");
```

```
break;
```

```
case 'D': document.write("Not so good
");
```

```
break;
```

```
case 'F': document.write("Failed
");
```

```
break;
```

```
default: document.write("Unknown grade
")
```

```
}
```

```
document.write("Exiting switch block");
```

```
//-->
```

```
</script>
```

```
<p>Set the variable to different value and then try...</p>
```

```
</body>
```

```
</html>
```

## Output

Entering switch block

Good job

Exiting switch block

Set the variable to different value and then try...

Break statements play a major role in switch-case statements. Try the following code that uses switch-case statement without any break statement.

```
<html>

<body>

<script type="text/javascript">

<!--
var grade='A';
document.write("Entering switch block
");

switch (grade)
{
case 'A': document.write("Good job
");
case 'B': document.write("Pretty good
");
case 'C': document.write("Passed
");
case 'D': document.write("Not so good
");
case 'F': document.write("Failed
");
default: document.write("Unknown grade
")
}
document.write("Exiting switch block");
//-->
</script>

<p>Set the variable to different value and then try...</p>

</body>

</html>
```

## Output

Entering switch block

Good job

Pretty good

Passed

Not so good

Failed

Unknown grade

Exiting switch block

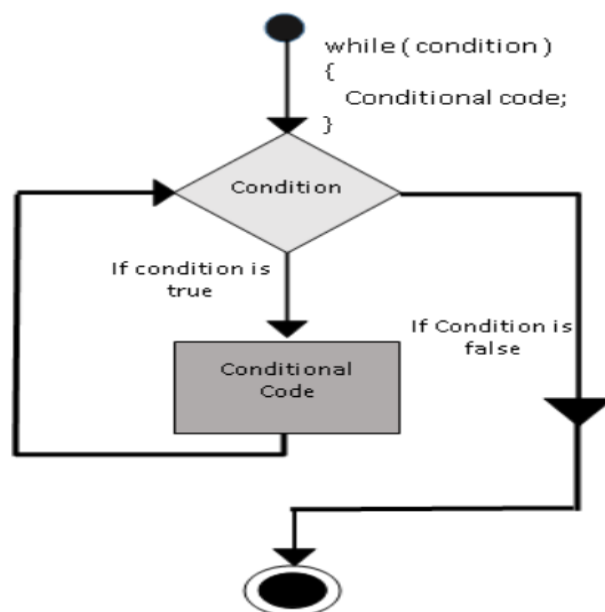
Set the variable to different value and then try...

## WHILE LOOP

While writing a program, you may encounter a situation where you need to perform an action over and over again. In such situations, you would need to write loop statements to reduce the number of lines. JavaScript supports all the necessary loops to ease down the pressure of programming.

### The while Loop

The most basic loop in JavaScript is the while loop which would be discussed in this chapter. The purpose of a while loop is to execute a statement or code block repeatedly as long as an expression is true. Once the expression becomes false, the loop terminates.





**Fig. 2.4 While loop**

**Syntax**

```
while (expression){
 Statement(s) to be executed if expression is true
}
```

**Example**

```
<html>
<body>
<script type="text/javascript">
<!--
var count = 0;
document.write("Starting Loop ");
while (count < 10){
 document.write("Current Count : " + count + "
");
 count++;
}
document.write("Loop stopped!");
//-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

**Output**

```
Starting Loop Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
```

Current Count : 7

Current Count : 8

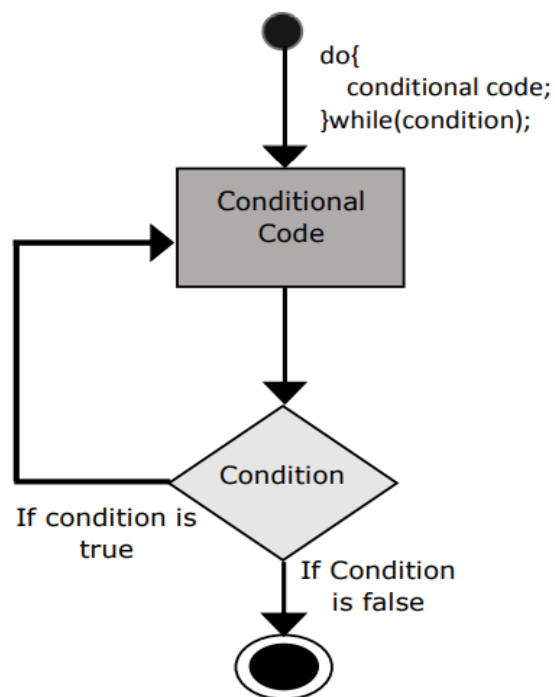
Current Count : 9

Loop stopped!

Set the variable to different value and then try...

### The do...while Loop

The do...while loop is similar to the while loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is false.



**Fig. 2.5 do-while loop**

### Syntax

do

{

Statement(s) to be executed;

```
} while (expression);
```

### **Example**

```
<html>

<body>

<script type="text/javascript">

<!--

var count = 0;

document.write("Starting Loop" + "
");

do{

 document.write("Current Count : " + count + "
");

 count++;

}while (count < 5);

document.write ("Loop stopped!");

//-->

</script>

<p>Set the variable to different value and then try...</p>

</body>

</html>
```

### **Output**

Starting Loop

Current Count : 0

Current Count : 1

Current Count : 2

Current Count : 3

Current Count : 4

Loop Stopped!

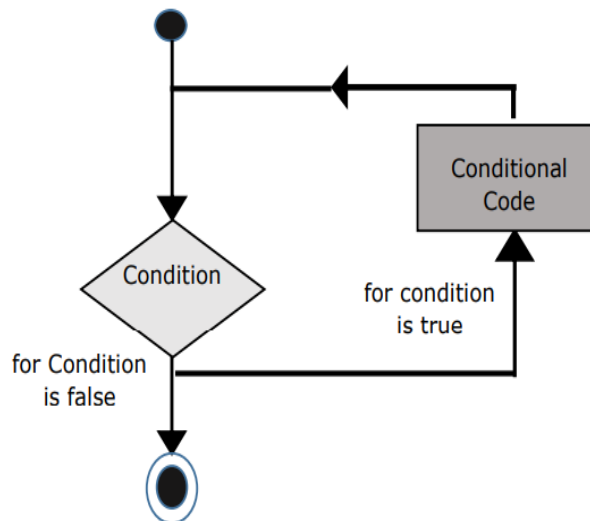
Set the variable to different value and then try...

## **FOR LOOP**

The 'for' loop is the most compact form of looping. It includes the following three important parts:

- The loop initialization where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The test statement which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.
- The iteration statement where you can increase or decrease your counter.

You can put all the three parts in a single line separated by semicolons.



**Fig. 2.6 for loop**

### Syntax

```

for (initialization; test condition; iteration statement)
{
 Statement(s) to be executed if test condition is true
}

```

### Example

```

<html>

<body>

<script type="text/javascript">

```

```
<!--
var count;
document.write("Starting Loop" + "
");
for(count = 0; count < 10; count++){
 document.write("Current Count : " + count);
 document.write("
");
}
document.write("Loop stopped!");
//-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

## **Output**

Starting Loop

Current Count : 0

Current Count : 1

Current Count : 2

Current Count : 3

Current Count : 4

Current Count : 5

Current Count : 6

Current Count : 7

Current Count : 8

Current Count : 9

Loop stopped!

Set the variable to different value and then try...

## **FOR-IN LOOP**

## Syntax

```
for (variablename in object){
 statement or block to execute
}
```

## Example

```
<html>
<body>
<script type="text/javascript">
<!--
var aProperty;
document.write("Navigator Object Properties
 ");
for (aProperty in navigator)
{
 document.write(aProperty);
 document.write("
");
}
document.write ("Exiting from the loop!");
//-->
</script>
<p>Set the variable to different object and then try...</p>
</body>
</html>
```

## Output

```
Navigator Object Properties
serviceWorker
webkitPersistentStorage
webkitTemporaryStorage
geolocation
doNotTrack
onLine
```

languages  
language  
userAgent  
product  
platform  
appVersion  
appName  
appCodeName  
hardwareConcurrency  
maxTouchPoints  
vendorSub  
vendor  
productSub  
cookieEnabled  
mimeTypes  
plugins  
javaEnabled  
getStorageUpdates  
getGamepads  
webkitGetUserMedia  
vibrate  
getBattery  
sendBeacon  
registerProtocolHandler  
unregisterProtocolHandler

Exiting from the loop!

Set the variable to different object and then try..

## **LOOP CONTROL**

JavaScript provides full control to handle loops and switch statements. There may be a situation when you need to come out of a loop without reaching at its bottom. There may also be a situation when

you want to skip a part of your code block and start the next iteration of the loop. To handle all such situations, JavaScript provides break and continue statements. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

### The break Statement

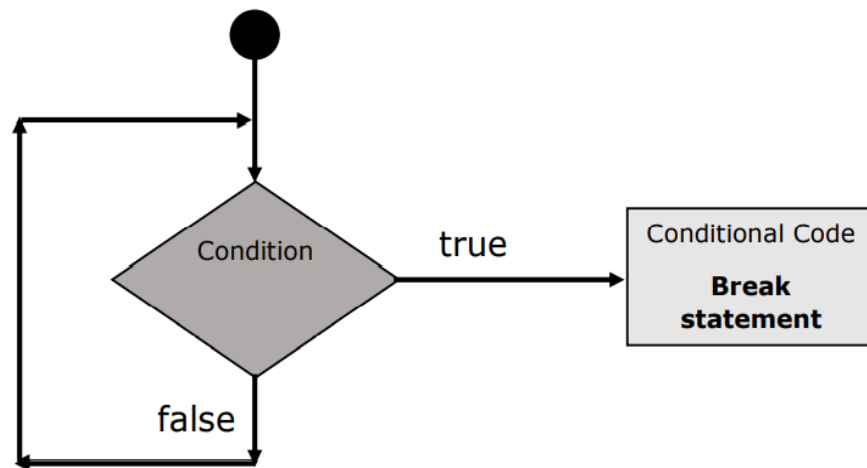


Fig. 2.7 break statement

### Example

```
<html>
<body>
<script type="text/javascript">
<!--
var x = 1;
document.write("Entering the loop
 ");
while (x < 20)
{
 if (x == 5){
 break; // breaks out of loop completely
 }
 x = x + 1;
 document.write(x + "
");
}
document.write("Exiting the loop!
 ");
```



```
//-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

## Output

Entering the loop

2

3

4

5

Exiting the loop!

Set the variable to different value and then try...

## The continue Statement

The continue statement tells the interpreter to immediately start the next iteration of the loop and skip the remaining code block. When a continue statement is encountered, the program flow moves to the loop check expression immediately and if the condition remains true, then it starts the next iteration, otherwise the control comes out of the loop.

## Example

```
<html>
<body>
<script type="text/javascript">
<!--
var x = 1;
document.write("Entering the loop
 ");
while (x < 10)
{
 x = x + 1;
 if (x == 5){
 continue; // skip rest of the loop body
```

```
}
document.write(x + "
");
}
document.write("Exiting the loop!
 ");
//-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

## Output

### Entering the loop

2  
3  
4  
6  
7  
8  
9  
10

### Exiting the loop!

## FUNCTIONS

A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again. It helps programmers in writing modular codes. Functions allow a programmer to divide a big program into a number of small and manageable functions.

Like any other advanced programming language, JavaScript also supports all the features necessary to write modular code using functions. You must have seen functions like alert() and write().

JavaScript allows us to write our own functions as well.

### Syntax

```
<script type="text/javascript">
<!--
```

```
function functionname(parameter-list)
```

```
{
```

```
 statements
```

```
}
```

```
//-->
```

```
</script>
```

### **Example**

```
<script type="text/javascript">
```

```
<!--
```

```
function sayHello()
```

```
{
```

```
 alert("Hello there");
```

```
}
```

```
//-->
```

```
</script>
```

```
<html>
```

```
<head>
```

```
<script type="text/javascript">
```

```
function sayHello()
```

```
{
```

```
 document.write ("Hello there!");
```

```
}
```

```
</script>
```

```
</head>
```

```
<body>
```

```
<p>Click the following button to call the function</p>
```

```
<form>
```

```
<input type="button" onclick="sayHello()" value="Say Hello">
```

```
</form>
<p>Use different text in write method and then try...</p>
</body>
</html>
```

## Function Parameters

```
<html>
<head>
<script type="text/javascript">
function sayHello(name, age)
{
 document.write (name + " is " + age + " years old.");
}
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="sayHello('Zara', 7)" value="Say Hello">
</form>
<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

## Nested Functions

```
<html>
<head>
<script type="text/javascript">
<!--
function hypotenuse(a, b) {
 function square(x) { return x*x; }
```

```

 return Math.sqrt(square(a) + square(b));
}
function secondFunction(){
 var result;
 result = hypotenuse(1,2);
 document.write (result);
}
//-->
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="secondFunction()" value="Call Function">
</form>
<p>Use different parameters inside the function and then try...</p>
</body>
</html>

```

## JavaScript Objects

JavaScript is an Object Oriented Programming (OOP) language. A programming language can be called object-oriented if it provides four basic capabilities to developers –

- **Encapsulation** – the capability to store related information, whether data or methods, together in an object.
- **Aggregation** – the capability to store one object inside another object.
- **Inheritance** – the capability of a class to rely upon another class (or number of classes) for some of its properties and methods.
- **Polymorphism** – the capability to write one function or method that works in a variety of different ways.

Objects are composed of attributes. If an attribute contains a function, it is considered to be a method of the object, otherwise the attribute is considered a property.

## Object Properties

Object properties can be any of the three primitive data types, or any of the abstract data types, such as another object. Object properties are usually variables that are used internally in the object's methods, but can also be globally visible variables that are used throughout the page.

The syntax for adding a property to an object is –

```
objectName.objectProperty = propertyValue;
```

**For example** – The following code gets the document title using the **"title"** property of the **document** object.

```
var str = document.title;
```

## Object Methods

Methods are the functions that let the object do something or let something be done to it. There is a small difference between a function and a method – a function is a standalone unit of statements and a method is attached to an object and can be referenced by the **this** keyword.

Methods are useful for everything from displaying the contents of the object to the screen to performing complex mathematical operations on a group of local properties and parameters.

**For example** – Following is a simple example to show how to use the **write()** method of document object to write any content on the document.

```
document.write("This is test");
```

## User-Defined Objects

All user-defined objects and built-in objects are descendants of an object called **Object**.

## The new Operator

The **new** operator is used to create an instance of an object. To create an object, the **new** operator is followed by the constructor method.

In the following example, the constructor methods are `Object()`, `Array()`, and `Date()`. These constructors are built-in JavaScript functions.

```
var employee = new Object();
var books = new Array("C++", "Perl", "Java");
var day = new Date("August 15, 1947");
```

## The Object() Constructor

A constructor is a function that creates and initializes an object. JavaScript provides a special constructor function called **Object()** to build the object. The return value of the **Object()** constructor is assigned to a variable.

The variable contains a reference to the new object. The properties assigned to the object are not variables and are not defined with the **var** keyword.

### Example 1

```
<html>
<head>
 <title>User-defined objects</title>
 <script type = "text/javascript">
 var book = new Object(); // Create the object
 book.subject = "Perl"; // Assign properties to the object
 book.author = "Mohtashim";
 </script>
</head>

<body>
 <script type = "text/javascript">
 document.write("Book name is : " + book.subject + "
");
 document.write("Book author is : " + book.author + "
");
 </script>
</body>
</html>
```

## Output

Book name is : Perl  
Book author is : Mohtashim

## JavaScript - The Number Object

The **Number** object represents numerical data, either integers or floating-point numbers. In general, you do not need to worry about **Number** objects because the browser automatically converts number literals to instances of the number class.

## Syntax

The syntax for creating a **number** object is as follows –

```
var val = new Number(number);
```

In the place of number, if you provide any non-number argument, then the argument cannot be converted into a number, it returns **NaN** (Not-a-Number).

## Number Properties

Here is a list of each property and their description.

**Table 2.2 Number Object - Properties and Description**

Sr.No.	Property & Description
1	MAX_VALUE The largest possible value a number in JavaScript can have 1.7976931348623157E+308
2	MIN_VALUE The smallest possible value a number in JavaScript can have 5E-324
3	NaN Equal to a value that is not a number.
4	NEGATIVE_INFINITY A value that is less than MIN_VALUE.



5	<b>POSITIVE_INFINITY</b> A value that is greater than MAX_VALUE
6	<b>prototype</b> A static property of the Number object. Use the prototype property to assign new properties and methods to the Number object in the current document
7	<b>constructor</b> Returns the function that created this object's instance. By default this is the Number object.

## Number Methods

The Number object contains only the default methods that are a part of every object's definition.

**Table 2.3 Number Object – Method and Description**

Sr.No.	Method & Description
1	<b>toExponential()</b> Forces a number to display in exponential notation, even if the number is in the range in which JavaScript normally uses standard notation.
2	<b>toFixed()</b> Formats a number with a specific number of digits to the right of the decimal.
3	<b>toLocaleString()</b> Returns a string value version of the current number in a format that may vary according to a browser's local settings.
4	<b>toPrecision()</b> Defines how many total digits (including digits to the left and right of the decimal) to display of a number.
5	<b>toString()</b>

	Returns the string representation of the number's value.
6	valueOf() Returns the number's value.

## JavaScript Number - toExponential()

### Description

This method returns a string representing the **number** object in exponential notation.

### Syntax

Its syntax is as follows –

`number.toExponential( [fractionDigits] )`

### Parameter Details

**fractionDigits** – An integer specifying the number of digits after the decimal point. Defaults to as many digits as necessary to specify the number.

### Return Value

A string representing a Number object in exponential notation with one digit before the decimal point, rounded to **fractionDigits** digits after the decimal point. If the **fractionDigits** argument is omitted, the number of digits after the decimal point defaults to the number of digits necessary to represent the value uniquely.

### Example

```
<html>
 <head>
 <title>Javascript Method toExponential()</title>
 </head>

 <body>
```

```
<script type = "text/javascript">
 var num = 77.1234;
 var val = num.toExponential();
 document.write("num.toExponential() is : " + val);
 document.write("
");

 val = num.toExponential(4);
 document.write("num.toExponential(4) is : " + val);
 document.write("
");

 val = num.toExponential(2);
 document.write("num.toExponential(2) is : " + val);
 document.write("
");

 val = 77.1234.toExponential();
 document.write("77.1234.toExponential()is : " + val);
 document.write("
");

 val = 77.1234.toExponential();
 document.write("77 .toExponential() is : " + val);
</script>
</body>
</html>
```

### Output

```
num.toExponential() is : 7.71234e+1
num.toExponential(4) is : 7.7123e+1
num.toExponential(2) is : 7.71e+1
77.1234.toExponential()is:7.71234e+1
77 .toExponential() is : 7.71234e+1
```

JavaScript Number - valueOf()

### Description

This method returns the primitive value of the specified **number** object.

### Syntax

Its syntax is as follows –

number.valueOf()

## Return Value

Returns the primitive value of the specified **number** object.

```
<html>
<head>
 <title>JavaScript valueOf() Method </title>
</head>

<body>
 <script type = "text/javascript">
 var num = new Number(15.11234);
 document.write("num.valueOf() is " + num.valueOf());
 </script>
</body>
</html>
```

### Output

num.valueOf() is 15.11234

Two values, either "true" or "false". If *value* parameter is omitted or is 0, -0, null, false, NaN, undefined, or the empty string (""), the object has an initial value of false.

## Syntax

Use the following syntax to create a **boolean** object.

```
var val = new Boolean(value);
```

## Boolean Properties

Here is a list of the properties of Boolean object –

**Table 2.4 Boolean Object – Properties and Description**

Sr.No.	Property & Description
1	<u>constructor</u> Returns a reference to the Boolean function that created the object.
2	<u>prototype</u> The prototype property allows you to add properties and methods to an object.

In the following sections, we will have a few examples to illustrate the properties of Boolean object.

## Boolean Methods

Here is a list of the methods of Boolean object and their description.

**Table 2.5 Boolean Object – Methods and Description**

Sr.No.	Method & Description
1	<u>toSource()</u> Returns a string containing the source of the Boolean object; you can use this string to create an equivalent object.
2	<u>toString()</u> Returns a string of either "true" or "false" depending upon the value of the object.
3	<u>valueOf()</u> Returns the primitive value of the Boolean object.

### JavaScript - Boolean valueOf() Method

#### Description

JavaScript boolean **valueOf()** method returns the primitive value of the specified **boolean** object.

#### Syntax

Its syntax is as follows –

```
boolean.valueOf()
```

#### Return Value

Returns the primitive value of the specified **boolean** object.

#### Example

```
<html>
 <head>
 <title>JavaScript valueOf() Method</title>
 </head>
```

```
<body>
 <script type = "text/javascript">
 var flag = new Boolean(false);
 document.write("flag.valueOf is : " + flag.valueOf());
 </script>
</body>
</html>
```

### Output

flag.valueOf is : false

work with a series of characters; it wraps Javascript's string primitive data type with a number of helper methods.

As JavaScript automatically converts between string primitives and String objects, you can call any of the helper methods of the String object on a string primitive.

### Syntax

Use the following syntax to create a String object –

```
var val = new String(string);
```

The **String** parameter is a series of characters that has been properly encoded.

### String Properties

Here is a list of the properties of String object and their description.

**Table 2.6 String Object – Property and Description**

Sr.No.	Property & Description
1	<u>constructor</u> Returns a reference to the String function that created the object.
2	<u>length</u> Returns the length of the string.
3	<u>prototype</u> The prototype property allows you to add properties and methods to an object.

## String Methods

Here is a list of the methods available in String object along with their description.

**Table 2.7 String Object – Method and Description**

Sr.No.	Method & Description
1	<u>charAt()</u> Returns the character at the specified index.
2	<u>charCodeAt()</u> Returns a number indicating the Unicode value of the character at the given index.
3	<u>concat()</u> Combines the text of two strings and returns a new string.
4	<u>indexOf()</u> Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.
5	<u>lastIndexOf()</u> Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found.
6	<u>localeCompare()</u> Returns a number indicating whether a reference string comes before or after or is the same as the given string in sort order.
7	<u>match()</u> Used to match a regular expression against a string.
8	<u>replace()</u>

	Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.
9	<u>search()</u> Executes the search for a match between a regular expression and a specified string.
10	<u>slice()</u> Extracts a section of a string and returns a new string.
11	<u>split()</u> Splits a String object into an array of strings by separating the string into substrings.
12	<u>substr()</u> Returns the characters in a string beginning at the specified location through the specified number of characters.
13	<u>substring()</u> Returns the characters in a string between two indexes into the string.
14	<u>toLocaleLowerCase()</u> The characters within a string are converted to lower case while respecting the current locale.
15	<u>toLocaleUpperCase()</u> The characters within a string are converted to upper case while respecting the current locale.
16	<u>toLowerCase()</u> Returns the calling string value converted to lower case.
17	<u>toString()</u> Returns a string representing the specified object.
18	<u>toUpperCase()</u>



	Returns the calling string value converted to uppercase.
19	<u>valueOf()</u> Returns the primitive value of the specified object.

```

<!DOCTYPE html>

<html>

<body>

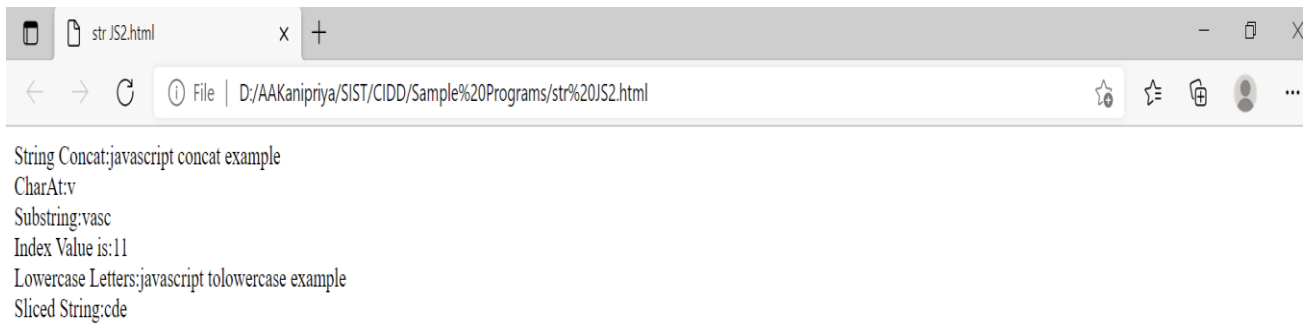
<script>
var s1="javascript ";
var s2="concat example";
var s3=s1.concat(s2);
document.write("String Concat:"+s3+"</br>");
var str="javascript";
document.write("CharAt:"+str.charAt(2)+"</br>");
document.write("Substring:"+str.substr(2,4)+"</br>");
var s1="javascript from sample programs";
var n=s1.indexOf("from");
document.write("Index Value is:"+n+"</br>");
var s1="JavaScript toLowerCase Example";
var s2=s1.toLowerCase();
document.write("Lowercase Letters:"+s2+"</br>");
var s1="abcdefgh";
var s2=s1.slice(2,5);
document.write("Sliced String:"+s2);
</script>

</body>

</html>

```

**Output:**



## JavaScript - The Arrays Object

The **Array** object lets you store multiple values in a single variable. It stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

### Syntax

Use the following syntax to create an **Array** object –

```
var fruits = new Array("apple", "orange", "mango");
```

The **Array** parameter is a list of strings or integers. When you specify a single numeric parameter with the Array constructor, you specify the initial length of the array. The maximum length allowed for an array is 4,294,967,295.

You can create array by simply assigning values as follows –

```
var fruits = ["apple", "orange", "mango"];
```

You will use ordinal numbers to access and to set values inside an array as follows.

fruits[0] is the first element

fruits[1] is the second element

fruits[2] is the third element

## Array Properties

Here is a list of the properties of the Array object along with their description.

**Table 2.8 Array Object – Properties and Description**

Sr.No.	Property & Description
1	<u>constructor</u> Returns a reference to the array function that created the object.
2	<b>index</b> The property represents the zero-based index of the match in the string
3	<b>input</b> This property is only present in arrays created by regular expression matches.
4	<u>length</u> Reflects the number of elements in an array.
5	<u>prototype</u> The prototype property allows you to add properties and methods to an object.

In the following sections, we will have a few examples to illustrate the usage of Array properties.

## Array Methods

Here is a list of the methods of the Array object along with their description.

**Table 2.9 Array Object – Method and Description**

Sr.No.	Method & Description
1	<u>concat()</u> Returns a new array comprised of this array joined with other array(s) and/or value(s).
2	<u>every()</u> Returns true if every element in this array satisfies the provided testing function.
3	<u>filter()</u>

	Creates a new array with all of the elements of this array for which the provided filtering function returns true.
4	<u>forEach()</u> Calls a function for each element in the array.
5	<u>indexOf()</u> Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.
6	<u>join()</u> Joins all elements of an array into a string.
7	<u>lastIndexOf()</u> Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found.
8	<u>map()</u> Creates a new array with the results of calling a provided function on every element in this array.
9	<u>pop()</u> Removes the last element from an array and returns that element.
10	<u>push()</u> Adds one or more elements to the end of an array and returns the new length of the array.
11	<u>reduce()</u> Apply a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.
12	<u>reduceRight()</u> Apply a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value.

13	<u>reverse()</u> Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.
14	<u>shift()</u> Removes the first element from an array and returns that element.
15	<u>slice()</u> Extracts a section of an array and returns a new array.
16	<u>some()</u> Returns true if at least one element in this array satisfies the provided testing function.
17	<u>toSource()</u> Represents the source code of an object
18	<u>sort()</u> Sorts the elements of an array
19	<u>splice()</u> Adds and/or removes elements from an array.
20	<u>toString()</u> Returns a string representing the array and its elements.
21	<u>unshift()</u> Adds one or more elements to the front of an array and returns the new length of the array.

## JavaScript - Array concat() Method

```
<html>
<head>
 <title>JavaScript Array concat Method</title>
</head>
```

```

<body>
 <script type = "text/javascript">
 var alpha = ["a", "b", "c"];
 var numeric = [1, 2, 3];
 var alphaNumeric = alpha.concat(numeric);
 document.write("alphaNumeric : " + alphaNumeric);
 </script>
</body>
</html>

```

### Output

alphaNumeric : a,b,c,1,2,3

## JavaScript - Array pop() Method

```

<html>
 <head>
 <title>JavaScript Array pop Method</title>
 </head>

 <body>
 <script type = "text/javascript">
 var numbers = [1, 4, 9];

 var element = numbers.pop();
 document.write("element is : " + element);

 var element = numbers.pop();
 document.write("
element is : " + element);
 </script>
 </body>
</html>

```

### Output

element is : 9  
element is : 4

Method

```

<html>
 <head>
 <title>JavaScript Array push Method</title>
 </head>

 <body>
 <script type = "text/javascript">
 var numbers = new Array(1, 4, 9);
 var length = numbers.push(10);
 </script>
 </body>
</html>

```

```
document.write("new numbers is : " + numbers);
length = numbers.push(20);
document.write("
new numbers is : " + numbers);
</script>
</body>
</html>
```

### Output

new numbers is : 1,4,9,10  
new numbers is : 1,4,9,10,20

## JavaScript - Array sort() Method

```
<html>
<head>
 <title>JavaScript Array sort Method</title>
</head>

<body>
 <script type = "text/javascript">
 var arr = new Array("orange", "mango", "banana", "sugar");
 var sorted = arr.sort();
 document.write("Returned string is : " + sorted);
 </script>
</body>
</html>
```

### Output

Returned array is :  
banana,mango,orange,sugar

## Date Object

- Date is a data type.
- Date object manipulates date and time.
- Date() constructor takes no arguments.

- Date object allows you to get and set the year, month, day, hour, minute, second and millisecond fields.

**Syntax:**

```
var variable_name = new Date();
```

**Example:**

```
var current_date = new Date();
```

**Date Methods**

**Table 2.10 Date Object – Methods and Description**

Methods	Description
Date()	Returns current date and time.
getDate()	Returns the day of the month.
getDay()	Returns the day of the week.
getFullYear()	Returns the year.
getHours()	Returns the hour.
getMinutes()	Returns the minutes.
getSeconds()	Returns the seconds.
getMilliseconds()	Returns the milliseconds.
getTime()	Returns the number of milliseconds since January 1, 1970 at 12:00 AM.
getTimezoneOffset()	Returns the timezone offset in minutes for the current locale.
getMonth()	Returns the month.
setDate()	Sets the day of the month.
setFullYear()	Sets the full year.
setHours()	Sets the hours.
setMinutes()	Sets the minutes.
setSeconds()	Sets the seconds.
setMilliseconds()	Sets the milliseconds.

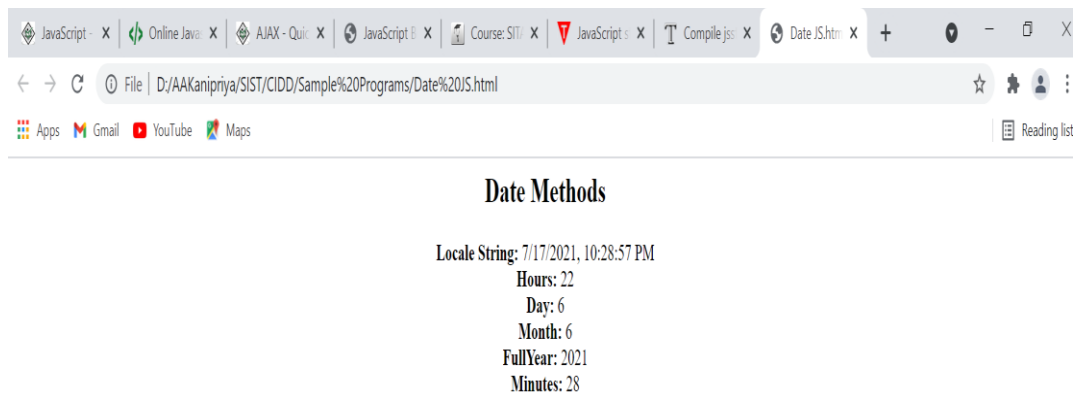


setTime()	Sets the number of milliseconds since January 1, 1970 at 12:00 AM.
setMonth()	Sets the month.
toString()	Returns the date portion of the Date as a human-readable string.
toLocaleString()	Returns the Date object as a string.
toGMTString()	Returns the Date object as a string in GMT timezone.
valueOf()	Returns the primitive value of a Date object.

### Example : JavaScript Date() Methods Program

```
<html>
<body>
<center>
<h2>Date Methods</h2>
<script type="text/javascript">
 var d = new Date();
 document.write("Locale String: " + d.toLocaleString()+"
");
 document.write("Hours: " + d.getHours()+"
");
 document.write("Day: " + d.getDay()+"
");
 document.write("Month: " + d.getMonth()+"
");
 document.write("FullYear: " + d.getFullYear()+"
");
 document.write("Minutes: " + d.getMinutes()+"
");
</script>
</center>
</body>
</html>
```

### Output



## JavaScript - The Math Object

The **math** object provides you properties and methods for mathematical constants and functions. Unlike other global objects, **Math** is not a constructor. All the properties and methods of **Math** are static and can be called by using Math as an object without creating it.

Thus, you refer to the constant **pi** as **Math.PI** and you call the *sine* function as **Math.sin(x)**, where x is the method's argument.

### Syntax

The syntax to call the properties and methods of Math are as follows

```
var pi_val = Math.PI;
var sine_val = Math.sin(30);
```

### Math Methods

Here is a list of the methods associated with Math object and their description

**Table 2.11 Math Object – Method and Description**

Sr.No.	Method & Description
1	<u>abs()</u> Returns the absolute value of a number.
2	<u>acos()</u> Returns the arccosine (in radians) of a number.
3	<u>asin()</u> Returns the arcsine (in radians) of a number.
4	<u>atan()</u> Returns the arctangent (in radians) of a number.
5	<u>atan2()</u> Returns the arctangent of the quotient of its arguments.
6	<u>ceil()</u>

	Returns the smallest integer greater than or equal to a number.
7	<u>cos()</u> Returns the cosine of a number.
8	<u>exp()</u> Returns $E^N$ , where N is the argument, and E is Euler's constant, the base of the natural logarithm.
9	<u>floor()</u> Returns the largest integer less than or equal to a number.
10	<u>log()</u> Returns the natural logarithm (base E) of a number.
11	<u>max()</u> Returns the largest of zero or more numbers.
12	<u>min()</u> Returns the smallest of zero or more numbers.
13	<u>pow()</u> Returns base to the exponent power, that is, base exponent.
14	<u>random()</u> Returns a pseudo-random number between 0 and 1.
15	<u>round()</u> Returns the value of a number rounded to the nearest integer.
16	<u>sin()</u> Returns the sine of a number.
17	<u>sqrt()</u> Returns the square root of a number.

18	<u>tan()</u> Returns the tangent of a number.
19	<u>toSource()</u> Returns the string "Math".

```
<html>
```

```
<head>
```

```
<title>JavaScript Math functions </title>
```

```
</head>
```

```
<body>
```

```
<script type = "text/javascript">
```

```
var value = Math.max(10, 20, -1, 100);
```

```
document.write("Max Value : " + value);
```

```
var value = Math.min(10, 20, -1, 100);
```

```
document.write("
Min Value : " + value);
```

```
var value = Math.pow(7, 2);
```

```
document.write("
Square value : " + value);
```

```
var value = Math.sqrt(81);
```

```
document.write("
Square root Value : " + value);
```

```
var value = Math.tan(45);
```

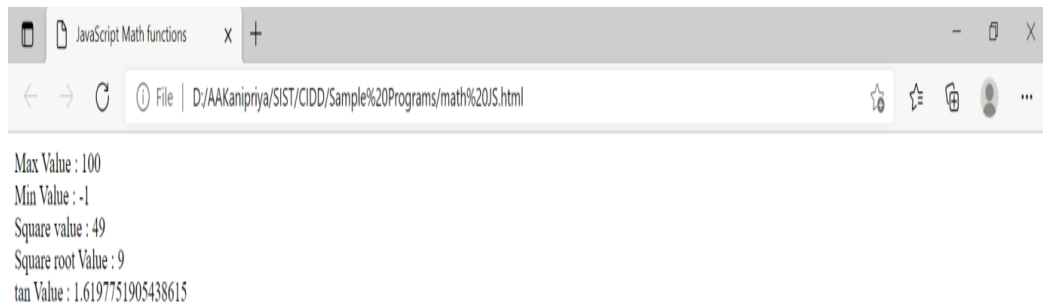
```
document.write("
tan Value : " + value);
```

```
</script>
```

```
</body>
```

```
</html>
```

**Output**



## Window Object

- Window object is a top-level object in Client-Side JavaScript.
- Window object represents the browser's window.
- It represents an open window in a browser.
- It supports all browsers.
- The document object is a property of the window object. So, typing window.document.write is same as document.write.
- All global variables are properties and functions are methods of the window object.

**Table 2.12 WindowObject – Properties and Description**

Property	Description
Document	It returns the document object for the window (DOM).
Frames	It returns an array of all the frames including iframes in the current window.
Closed	It returns the boolean value indicating whether a window has been closed or not.
History	It returns the history object for the window.
innerHeight	It sets or returns the inner height of a window's content area.
innerWidth	It sets or returns the inner width of a window's content area.
Length	It returns the number of frames in a window.
Location	It returns the location object for the window.
Name	It sets or returns the name of a window.
Navigator	It returns the navigator object for the window.
Opener	It returns a reference to the window that created the window.
outerHeight	It sets or returns the outer height of a window, including toolbars/scrollbars.

outerWidth	It sets or returns the outer width of a window, including toolbars/scrollbars.
Parent	It returns the parent window of the current window.
Screen	It returns the screen object for the window.
screenX	It returns the X coordinate of the window relative to the screen.
screenY	It returns the Y coordinate of the window relative to the screen.
Self	It returns the current window.
Status	It sets the text in the status bar of a window.
Top	It returns the topmost browser window that contains frames.

## Window Object Method

**Table 2.13 WindowObject – Methods and Description**

Method	Description
alert()	It displays an alert box.
confirm()	It displays a dialog box.
prompt()	It displays a dialog box that prompts the visitor for input.
setInterval()	It calls a function or evaluates an expression at specified intervals.
setTimeout()	It evaluates an expression after a specified number of milliseconds.
clearInterval()	It clears a timer specified by setInterval().
clearTimeOut()	It clears a timer specified by setTimeout().
close()	It closes the current window.
open()	It opens a new browser window.
createPopup()	It creates a pop-up window.
focus()	It sets focus to the current window.
blur()	It removes focus from the current window.
moveBy()	It moves a window relative to its current position.
moveTo()	It moves a window to the specified position.
resizeBy()	It resizes the window by the specified pixels.

resizeTo()	It resizes the window to the specified width and height.
print()	It prints the content of the current window.
scrollBy()	It scrolls the content by the specified number of pixels.
scrollTo()	It scrolls the content to the specified coordinates.

Example : Simple Program on Window Object

**open\_window.html      //File name**

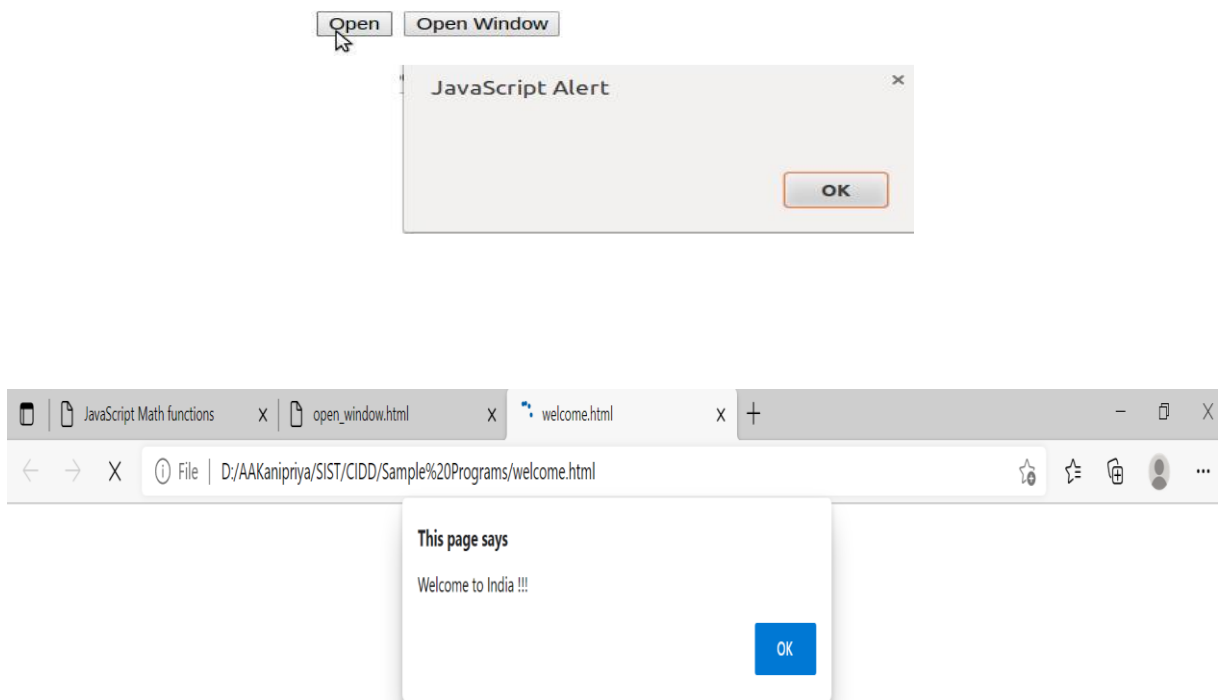
```
<html>
<head>
 <script type="text/javascript">
 function openwindow()
 {
 window.open("welcome.html");
 }
 </script>
</head>
<body>
 <form>
 <input type="button" value="Open" onClick=window.alert()>
 <input type="button" onClick="openwindow()" value="Open Window">
 </form>
</body>
</html>
```

**welcome.html      //File name**

```
<html>
 <body>
 <script type="text/javascript">
 {
 document.write("Welcome to India !!!");
 }
 </script>
```

```
</body>
</html>
```

### Output:



- In the above JavaScript program, when you click on the 'Open Window', you will see the 'welcome.html' opened in another window.
- When you click on the 'Open' button, you will see the alert message box.

### Navigator Object

- Navigator object is the representation of Internet browser.
- It contains all the information about the visitor's (client) browser.

**Table 2.14 Navigator Object – Properties and Description**

Property	Description
appName	It returns the name of the browser.
appCodeName	It returns the code name of the browser.



appVersion	It returns the version information of the browser.
cookieEnabled	It determines whether cookies are enabled in the browser.
platform	It returns which platform the browser is compiled.
userAgent	It returns the user agent header sent by the browser to the server.

## Navigator Object Methods

**Table 2.15 Navigator Object – Method and Description**

Method	Description
javaEnabled()	It specifies whether or not the browser is Java enabled.

Example : Simple Program on Navigator Object

```
<html>
 <body>
 <script type="text/javascript">
 document.write("Browser: " + navigator.appName + "

");
 document.write("Browser Version: " + navigator.appVersion + "

");
 document.write("Browser Code: " + navigator.appCodeName + "

");
 document.write("Platform: " + navigator.platform + "

");
 document.write("Cookie Enabled: " + navigator.cookieEnabled + "

");
 document.write("User Agent: " + navigator.userAgent + "

");
 document.write("Java Enabled: " + navigator.javaEnabled() + "

");
 </script>
 </body>
</html>
```

### OUTPUT

### History Object

- History object is

```
Browser: Netscape
Browser Version: 5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.159
Safari/537.36
Browser Code: Mozilla
Platform: Win32
Cookie Enabled: true
User Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.159
Safari/537.36
Java Enabled: false
```

- It is accessed through the window.history property.
- It contains the information of the URLs visited by the user within a browser window.

### History Object Properties

**Table 2.16 History Object – Properties and Description**

Property	Description
Length	It returns the number of URLs in the history list.
Current	It returns the current document URL.
Next	It returns the URL of the next document in the History object.
Previous	It returns the URL of the last document in the history object.

### History Object Methods

**Table 2.17 History Object – Methods and Description**

Method	Description
back()	It loads the previous URL in the history list.
forward()	It loads the next URL in the history list.
go("URL")	It loads a specific URL from the history list.

### Location Object

- Location object is a part of the window object.
- It is accessed through the '**window.location**' property.
- It contains the information about the current URL.

### Location Object Properties

**Table 2.18 Location Object – Properties and Description**

Property	Description
hash	It returns the anchor portion of a URL.
host	It returns the hostname and port of a URL.
hostname	It returns the hostname of a URL.

href	It returns the entire URL.
pathname	It returns the path name of a URL.
port	It returns the port number the server uses for a URL.
protocol	It returns the protocol of a URL.
search	It returns the query portion of a URL.

## Location Object Methods

**Table 2.19 Location Object – Method and Description**

Method	Description
assign()	It loads a new document.
reload()	It reloads the current document.
replace()	It replaces the current document with a new one.

### Example : Simple Program on Location Object

```
<html>
 <body>
 <script type="text/javascript">
 document.write("Path Name: " + location.pathname + "

");
 document.write("Href: " + location.href + "

");
 document.write("Protocol: " + location.protocol + "

");
 </script>
 </body>
</html>
```

### Output:

#### **Path Name:**

/home/tutorialride2/location\_object.  
html

#### **Href:**

file:///home/tutorialride2/location\_o  
bject.html

Protocol: file:

## Frame Object

- Frame object represents an HTML frame which defines one particular window(frame) within a frameset.
- It defines the set of frame that make up the browser window.
- It is a property of the window object.

**Syntax:**<frame>

- It has no end tag but they need to be closed properly.
- It is an HTML element.
- It defines a particular area in which another HTML document can be displayed.
- A frame should be used within a <FRAMESET> tag.

### <FRAME> Tag Attributes

**Table 2.20 Frame Object – Attribute and Description**

Attribute	Description
src	It is used to give the file name that should be located in the frame. Its value can be any URL. <b>For example:</b> src= “/html/abc.html”
name	It allows you to give a name to a frame. This attribute is used to indicate that a document should be loaded into a frame.
frameborder	It specifies whether or not the borders of that frame are shown. This attribute overrides the value given in the frameborder attribute on the tag if one is given. This can take values either 1 (Yes) or 0 (No).
marginwidth	It allows you to specify the width of the space between the left and right of the frame's border and the content. The value is given in pixels. <b>For example:</b> marginwidth = “10”.

marginheight	It allows you to specify the height of the space between the top and bottom of the frame's borders and its contents. The value is given in pixels. <b>For example:</b> marginheight = "10".
noresize	By default, you can resize any frame by clicking and dragging on the borders of a frame. It prevents a user from being able to resize the frame. <b>For example:</b> noresize = "noresize".
scrolling	It controls the appearance of the scrollbars that appear on the frame. It takes values either "Yes", "No" or "Auto". <b>For example:</b> scrolling = "no" means it should not have scroll bars.
longdesc	It allows you to provide a link to another page which contains a long description of the contents of the frame. <b>For example:</b> longdesc = "framedescription.html"

## Frameset Object

- Frameset object holds two or more frame elements and each frame element in turn holds a separate document.
- This object states only how many columns or rows there will be in the frameset.

### Syntax:

<frameset> . . . </frameset>

### <FRAMESET> Tag Attributes

**Table 2.21 Frameset Object - Attribute and Description**

Attribute	Description
cols (Columns)	It specifies how many columns are to be contained in the frameset and the size of each column.
rows	It works like the 'cols' attribute and takes the same values, but it is used to specify the number of rows in the frameset.
border	It specifies the width of the border of each frame in pixels. For example; border = "5". A value of zero means no border.
frame	It specifies whether a three-dimensional border should be displayed between frames. It takes value either 1 (Yes) or 0 (No). <b>For example:</b> frameborder = "0" specifies no border.
framespacing	It specifies the amount of space between frames in a frameset. It can take any integer value. <b>For example:</b> framespacing = "10" means there should be 10 pixels spacing between each frame.

### Example : Frameset Tag

```
<frameset cols="50%,50%">
 <frame src="http://www.tutorialride.com" />
 <frame src="http://www.tutorialride.com/javascript/javascript-dom-frame-object.htm" />
</frameset>
```

### Output:



### IFrame Object

IFrame object represents an HTML inline frame that contains another document.

### Frame Object Properties

**Table 2.22 IFrame Object - Properties and Description**

Property	Description
Align	It aligns the iframe.
contentDocument	It returns the document object generated by a frame/iframe.
contentWindow	It returns the window object generated by a frame/iframe.
frameBorder	It sets frame border in a frame/iframe.
Height	It sets the height in an iframe.

longDesc	It sets the value of the longdesc attribute in a frame/iframe.
marginHeight	It sets the margin height in a frame/iframe.
marginWidth	It sets the margin width in a frame/iframe.
Name	It specifies the name to the frame.
noResize	It disables the frame resizing capability.
scrolling	It controls the appearance of horizontal and vertical scrollbars in a frame.
Src	It indicates the location of the web page to be loaded into the frame.
Width	It sets or returns the value of the width attribute in an iframe.

## Link Object

- Link object represents an HTML link element.
- It is used to link to external stylesheets.
- It must be placed inside the head section of an HTML document.
- It specifies a link to an external resources.

## Link Object Properties

**Table 2.23 Link Object - Properties and Description**

Property	Description
href	It sets the URL of a linked document.
host	It sets the URL host name and port.
hostname	It sets the URL hostname.
pathname	It sets the URL pathname.
Port	It sets the URL port section.
Protocol	It sets the URL protocol section including the colon (:) after protocol name.
Search	It sets the URL query string section that is after including the questions mark (?).
Target	It sets the URL link's target name.

## Form Object

- Form object represents an HTML form.
- It is used to collect user input through elements like text fields, check box and radio button, select option, text area, submit buttons and etc.

**Syntax:**

<form> . . . </form>

**Form Object Properties**

**Table 2.24 Form Object - Properties and Description**

Property	Description
Action	It sets and returns the value of the action attribute in a form.
enctype	It sets and returns the value of the enctype attribute in a form.
Length	It returns the number of elements in a form.
Method	It sets and returns the value of the method attribute in a form that is GET or POST.
Name	It sets and returns the value of the name attribute in a form.
Target	It sets and returns the value of the target attribute in a form.

**Form Object Methods**

**Table 2.25 Form Object - Methods and Description**

Method	Description
reset()	It resets a form.
submit()	It submits a form.

**Hidden Object**

- Hidden object represents a hidden input field in an HTML form and it is invisible to the user.
- This object can be placed anywhere on the web page.
- It is used to send hidden form of data to a server.

**Syntax:**

<input type= “hidden”>

**Table 2.26 Hidden Object - Property and Description**

Property	Description
Name	It sets and returns the value of the name attribute of the hidden input field.
Type	It returns type of a form element.



Value	It sets or returns the value of the value attribute of the hidden input field.
-------	--------------------------------------------------------------------------------

## Password Object

- Password object represents a single-line password field in an HTML form.
- The content of a password field will be masked – appears as spots or asterisks in the browser using password object.

### Syntax:

<input type= “password”>

## Password Object Properties

**Table 2.27 Password Object - Property and Description**

Property	Description
defaultValue	It sets or returns the default value of a password field.
maxLength	It sets or returns the maximum number of characters allowed in a password field.
Name	It sets or returns the value of the name attribute of a password field.
readOnly	It sets or returns whether a password field is read only or not.
Size	It sets or returns the width of a password field.
Value	It sets or returns the value of the attribute of the password field.

## Password Object Methods

**Table 2.28 Password Object - Method and Description**

Method	Description
select()	It selects the content of a password field.

## Checkbox Object

- Check box object represents a checkbox in an HTML form.
- It allows the user to select one or more options from the available choices.

### Syntax:

<input type= “checkbox”>

## Checkbox Object Properties

**Table 2.29 Checkbox Object - Property and Description**

Property	Description
Name	It sets or returns the name of the checkbox.
Type	It returns the value “check”.
Value	It sets or returns the value of the attribute of a checkbox.
checked	It sets or returns the checked state of a checkbox.
defaultChecked	It returns the default value of the checked attribute.

### Checkbox Object Methods

**Table 2.30 Checkbox Object – Method and Description**

Method	Description
click()	It sets the checked property.

### Select Object

- Select object represents a dropdown list in an HTML form.
- It allows the user to select one or more options from the available choices.

#### Syntax:

<select> ... </select>

### Select Object Collections

**Table 2.31 Select Object – Collection and Description**

Collection	Description
options	It returns a collection of all the options in a dropdown list.

### Select Object Properties

**Table 2.32 Select Object – Property and Description**

Property	Description
Length	It returns the number of options in a dropdown list.
selectedIndex	It sets or returns the index of the selected option in a dropdown list.

Type	It returns a type of form element.
name	It returns the name of the selection list.

## Select Object Methods

**Table 2.33 Select Object – Method and Description**

Method	Description
add()	It adds an option to a dropdown list.
remove()	It removes an option from a dropdown list.

## Option Object

- Option object represents an HTML <option> element.
- It is used to add items to a select element.

### Syntax:

<option value> . . . </option>

**Table 2.34 Option Object – Property and Description**

Property	Description
Index	It sets or returns the index position of an option in a dropdown list.
Text	It sets or returns the text of an option element.
defaultSelected	It determines whether the option is selected by default.
Value	It sets or returns the value to the server if the option was selected.
Prototype	It is used to create additional properties.

## Option Object Methods

**Table 2.35 Option Object – Methods and Description**

Methods	Description
blur()	It removes the focus from the option.
focus()	It gives the focus to the option.

### Example : Simple Program on Option Object Method

```
<html>
<head>
 <script type="text/javascript">
 function optionfruit(select)
 {
 var a = select.selectedIndex;
 var fav = select.options[a].value;
 if(a==0)
 {
 alert("Please select a fruit");
 }
 else
 {
 document.write("Your Favorite Fruit is "+fav+".");
 }
 }
 </script>
</head>
<body>
 <form>
 List of Fruits:
 <select name="fruit">
 <option value="0">Select a Fruit</option>
 <option value="Mango">Mango</option>
 <option value="Apple">Apple</option>
 <option value="Banana">Banana</option>
 <option value="Strawberry">Strawberry</option>
 <option value="Orange">Orange</option>
 </select>
 <input type="button" value="Select" onClick="optionfruit(this.form.fruit);">
 </form>
</body>
</html>
```

### Output:

List of Fruits:

**Mango**

Apple  
Banana  
Strawberry  
Orange

Your Favorite Fruit is **Mango**.

### Radio Object

Radio object represents a radio button in an HTML form.

#### Syntax:

<input type= “radio”>

### Radio Object Properties

**Table 2.36 Radio Object – Property and Description**

Property	Description
Checked	It sets or returns the checked state of a radio button.
defaultChecked	Returns the default value of the checked attribute.
Name	It sets or returns the value of the name attribute of a radio button.
Type	It returns the type of element which is radio button.
Value	It sets or returns the value of the radio button.

### Radio Object Methods

**Table 2.37 Radio Object – Property and Description**

Method	Description
--------	-------------

blur()	It takes the focus away from the radio button.
click()	It acts as if the user clicked the button.
focus()	It gives the focus to the radio button.

## Text Object

Text object represents a single-line text input field in an HTML form.

### Syntax:

```
<input type= "text">
```

## Text Object Properties

**Table 2.38 Text Object – Property and Description**

Property	Description
Value	It sets or returns the value of the text field.
defaultValue	It sets or returns the default value of a text field.
Name	It sets or returns the value of the name attribute of a text field.
maxLength	It sets or returns the maximum number of characters allowed in a text field.
readOnly	It sets or returns whether a text field is read-only or not.
Size	It sets or returns the width of a text field.
Type	It returns type of form element of a text field.

## Textarea Object

Textarea object represents a text-area in an HTML form.

### Syntax:

```
<textarea> . . . </textarea>
```

## Textarea Object Properties

**Table 2.39 Textarea Object – Property and Description**

Property	Description
----------	-------------

Value	It sets or returns the value of the text field.
defaultValue	It sets or returns the default value of a text field.
Name	It sets or returns the value of the name attribute of a text field.
Type	It returns type of form element of a text field.
Rows	It displays the number of rows in a text area.
Cols	It displays the number of columns in a text area.

## What is AJAX?

AJAX = **A**synchronous **J**avaScript **A**nd **X**ML.

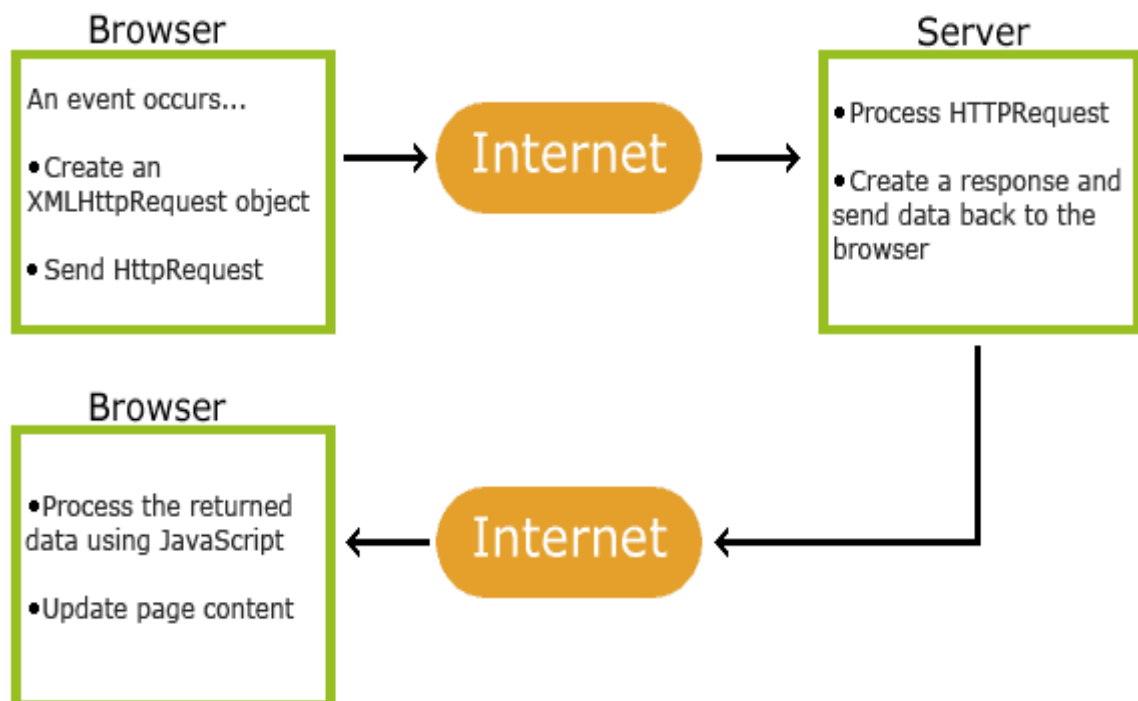
AJAX is not a programming language.

AJAX just uses a combination of:

- A browser built-in XMLHttpRequest object (to request data from a web server)
- JavaScript and HTML DOM (to display or use the data)

AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

## How AJAX Works



**Fig. 2.8 Working Process of AJAX**

1. An event occurs in a web page (the page is loaded, a button is clicked)
2. An XMLHttpRequest object is created by JavaScript
3. The XMLHttpRequest object sends a request to a web server
4. The server processes the request
5. The server sends a response back to the web page
6. The response is read by JavaScript
7. Proper action (like page update) is performed by JavaScript

## **The XMLHttpRequest Object**

All modern browsers support the XMLHttpRequest object.

The XMLHttpRequest object can be used to exchange data with a server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page

## **Create an XMLHttpRequest Object**

All modern browsers (Chrome, Firefox, IE7+, Edge, Safari Opera) have a built-in XMLHttpRequest object.

Syntax for creating an XMLHttpRequest object:

```
variable = new XMLHttpRequest();
```

Example

```
var xhttp = new XMLHttpRequest();
```

Old Versions of Internet Explorer (IE5 and IE6)

Old versions of Internet Explorer (IE5 and IE6) use an ActiveX object instead of the XMLHttpRequest object:

```
variable = new ActiveXObject("Microsoft.XMLHTTP");
```



To handle IE5 and IE6, check if the browser supports the XMLHttpRequest object, or else create an ActiveX object:

#### Example

```
if (window.XMLHttpRequest) {
 // code for modern browsers
 xmlhttp = new XMLHttpRequest();
} else {
 // code for old IE browsers
 xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
}
```

#### XMLHttpRequest Object Methods

**Table 2.40 XMLHttpRequest Object – Method and Description**

Method	Description
new XMLHttpRequest()	Creates a new XMLHttpRequest object
abort()	Cancels the current request
getAllResponseHeaders()	Returns header information
getResponseHeader()	Returns specific header information
open( <i>method,url,async,user,psw</i> )	Specifies the request  <i>method</i> : the request type GET or POST <i>url</i> : the file location <i>async</i> : true (asynchronous) or false (synchronous)

	<i>user</i> : optional user name <i>psw</i> : optional password
send()	Sends the request to the server Used for GET requests
send( <i>string</i> )	Sends the request to the server. Used for POST requests
setRequestHeader()	Adds a label/value pair to the header to be sent

## XMLHttpRequest Object Properties

**Table 2.41 XMLHttpRequest Object – Property and Description**

Property	Description
onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready

responseText	Returns the response data as a string
responseXML	Returns the response data as XML data
status	Returns the status-number of a request 200: "OK" 403: "Forbidden" 404: "Not Found" For a complete list go to the <a href="#">Http Messages Reference</a>
statusText	Returns the status-text (e.g. "OK" or "Not Found")

The XMLHttpRequest object is used to exchange data with a server.

### Send a Request To a Server

To send a request to a server, we use the open() and send() methods of the XMLHttpRequest object:

```
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

**Table 2.42 XMLHttpRequest Object – Method and Description**

Method	Description
<code>open(<i>method</i>, <i>url</i>, <i>async</i>)</code>	Specifies the type of request  <i>method</i> : the type of request: GET or POST

	<i>url</i> : the server (file) location <i>async</i> : true (asynchronous) or false (synchronous)
send()	Sends the request to the server (used for GET)
send( <i>string</i> )	Sends the request to the server (used for POST)

## GET or POST?

GET is simpler and faster than POST, and can be used in most cases.

However, always use POST requests when:

- A cached file is not an option (update a file or database on the server).
- Sending a large amount of data to the server (POST has no size limitations).
- Sending user input (which can contain unknown characters), POST is more robust and secure than GET.

## POST Requests

A simple POST request:

Example

```
xhttp.open("POST", "demo_post.asp", true);
xhttp.send();
```

## AJAX Example

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<div id="demo">
```

```
<h1>The XMLHttpRequest Object</h1>
```

```
<button type="button" onclick="loadDoc()">Change Content</button>
```

```
</div>
```

```
<script>
```

```
function loadDoc() {
```

```
 var xhttp = new XMLHttpRequest();
```

```
 xhttp.onreadystatechange = function() {
```

```
 if (this.readyState == 4 && this.status == 200) {
```

```
 document.getElementById("demo").innerHTML =
```

```
 this.responseText;
```

```
 };
```

```
 xhttp.open("GET", "ajax_info.txt", true);
```

```
 xhttp.send();
```

```
}
```

```
</script>
```

```
</body>
```

```
</html>
```

ajax\_info.txt

```
<h1>AJAX</h1>
```

```
<p>AJAX is not a programming language.</p>
```

```
<p>AJAX is a technique for accessing web servers from a web page.</p>
```

```
<p>AJAX stands for Asynchronous JavaScript And XML.</p>
```

## Output

# The XMLHttpRequest Object

Change Content

The output will look like the following after the change content button has been clicked.

p (which

## AJAX

AJAX is not a programming language.

AJAX is a technique for accessing web servers from a web page.

AJAX stands for Asynchronous JavaScript And XML.

```
xhttp.open("GET", "ajax_test.asp", true);
```

By sending asynchronously, the JavaScript does not have to wait for the server response, but can instead:

- execute other scripts while waiting for server response
- deal with the response after the response is read

## The onreadystatechange Property

With the XMLHttpRequest object you can define a function to be executed when the request receives an answer.

The function is defined in the **onreadystatechange** property of the XMLHttpRequest object:

```
xhttp.onreadystatechange = function() {
 if (this.readyState == 4 && this.status == 200) {
 document.getElementById("demo").innerHTML = this.responseText;
 }
};
```

```
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

## Synchronous Request

To execute a synchronous request, change the third parameter in the open() method to false:

```
xhttp.open("GET", "ajax_info.txt", false);
```

Sometimes `async = false` are used for quick testing. You will also find synchronous requests in older JavaScript code.

Since the code will wait for server completion, there is no need for an `onreadystatechange` function:

### Example

```
xhttp.open("GET", "ajax_info.txt", false);
xhttp.send();
document.getElementById("demo").innerHTML = xhttp.responseText;
```

## AJAX - Server Response

### The `onreadystatechange` Property

The **readyState** property holds the status of the XMLHttpRequest.

The **onreadystatechange** property defines a function to be executed when the `readyState` changes.

The **status** property and the **statusText** property holds the status of the XMLHttpRequest object.

**Table 2.43 XMLHttpRequest Object – Property and Description**

Property	Description
onreadystatechange	Defines a function to be called when the <code>readyState</code> property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established

	2: request received 3: processing request 4: request finished and response is ready
status	200: "OK" 403: "Forbidden" 404: "Page not found" For a complete list go to the <a href="#">Http Messages Reference</a>
statusText	Returns the status-text (e.g. "OK" or "Not Found")

The onreadystatechange function is called every time the readyState changes.

When readyState is 4 and status is 200, the response is ready:

#### Example

```
function loadDoc() {
 var xhttp = new XMLHttpRequest();
 xhttp.onreadystatechange = function() {
 if (this.readyState == 4 && this.status == 200) {
 document.getElementById("demo").innerHTML =
 this.responseText;
 }
 };
 xhttp.open("GET", "ajax_info.txt", true);
 xhttp.send();
}
```

#### Server Response Properties

**Table 2.44 XMLHttpRequest Object – Property and Description**

Property	Description
----------	-------------



responseText	get the response data as a string
responseXML	get the response data as XML data

## Server Response Methods

**Table 2.45 XMLHttpRequest Object – Method and Description**

Method	Description
getResponseHeader()	Returns specific header information from the server resource
getAllResponseHeaders()	Returns all the header information from the server resource

## The responseText Property

The **responseText** property returns the server response as a JavaScript string, and you can use it accordingly:Example

```
document.getElementById("demo").innerHTML = xhttp.responseText;
```

## The responseXML Property

## Example

Request the file [cd\\_catalog.xml](#) and parse the response:

```
<!DOCTYPE html>
<html>
<body>
<h2>The XMLHttpRequest Object</h2>
<p id="demo"></p>

<script>
const xhttp = new XMLHttpRequest();
xhttp.onload = function() {
 const xmlDoc = this.responseXML;
 const x =
xmlDoc.getElementsByTagName("ARTIST");
 let txt = "";

for (let i = 0; i < x.length; i++) {

 txt = txt + x[i].childNodes[0].nodeValue + "
";

}

document.getElementById("demo").innerHTML = txt;

}

xhttp.open("GET", "cd_catalog.xml");

xhttp.send();

</script>

</body></html>
```

## Output

### The XMLHttpRequest Object

Bob Dylan  
Bonnie Tyler  
Dolly Parton  
Gary Moore  
Eros Ramazzotti  
Bee Gees  
Dr.Hook  
Rod Stewart  
Andrea Bocelli  
Percy Sledge  
Savage Rose  
Many  
Kenny Rogers  
Will Smith  
Van Morrison  
Jorn Hoel  
Cat Stevens  
Sam Brown

## QUESTION BANK

S.No	Questions (2 marks)	Competence	BT Level
16.	Write the JavaScript to print “Good Day” using IF-ELSE condition.	Create	BTL5
17.	What is a JavaScript statement? Give an example.	Knowledge	BTL1

18.	Write some advantages of using JavaScript.	Create	BTL2
19.	List the JavaScript datatypes.	Knowledge	BTL1
20.	Differentiate the return and continue statements in JavaScript	Analyze	BTL4
21.	Construct the statement to read the properties of an object in javascript.	Create	BTL6
22.	Identify the built-in method returns the character at the specified index?	Understand	BTL2
23.	List out the properties to determine the size of the browser window.	Knowledge	BTL1
24.	Define Java Script.	Knowledge	BTL1
25.	Write the syntax to create an object in javascript.	Create	BTL6
26.	Express the statement in java script to display content on webpage.	Understand	BTL2
27.	Connect the role of a callback function in performing a partial page update in an AJAX application.	Analyze	BTL4
28.	Create an appropriate Javascript code to remove an element (current element) from a DOM.	Create	BTL6
29.	Write down the Javascript code to swap the contents of the two text boxes  <form name="form1">  Input text 1: <input type="text" name="A">  Input text 2: <input type="text" name="B">  </form>	Create	BTL6
30.	Summarize benefits of using Javascript code in an HTML document.	Understand	BTL2

S.No	Questions (16 marks)	Competence	BT Level
------	----------------------	------------	----------

11.	Explain how local and global functions can be written using JavaScript with example	Understand	BTL2
12.	Write JavaScript to find sum of first 'n' even numbers and display the result. Get the value of 'n' from user.	Create	BTL6
13.	Write JavaScript to find factorial of a given number.	Create	BTL6
14.	Show in detail about JavaScript variables and operators.	Apply	BTL3
15.	Classify the various kinds of javascript pop up boxes with example programs.	Analyze	BTL4
16.	Explain about Math object and the various methods in javascript with example program.	Understand	BTL2
17.	Demonstrate the working process of AJAX. Develop the program to retrieve the names of the artists from an XML file.	Apply	BTL3
18.	Summarize the methods associated with array object in JavaScript. Explain any two methods with an example.	Evaluate	BTL5
19.	<p>Develop a javascript program to perform the email validation. Assume the following criteria to validate the email id.</p> <ul style="list-style-type: none"> <li>i. email id must contain the @ and . character</li> <li>ii. There must be at least one character before and after the @.</li> <li>iii. There must be at least two characters after . (dot).</li> </ul>	Create	BTL6
20.	<p>Use Javascript and HTML to create a page with two panes.</p> <p>The first pane (on the left) should have a text area where HTML code can be typed by the user. The pane on the right side should display the preview of the HTML code typed by the user, as it would be seen on the browser.</p>	Apply	BTL3



**SATHYABAMA**

INSTITUTE OF SCIENCE AND TECHNOLOGY  
(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

[www.sathyabama.ac.in](http://www.sathyabama.ac.in)

## UNIT – III – Customer Interface Design and Development – SITA1502

## **UNIT 3 SERVER-SIDE SCRIPTING**

Introduction To PHP – Data Types – Control Structures – Arrays - Function – HTML Form with PHP – Form Handling and Validation - File Handling – Cookies – Sessions – Filters – Exception Handling - Database Connectivity with MySQL.

### **Introduction to PHP**

- Acronym for “PHP: Hypertext Preprocessor”.
- PHP is a widely-used, open-source server-side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.
- It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.
- PHP Syntax is C-Like.
- PHP scripts are executed on the server
- PHP is free to download and use

### **Common uses of PHP**

- PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them.
- PHP can handle forms, i.e. gather data from files, save data to a file, through email you can send data, return data to the user.
- You add, delete, modify elements within your database through PHP.
- Access cookies variables and set cookies.
- Using PHP, you can restrict users to access some pages of your website.
- It can encrypt data.

### **What is a PHP File?**

- PHP files can contain text, HTML, CSS, JavaScript, and PHP code
- PHP code is executed on the server, and the result is returned to the browser as plain HTML
- PHP files have extension ".php"

### **What Can PHP Do?**

- PHP can generate dynamic page content
- PHP can create, open, read, write, delete, and close files on the server
- PHP can collect form data
- PHP can send and receive cookies
- PHP can add, delete, modify data in your database
- PHP can be used to control user-access
- PHP can encrypt data

With PHP you are not limited to output HTML. You can output images, PDF files, and even Flash movies. You can also output any text, such as XHTML and XML.

## Why PHP?

- PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP supports a wide range of databases
- PHP is free. Download it from the official PHP resource: [www.php.net](http://www.php.net)
- PHP is easy to learn and runs efficiently on the server side

## Example: - "Hello World" Script in PHP

```
<html>
 <head>
 <title>Hello World</title>
 </head>
 <body>
 <?php echo "Hello, World!";?>
 </body>
</html>
```

## Output:

Hello, World!

All PHP code must be included inside one of the three special markup tags which are recognised by the PHP Parser.

```
<?php PHP code goes here ?>
<? PHP code goes here ?>
<script language = "php"> PHP code goes here </script>
A most common tag is the <?php...?>
```

## PHP Variables

- All php variables are denoted with a leading dollar sign (\$).
- The value of a variable is the value of its most recent assignment.
- Variables are assigned with the = operator, with the variable on the left-hand side and the expression to be evaluated on the right.
- Variables can, but do not need, to be declared before assignment.
- Variables in PHP do not have intrinsic types - a variable does not know in advance whether it will be used to store a number or a string of characters.
- Variables used before they are assigned have default values.
- PHP does a good job of automatically converting types from one to another when necessary.

## PHP Data Types

Variables can store data of different types, and different data types can do different things.

PHP has a total of eight data types which we use to construct our variables –

- **Integers** – are whole numbers, without a decimal point, like 4195.
- **Doubles** – are floating-point numbers, like 3.14159 or 49.1.
- **Booleans** – have only two possible values either true or false.
- **NULL** – is a special type that only has one value: NULL.
- **Strings** – are sequences of characters, like 'PHP supports string operations.'
- **Arrays** – are named and indexed collections of other values.
- **Objects** – are instances of programmer-defined classes, which can package up both other kinds of values and functions that are specific to the class.
- **Resources** – are special variables that hold references to resources external to PHP (such as database connections).

The first five are *simple types*, and the next two (arrays and objects) are compound - the compound types can package up other arbitrary values of arbitrary type, whereas the simple types cannot.

## 1. Integers

They are whole numbers, without a decimal point, like 4195. They are the simplest type .they correspond to simple whole numbers, both positive and negative. Integers can be assigned to variables, or they can be used in expressions as following,

```
$int_var = 12345;
$another_int = -12345 + 12345;
```

## 2. Doubles

They like 3.14159 or 49.1. By default, doubles print with the minimum number of decimal places needed. For example, the code –

**Example:**

```
<?php
$a = 2.2888800;
$b = 2.2111200;
$c = $a + $b;
print("$a + $b = $c
");
?>
```

**Output:**

2.28888 + 2.21112 = 4.5

## 3. Boolean



They have only two possible values either true or false. PHP provides a couple of constants especially for use as Booleans: TRUE and FALSE, which can be used like so –

```
if (TRUE)
 print("This will always print
");

else
 print("This will never print
");
```

Each of the following variables has the truth value embedded in its name when it is used in a Boolean context.

```
$true_num = 3 + 0.14159;
>true_str = "Tried and true"
>true_array[49] = "An array element";
>false_array = array();
>false_null = NULL;
>false_num = 999 - 999;
>false_str = "";
```

#### 4. NULL

NULL is a special type that only has one value: NULL. To give a variable the NULL value, simply assign it like this –

```
$my_var = null;
 Or
$my_var = NULL;
```

#### 5. String

A string is a sequence of characters, like "Hello world!". A string can be any text inside quotes. You can use single or double quotes:

**Example:**

```
<?php
$x= "Helloworld!";
$y= 'Helloworld!';
echo $x;
echo "
";
echo $y;
?>
```

**Output:**

```
Helloworld!
Helloworld!
```

## 6. Array

An array stores multiple values in one single variable. In the following example \$cars is an array. The PHP var\_dump() function returns the data type and value:

### Example:

```
<html>
<body>
<?php
$cars = array("Volvo","BMW","Toyota");
var_dump($cars);
?>
</body>
</html>
```

### Output:

```
array(3) {
 [0]=>
 string(5) "Volvo"
 [1]=>
 string(3) "BMW"
 [2]=>
 string(6) "Toyota"
}
```

## 7. Object

- Classes and objects are the two main aspects of object-oriented programming.
- A class is a template for objects, and an object is an instance of a class.
- When the individual objects are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.
- Let's assume we have a class named Car. A Car can have properties like model, color, etc. We can define variables like \$model, \$color, and so on, to hold the values of these properties.
- When the individual objects (Volvo, BMW, Toyota, etc.) are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.
- If you create a \_\_construct() function, PHP will automatically call this function when you create an object from a class. Notice that the construct function starts with two underscores (\_\_)!

### Example:-

```
<?php
class Car {
```

```

public $color;
public $model;
public function construct($color, $model) {
 $this->color = $color;
 $this->model = $model;
}
public function message() {
 return "My car is a ". $this->color . " " . $this->model . "!";
}
}

```

```

$myCar = new Car("black", "Volvo");
echo $myCar -> message();
echo "
";
$myCar = new Car("red", "Toyota");
echo $myCar -> message();
?>

```

### **Output:**

```

My car is a black Volvo!
My car is a red Toyota!

```

## **8. Resource**

The special resource type is not an actual data type. It is the storing of a reference to functions and resources external to PHP. They are special variables that hold references to resources external to PHP (such as database connections). A common example of using the resource data type is a database call.

### **PHP Control Structures and Loops: if, else, for, foreach, while**

The control structure controls the flow of code execution in application. Generally, a program is executed sequentially, line by line, and a control structure allows you to alter that flow, usually depending on certain conditions.

Control structures are core features of the PHP language that allow script to respond differently to different inputs or situations. This could allow script to give different responses based on user input, file contents, or some other data.

PHP supports a number of different control structures:

- if
- else

- elseif
- switch
- while
- do-while
- for
- foreach

## 1. PHP If Statement

The 'if' construct allows you to execute a piece of code if the expression provided along with it evaluates to true.

**Example:**

```
<?php
$age = 50;
if ($age > 30)
{
 echo "Your age is greater than 30!";
}
?>
```

**Output:** Your age is greater than 30!

## 2. PHP Else Statement

The 'if' construct, which allows you to execute a piece of code if the expression evaluates to true. On the other hand, if the expression evaluates to false, it won't do anything. More often than not, you also want to execute a different code snippet if the expression evaluates to false. That's where the 'else' statement comes into the picture.

**Example:**

```
<?php
$age = 50;
if ($age < 30)
{
 echo "Your age is less than 30!";
}
```

```
else
{
 echo "Your age is greater than or equal to 30!";
}
?>
```

**Output:** Your age is greater than or equal to 30!

### 3. PHP Else If Statement

We can consider the elseif statement as an extension to the if-else construct. If there are more than two choices to choose from, we can use the elseif statement.

**Example:**

```
<?php
$age = 50;
if ($age < 30)
{
 echo "Your age is less than 30!";
}
elseif ($age > 30 && $age < 40)
{
 echo "Your age is between 30 and 40!";
}
elseif ($age > 40 && $age < 50)
{
 echo "Your age is between 40 and 50!";
}
else
{
 echo "Your age is greater than 50!";
}
?>
```

**Output :** Your age is greater than 50!

### 4. PHP Switch Statement

The switch statement is similar to a series of IF statements on the same expression. In many occasions, you may want to compare the same variable (or expression) with many different values, and execute a different piece of code depending on which value it equals to.

**Example:**

```
<?php
```

```
$favourite_site = 'Code';
switch ($favourite_site) {
 case 'Business':
 echo "My favourite site is business.tutsplus.com!";
 break;
 case 'Code':
 echo "My favourite site is code.tutsplus.com!";
 break;
 case 'Web Design':
 echo "My favourite site is webdesign.tutsplus.com!";
 break;
 case 'Music':
 echo "My favourite site is music.tutsplus.com!";
 break;
 case 'Photography':
 echo "My favourite site is photography.tutsplus.com!";
 break;
 default:
 echo "I like everything at tutsplus.com!";
}
?>
```

**Output:** My favourite site is code.tutsplus.com!

In the above example, check the value of the `$favourite_site` variable, and based on the value of the `$favourite_site` variable, print a message.

For each value you want to check with the `$favourite_site` variable, you have to define the case block. If the value is matched with a case, the code associated with that case block will be executed. After that, you need to use the `break` statement to end code execution. If you don't use the `break` statement, script execution will be continued up to the last block in the switch statement.

Finally, if you want to execute a piece of code if the variable's value doesn't match any case, you can define it under the `default` block. Of course, it's not mandatory—it's just a way to provide a default case.

## Loops in PHP

### 5. While Loop in PHP

The while loop is used when you want to execute a piece of code repeatedly until the while condition evaluates to false.

**Example:**

```
<?php
$max = 0;
echo $i = 0;
echo ",";
echo $j = 1;
echo ",";
$result=0;
while ($max < 10)
{
 $result = $i + $j;

 $i = $j;
 $j = $result;

 $max = $max + 1;
 echo $result;
 echo ",";
}
?>
```

It outputs the Fibonacci series for the first ten numbers.

**Output:**

0,1,1,2,3,5,8,13,21,34,55,89,

### 6. Do-While Loop in PHP

The do-while loop is very similar to the while loop, with the only difference being that the while condition is checked at the end of the first iteration. Thus, we can guarantee that the loop code is executed at least once, irrespective of the result of the while expression.

**Example:**

```
<?php
$handle = fopen("file.txt", "r");
if ($handle)
{
 do
 {
 $line = fgets($handle);
 // process the line content
 } while($line !== false);
}
```

```
}
fclose($handle);
?>
```

## 7. For Loop in PHP

Generally, the for loop is used to execute a piece of code a specific number of times. In other words, if you already know the number of times you want to execute a block of code, it's the for loop which is the best choice.

### Example:

```
<?php

for ($i=1; $i<=10; ++$i)
{
 echo sprintf("The square of %d is %d.
", $i, $i*$i);
}
?>
```

The above program outputs the square of the first ten numbers. It initializes \$i to 1, repeats as long as \$i is less than or equal to 10, and adds 1 to \$i at each iteration.

### Output:

The square of 1 is 1.  
The square of 2 is 4.  
The square of 3 is 9.  
The square of 4 is 16.  
The square of 5 is 25.  
The square of 6 is 36.  
The square of 7 is 49.  
The square of 8 is 64.  
The square of 9 is 81.  
The square of 10 is 100.

## 8. For Each in PHP

The foreach loop is used to iterate over array variables. If you have an array variable, and you want to go through each element of that array, the foreach loop is the best choice.

### Example:

```
<?php
```



```

$fruits = array('apple', 'banana', 'orange', 'grapes');
foreach ($fruits as $fruit)
{
 echo $fruit;
 echo "
";
}

$employee = array('name' => 'John Smith', 'age' => 30, 'profession' => 'Software Engineer');
foreach ($employee as $key => $value)
{
 echo sprintf("%s: %s
", $key, $value);
 echo "
";
}
?>

```

### **Output:**

```

apple
banana
orange
grapes
name: John Smith

age: 30

profession: Software Engineer

```

If you want to access array values, you can use the first version of the foreach loop, as shown in the above example. On the other hand, if you want to access both a key and a value, you can do it as shown in the \$employee example above.

## **PHP Arrays**

An array is a special variable, which can hold more than one value at a time. If you have a list of items (a list of car names, for example), the array can be created as follows:

### **Example:**

```

<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";

```

```
?>
```

**Output:** I like Volvo, BMW and Toyota.

## Create an Array in PHP

In PHP, the `array()` function is used to create an array. In PHP, there are three types of arrays:

- Indexed arrays - Arrays with a numeric index
- Associative arrays - Arrays with named keys
- Multidimensional arrays - Arrays containing one or more arrays

### The `count()` Function - Get The Length of an Array

The `count()` function is used to return the length (the number of elements) of an array:

#### Example:

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo count($cars);
?>
```

**Output:** 3

## 1. PHP Indexed Arrays

There are two ways to create indexed arrays. The index can be assigned automatically (index always starts at 0), like this:

```
$cars = array("Volvo", "BMW", "Toyota");
(Or)
```

The index can be assigned manually:

```
$cars[0] = "Volvo";
$cars[1] = "BMW";
$cars[2] = "Toyota";
```

#### Example:

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
```

**Output:** I like Volvo,BMW and Toyota

### Loop Through an Indexed Array:

To loop through and print all the values of an indexed array, you could use a for loop, like this:

**Example:**

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
$arlength = count($cars);
for($x = 0; $x < $arlength; $x++) {
 echo $cars[$x];
 echo "
";
}
?>
```

**Output:**

```
Volvo
BMW
Toyota
```

## 2. PHP Associative Arrays

Associative arrays are arrays that use named keys that you assign to them. There are two ways to create an associative array:

```
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
(Or)
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
The named keys can then be used in a script:
```

**Example**

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
echo "Peter is " . $age['Peter'] . " years old.";
?>
```

**Output:** Peter is 35 years old.

**Loop Through an Associative Array:**

To loop through and print all the values of an associative array, you could use a foreach loop, like this:

**Example:**

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
foreach($age as $x => $x_value) {
 echo "Key=" . $x . ", Value=" . $x_value;
 echo "
";
}
?>
```

**Output:**

Key=Peter,Value=35  
Key=Ben,Value=37  
Key=Joe, Value=43

### 3. PHP - Multidimensional Arrays

A multidimensional array is an array containing one or more arrays. PHP supports multidimensional arrays that are two, three, four, five, or more levels deep. However, arrays more than three levels deep are hard to manage for most people. The dimension of an array indicates the number of indices you need to select an element.

For a two-dimensional array, two indices are needed to select an element. For a three-dimensional array, three indices are needed to select an element.

#### PHP - Two-dimensional Arrays:

A two-dimensional array is an array of arrays (a three-dimensional array is an array of arrays of arrays). Table.3.1

**Table 3.1 Array information**

Name	Stock	Sold
Volvo	22	18
BMW	15	13
Saab	5	2
Land Rover	17	15

We can store the data from the table above in a two-dimensional array, like this:

```
$cars = array (
 array("Volvo",22,18),
 array("BMW",15,13),
 array("Saab",5,2),
 array("Land Rover",17,15)
);
```

Now the two-dimensional \$cars array contains four arrays, and it has two indices: row and column. To get access to the elements of the \$cars array we must point to the two indices (row and column):

#### Example:

```
<?php
echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2]."
";
echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2]."
";
echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2]."
";
echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2]."
";
?>
```

#### Output:

Volvo: In stock: 22, sold: 18.  
BMW: In stock: 15, sold: 13.  
Saab: In stock: 5, sold: 2.  
Land Rover: In stock: 17, sold: 15.

We can also put a for loop inside another for loop to get the elements of the \$cars array (we still have to point to the two indices):

**Example:**

```
<?php
for ($row = 0; $row < 4; $row++) {
 echo "<p>Row number $row</p>";
 echo "";
 for ($col = 0; $col < 3; $col++) {
 echo "".$cars[$row][$col]."";
 }
 echo "";
}
?>
```

**Output:**

```
Row number 0
Volvo
22
18
Row number 1
BMW
15
13
Row number 2
Saab
5
2
Row number 3
Land Rover
17
15
```

## PHP Sorting Arrays

The elements in an array can be sorted in alphabetical or numerical order, descending or ascending. The following are the PHP array sort functions:

1. `sort()` - Sort arrays in ascending order
2. `rsort()` - sort arrays in descending order
3. `asort()` - sort associative arrays in ascending order, according to the value
4. `ksort()` - sort associative arrays in ascending order, according to the key

5. `arsort()` - sort associative arrays in descending order, according to the value
6. `krsort()` - sort associative arrays in descending order, according to the key

### 1. `sort()`

The following example sorts the elements of the `$cars` array in ascending alphabetical order:

#### Example:-

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
sort($cars);
$length = count($cars);
for($x = 0; $x < $length; $x++) {
 echo $cars[$x];
 echo "
";
}
?>
```

#### Result:

```
BMW
Toyota
Volvo
```

### 2. `rsort()`

It sorts the elements of the `$cars` array in descending alphabetical order:

#### Example:-

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
rsort($cars);

$length = count($cars);
for($x = 0; $x < $length; $x++) {
 echo $cars[$x];
 echo "
";
}
?>
```

#### Result:

```
Volvo
Toyota
BMW
```

### 3. `asort()`

It sorts an associative array in ascending order, according to the value:

#### Example:-

```

<?php
$page = array("Peter"=>"37", "Ben"=>"35", "Joe"=>"29");
asort($page);
foreach($page as $x => $x_value) {
 echo "Key=" . $x . ", Value=" . $x_value;
 echo "
";
}
?>

```

**Result:**

```

Key=Joe, Value=29
Key=Ben, Value=35
Key=Peter, Value=37

```

#### 4. ksort()

The following example sorts an associative array in ascending order, according to the key:

**Example:-**

```

<?php
$page = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
ksort($page);

foreach($page as $x => $x_value) {
 echo "Key=" . $x . ", Value=" . $x_value;
 echo "
";
}
?>

```

**Result:**

```

Key=Ben, Value=37
Key=Joe, Value=43
Key=Peter, Value=35

```

#### 5. Sort Array (Descending Order), According to Value - arsort()

It sorts an associative array in descending order, according to the value: For the same example given above, the result for arsort() is as follows,

```

Key=Joe, Value=43
Key=Ben, Value=37
Key=Peter, Value=35

```

#### 6. Sort Array (Descending Order), According to Key - krsort()

It sorts an associative array in descending order, according to the key. For the above example, the result for krsort() is as follows,

Key=Peter, Value=35  
Key=Joe, Value=43  
Key=Ben, Value=37

## PHP Functions

PHP has more than 1000 built-in functions, and in addition we can create our own custom functions.

### PHP Built-in Functions

PHP has over 1000 built-in functions that can be called directly, from within a script, to perform a specific task.

#### PHP Date/Time Functions:

Function	Description
date_add()	Adds days, months, years, hours, minutes, and seconds to a date
date_diff()	Returns the difference between two dates
date_format()	Returns a date formatted according to a specified format

#### PHP String Functions:

Function	Description
print()	Outputs one or more strings
printf()	Outputs a formatted string
str_pad()	Pads a string to a new length
str_repeat()	Repeats a string a specified number of times

### PHP User Defined Functions

Besides the built-in PHP functions, it is possible to create your own functions.

- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute automatically when a page loads.
- A function will be executed by a call to the function.
- Create a User Defined Function in PHP
- A user-defined function declaration starts with the word function:

#### Syntax:

```
function functionName() {

 code to be executed;

}
```



Note: A function name must start with a letter or an underscore. Function names are NOT case-sensitive.

**Example:-**

```
<?php
function writeMsg() {
 echo "Hello world!";
}
writeMsg(); // call the function
?>
```

**Output:** Hello world!

## PHP Function Arguments

Information can be passed to functions through arguments. An argument is just like a variable. Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument (\$fname). When the familyName() function is called, we also pass along a name (e.g. Jani), and the name is used inside the function, which outputs several different first names, but an equal last name:

**Example:**

```
<?php
function familyName($fname) {
 echo "$fname
";
}

familyName("Jani");
familyName("Hege");
familyName("Stale");
familyName("Kai Jim");
familyName("Borge");
?>
```

**Result:**

```
Jani
Hege
Stale
Kai Jim
Borge
```

The following example has a function with two arguments (\$fname and \$year):

```
<?php
function familyName($fname, $year) {
 echo "$fname Born in $year
";
}
```

```
}
```

```
familyName("Hege","1975");
familyName("Stale","1978");
familyName("Kai Jim","1983");
?>
```

**Result:**

Hege Born in 1975

Stale Born in 1978

Kai Jim Born in 1983

**PHP is a Loosely Typed Language:**

In the example above, notice that we did not have to tell PHP which data type the variable is. PHP automatically associates a data type to the variable, depending on its value. Since the data types are not set in a strict sense, you can do things like adding a string to an integer without causing an error.

In PHP 7, type declarations were added. This gives us an option to specify the expected data type when declaring a function, and by adding the strict declaration, it will throw a "Fatal Error" if the data type mismatches.

In the following example we try to send both a number and a string to the function without using strict:

**Example:**

```
<?php
function addNumbers(int $a, int $b) {
 return $a + $b;
}
echo addNumbers(5, "5 days");
// since strict is NOT enabled "5 days" is changed to int(5), and it will return 10
?>
```

**Result: 10**

To specify strict we need to set `declare(strict_types=1);`. This must be on the very first line of the PHP file. In the following example we try to send both a number and a string to the function, but here we have added the strict declaration:

```
<?php declare(strict_types=1); // strict requirement
```

```
function addNumbers(int $a, int $b) {
 return $a + $b;
}

echo addNumbers(5, "5 days");

// since strict is enabled and "5 days" is not an integer, an error will be thrown
?>
```

**Result:**

PHP Fatal error: Uncaught TypeError:

**PHP Default Argument Value:**

The following example shows how to use a default parameter. If we call the function setHeight() without arguments it takes the default value as argument:

**Example:**

```
<?php

function setHeight(int $minheight = 50) {
 echo "The height is : $minheight
";
}
```

```
setHeight(350);
setHeight();
setHeight(135);
setHeight(80);
?>
```

**Result:**

The height is : 350

The height is : 50

The height is : 135

The height is : 80

**PHP Functions - Returning values**

To let a function return a value, use the return statement:

**Example:**

```
<?php
function sum(int $x, int $y) {
 $z = $x + $y;
 return $z;
}

echo "5 + 10 = " . sum(5,10) . "
";
echo "7 + 13 = " . sum(7,13) . "
";
echo "2 + 4 = " . sum(2,4);
?>
```

**Result:**

```
5 + 10 = 15
7 + 13 = 20
2 + 4 = 6
```

## PHP Return Type Declarations

PHP 7 also supports Type Declarations for the return statement. Like with the type declaration for function arguments, by enabling the strict requirement, it will throw a "Fatal Error" on a type mismatch.

To declare a type for the function return, add a colon ( : ) and the type right before the opening curly ( { ) bracket when declaring the function.

In the following example we specify the return type for the function:

**Example:**

```
<?php declare(strict_types=1); // strict requirement
function addNumbers(float $a, float $b) : float {
 return $a + $b;
}

echo addNumbers(1.2, 5.2);
```

?>

**Result:** 6.4

## Passing Arguments by Reference

In PHP, arguments are usually passed by value, which means that a copy of the value is used in the function and the variable that was passed into the function cannot be changed. When a function argument is passed by reference, changes to the argument also change the variable that was passed in. To turn a function argument into a reference, the & operator is used:

### Example

Use a pass-by-reference argument to update a variable:

```
<?php
function add_five(&$value) {
 $value += 5;
}
```

```
$num = 2;
add_five($num);
echo $num;
?>
```

**Result:** 7

## HTML Form with PHP

An HTML form is used to collect user input. The user input is most often sent to a server for processing. The PHP superglobals \$\_GET and \$\_POST are used to collect form-data.

### PHP - A Simple HTML Form

The example below displays a simple HTML form with two input fields and a submit button:

#### Example:

```
<html>
```

```
<body>

<form action="welcome.php" method="post">
Name: <input type="text" name="name">

E-mail: <input type="text" name="email">

<input type="submit">
</form>

</body>
</html>
```

**Result:**

Name:

E-mail:

When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "welcome.php". The form data is sent with the HTTP POST method. To display the submitted data you could simply echo all the variables. The "welcome.php" looks like this:

```
<html>

<body>

Welcome <?php echo $_POST["name"]; ?>

Your email address is: <?php echo $_POST["email"]; ?>

</body>
</html>
```

**Output:**

Welcome John

Your email address is john.doe@example.com

The code above is quite simple. However, the most important thing is missing. The form data need to be validated to protect the script from malicious code. Think **SECURITY** when processing PHP forms! This page does not contain any form validation, it just shows how to send and retrieve form data. However, the next topic will show how to process PHP forms with security in mind! Proper validation of form data is important to protect the form from hackers and spammers!

## GET vs. POST

Both GET and POST create an array (e.g. `array( key1 => value1, key2 => value2, key3 => value3, ...)`). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.

Both GET and POST are treated as `$_GET` and `$_POST`. These are superglobals, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

`$_GET` is an array of variables passed to the current script via the URL parameters.

`$_POST` is an array of variables passed to the current script via the HTTP POST method.

### When to use GET?

Information sent from a form with the GET method is **visible to everyone** (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases. GET may be used for sending non-sensitive data.

**Note:** GET should NEVER be used for sending passwords or other sensitive information!

### When to use POST?

Information sent from a form with the POST method is invisible to others (all names/values are embedded within the body of the HTTP request) and has no limits on the amount of information to send. Moreover, POST supports advanced functionality such as support for multi-part binary input while uploading files to server. However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

**Note:** Developers prefer POST for sending form data

## Form Validation

Proper validation of form data is important to protect your form from hackers and spammers! The HTML form contains various input fields: required and optional text fields, radio buttons, and a submit button:

### PHP Form Validation Example

\* required field

Name:  \*

E-mail:  \*

Website:

Comment:

Gender: ☐ Female ☐ Male ☐ Other \*

First write the plain HTML code for the form:

### Text Fields

The name, email, and website fields are text input elements, and the comment field is a textarea. The HTML code looks like this:

Name: `<input type="text" name="name">`

E-mail: `<input type="text" name="email">`

Website: `<input type="text" name="website">`

Comment: `<textarea name="comment" rows="5" cols="40"></textarea>`

### Radio Buttons

The gender fields are radio buttons and the HTML code looks like this:



Gender:

```
<input type="radio" name="gender" value="female">Female
```

```
<input type="radio" name="gender" value="male">Male
```

```
<input type="radio" name="gender" value="other">Other
```

## The Form Element

The HTML code of the form looks like this:

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

When the form is submitted, the form data is sent with method="post".

## `$_SERVER["PHP_SELF"]` variable

The `$_SERVER["PHP_SELF"]` is a super global variable that returns the filename of the currently executing script.

So, the `$_SERVER["PHP_SELF"]` sends the submitted form data to the page itself, instead of jumping to a different page. This way, the user will get error messages on the same page as the form.

## `htmlspecialchars()` function

The `htmlspecialchars()` function converts special characters to HTML entities. This means that it will replace HTML characters like `<` and `>` with `&lt;` and `&gt;`. This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.

The validation rules for the form above are as follows:

Field	Validation Rules
Name	- Required. + Must only contain letters and whitespace
E-mail	- Required. + Must contain a valid email address (with @ and .)
Website	- Optional. If present, it must contain a valid URL
Comment	- Optional. Multi-line input field (textarea)
Gender	- Required. Must select one

In the following code we have added some new variables: \$nameErr, \$emailErr, \$genderErr, and \$websiteErr. These error variables will hold error messages for the required fields. We have also added an if else statement for each \$\_POST variable. This checks if the \$\_POST variable is empty (with the PHP empty() function). If it is empty, an error message is stored in the different error variables, and if it is not empty, it sends the user input data through the test\_input() function.

```
<?php

// define variables and set to empty values

$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
 if (empty($_POST["name"])) {
 $nameErr = "Name is required";
 } else {
 $name = test_input($_POST["name"]);
 }

 if (empty($_POST["email"])) {
 $emailErr = "Email is required";
 } else {
 $email = test_input($_POST["email"]);
 }

 if (empty($_POST["website"])) {
 $website = "";
 } else {
 $website = test_input($_POST["website"]);
 }

 if (empty($_POST["comment"])) {
 $comment = "";
 }
```

```

 } else {
 $comment = test_input($_POST["comment"]);
 }

 if (empty($_POST["gender"])) {
 $genderErr = "Gender is required";
 } else {
 $gender = test_input($_POST["gender"]);
 }
}
?>

```

## PHP - Display the Error Messages

Then in the HTML form, we add a little script after each required field, which generates the correct error message if needed (that is if the user tries to submit the form without filling out the required fields):

### Example:

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

```
Name: <input type="text" name="name">
```

```
* <?php echo $nameErr;?>
```

```



```

E-mail:

```
<input type="text" name="email">
```

```
* <?php echo $emailErr;?>
```

```



```

Website:

```
<input type="text" name="website">
```

```
<?php echo $websiteErr;?>
```

```



```

```
Comment: <textarea name="comment" rows="5" cols="40"></textarea>
```

<br><br>

Gender:

<input type="radio" name="gender" value="female">Female

<input type="radio" name="gender" value="male">Male

<input type="radio" name="gender" value="other">Other

<span class="error">\* <?php echo \$genderErr;?></span>

<br><br>

<input type="submit" name="submit" value="Submit">

</form>

The next step is to validate the input data, that is "Does the Name field contain only letters and whitespace?", and "Does the E-mail field contain a valid e-mail address syntax?", and if filled out, "Does the Website field contain a valid URL?".

## PHP Forms - Validate E-mail and URL

### PHP - Validate Name:

The code below shows a simple way to check if the name field only contains letters, dashes, apostrophes and whitespaces. If the value of the name field is not valid, then store an error message:

```
$name = test_input($_POST["name"]);
if (!preg_match("/^[a-zA-Z-']*$/",$name)) {
 $nameErr = "Only letters and white space allowed";
}
```

Here the preg\_match() function searches for pattern, returns true if the pattern exists, and false otherwise.

### PHP - Validate E-mail:

The easiest and safest way to check whether an email address is well-formed is to use PHP's filter\_var() function. In the code below, if the e-mail address is not well-formed, then store an error message:

```
$email = test_input($_POST["email"]);
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
 $emailErr = "Invalid email format";
}
```

### PHP - Validate URL:

The code below shows a way to check if a URL address syntax is valid (this regular expression also allows dashes in the URL). If the URL address syntax is not valid, then store an error message:

```
$website = test_input($_POST["website"]);
if (!preg_match("/^b(?:(:?https?ftp):\\|www\\.)[-a-z0-9+&@#\\/%?~_!:,;]*[-a-z0-9+&@#\\/%~_!:/i",$website)) {
 $websiteErr = "Invalid URL";
}
```

Now, the script looks like this:

**Example:**

```
<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
 if (empty($_POST["name"])) {
 $nameErr = "Name is required";
 } else {
 $name = test_input($_POST["name"]);
 // check if name only contains letters and whitespace
 if (!preg_match("/^[a-zA-Z-']*$/", $name)) {
 $nameErr = "Only letters and white space allowed";
 }
 }
}

if (empty($_POST["email"])) {
 $emailErr = "Email is required";
} else {
 $email = test_input($_POST["email"]);
 // check if e-mail address is well-formed
 if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
 $emailErr = "Invalid email format";
 }
}

if (empty($_POST["website"])) {
 $website = "";
} else {
 $website = test_input($_POST["website"]);
 // check if URL address syntax is valid (this regular expression also allows dashes in the URL)
 if (!preg_match("/^b(?:(:?https?|ftp):\\|www\\.)[-a-z0-9+&@#/%?=_~!|.,:]*[-a-z0-9+&@#/%?=_~!|/i", $website)) {

```

```

 $websiteErr = "Invalid URL";
 }
}

if (empty($_POST["comment"])) {
 $comment = "";
} else {
 $comment = test_input($_POST["comment"]);
}

if (empty($_POST["gender"])) {
 $genderErr = "Gender is required";
} else {
 $gender = test_input($_POST["gender"]);
}
}
?>

```

### Output:

## PHP Form Validation Example

*\* required field*

Name:  \*

E-mail:  \*

Website:

Comment:

Gender: ☒ Female ☐ Male ☐ Other \*

## Your Input:

### Result after validating form:

After hitting the Submit button, the error message will be generated in the form output like this,

## PHP Form Validation Example

**\* required field**

Name:  \*

E-mail:  \* Invalid email format

Website:  Invalid URL

Comment:

Gender: ☐ Female ☐ Male ☐ Other \*

### Your Input:

Gom thi  
gomathi.it sathyabama.ac.in  
www sathyabama.

female

## PHP - Keep The Values in The Form

To show the values in the input fields after the user hits the submit button, we add a little PHP script inside the value attribute of the following input fields: name, email, and website. In the comment textarea field, we put the script between the <textarea> and </textarea> tags. The little script outputs the value of the \$name, \$email, \$website, and \$comment variables.

Then, we also need to show which radio button that was checked. For this, we must manipulate the checked attribute (not the value attribute for radio buttons):

```
Name: <input type="text" name="name" value="<?php echo $name;?>">
E-mail: <input type="text" name="email" value="<?php echo $email;?>">
Website: <input type="text" name="website" value="<?php echo $website;?>">
Comment: <textarea name="comment" rows="5" cols="40"><?php echo $comment;?></textarea>
Gender:
<input type="radio" name="gender"
<?php if (isset($gender) && $gender=="female") echo "checked";?>
value="female">Female
<input type="radio" name="gender"
<?php if (isset($gender) && $gender=="male") echo "checked";?>
value="male">Male
```

```
<input type="radio" name="gender"
<?php if (isset($gender) && $gender=="other") echo "checked";?>
value="other">Other
```

## PHP Form Validation Example

\* required field

Name:  \*

E-mail:  \* Invalid email format

Website:  Invalid URL

Comment:

Gender: ☒ Female ☐ Male ☐ Other \*

### Your Input:

gom thi  
gomathi sathyabama.ac.in  
www sathyabama.ac.in  
female

## PHP File Handling

File handling is an important part of any web application. You often need to open and process a file for different tasks.

## PHP Manipulating Files

PHP has several functions for creating, reading, uploading, and editing files.

Note: When you are manipulating files you must be very careful. You can do a lot of damage if you do something wrong. Common errors are: editing the wrong file, filling a hard-drive with garbage data, and deleting the content of a file by accident.

## PHP readfile() Function

The readfile() function reads a file and writes it to the output buffer. The readfile() function is useful if all you want to do is open up a file and read its contents.

Assume we have a text file called "**webdictionary.txt**", stored on the server, that looks like this:

AJAX = Asynchronous JavaScript and XML



CSS = Cascading Style Sheets

HTML = Hyper Text Markup Language

PHP = PHP Hypertext Preprocessor

SQL = Structured Query Language

SVG = Scalable Vector Graphics

XML = EXtensible Markup Language

The PHP code to read the file and write it to the output buffer is as follows (the `readfile()` function returns the number of bytes read on success):

### Example

```
<?php
echo readfile("webdictionary.txt");
?>
```

### Result:

AJAX = Asynchronous JavaScript and XML CSS = Cascading Style Sheets HTML = Hyper Text Markup Language PHP = PHP Hypertext Preprocessor SQL = Structured Query Language SVG = Scalable Vector Graphics XML = EXtensible Markup Language236

**PHP File Open/Read/Close** - To open, read, and close a file on the server

### PHP Open File - `fopen()`

A better method to open files is with the `fopen()` function. This function gives you more options than the `readfile()` function. The first parameter of `fopen()` contains the name of the file to be opened and the second parameter specifies in which mode the file should be opened. The following example also generates a message if the `fopen()` function is unable to open the specified file:

### PHP Read File - `fread()`

The `fread()` function reads from an open file. The first parameter of `fread()` contains the name of the file to read from and the second parameter specifies the maximum number of bytes to read.

The following PHP code reads the "webdictionary.txt" file to the end:

```
fread($myfile,filesize("webdictionary.txt"));
```

### PHP Close File - `fclose()`

The `fclose()` function is used to close an open file. It's a good programming practice to close all files after finished with them. You don't want an open file running around on your server taking

up resources! The `fclose()` requires the name of the file (or a variable that holds the filename) we want to close:

**Example:**

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
echo fread($myfile, filesize("webdictionary.txt"));
fclose($myfile);
?>
```

**Result:**

AJAX = Asynchronous JavaScript and XML CSS = Cascading Style Sheets HTML = Hyper Text Markup Language PHP = PHP Hypertext Preprocessor SQL = Structured Query Language SVG = Scalable Vector Graphics XML = EXtensible Markup Language

Modes	Description
r	Open a file for read only. File pointer starts at the beginning of the file
w	Open a file for write only. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a	Open a file for write only. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x	Creates a new file for write only. Returns FALSE and an error if file already exists
r+	Open a file for read/write. File pointer starts at the beginning of the file
w+	Open a file for read/write. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a+	Open a file for read/write. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x+	Creates a new file for read/write. Returns FALSE and an error if file already exists

**PHP Read Single Line - `fgets()`**

The `fgets()` function is used to read a single line from a file. The example below outputs the first line of the "webdictionary.txt" file:

**Example:**

```
<?php

$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");

echo fgets($myfile);
```

```
fclose($myfile);
```

```
?>
```

**Result:**

AJAX = Asynchronous JavaScript and XML

**Note:** After a call to the `fgets()` function, the file pointer has moved to the next line.

**PHP Check End-Of-File - `feof()`**

The `feof()` function checks if the "end-of-file" (EOF) has been reached. The `feof()` function is useful for looping through data of unknown length. The example below reads the "webdictionary.txt" file line by line, until end-of-file is reached:

**Example:**

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one line until end-of-file
while(!feof($myfile)) {
 echo fgets($myfile) . "
";
}
fclose($myfile);
?>
```

**Cookies and Sessions****Need for Cookies and Sessions**

- HTTP is a stateless protocol
- This means that each request is handled independently of all the other requests and it means that a server or a script cannot remember if a user has been there before
- However, knowing if a user has been there before is often required and therefore something known as cookies and sessions have been implemented

**Similarity between Cookies and Sessions**

- Different mechanisms of the same solution
- **Cookies**

A mechanism for storing data in the remote browser and thus tracking or identifying return users

- **Sessions**

It is support in PHP consists of a way to preserve certain data across subsequent accesses. This enables you to build more customized applications and increase the appeal of your web site.

## What is a Cookie?

- A cookie is a piece of text that a Web server can store on a users hard disk
- A cookie is a variable, sent by the server to the browser
- Cookies allow a Web site to store information on a users machine and later retrieve it.
- The pieces of information are stored as name-value pairs
- With PHP, one can both create and retrieve cookie values

## When are Cookies Created?

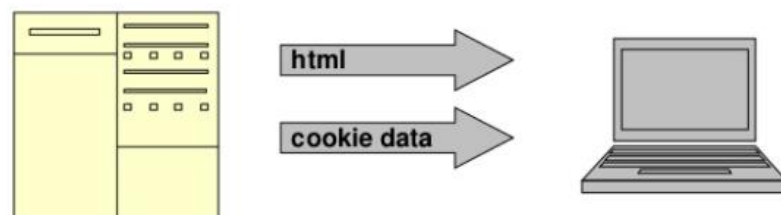
- When a new webpage is loaded
- If the user has elected to disable cookies then the write operation will fail, and subsequent sites which rely on the cookie will either have to take a default action Fig.3.1, Fig.3.2, Fig.3.3.

**Example (1) User sends a request for page at [www.example.com](http://www.example.com) for the first time. page request**



**Fig. 3.1 User request**

**Example (2) Server sends back the page html to the browser AND stores some data in a cookie on the user's PC. html cookie data**



**Fig. 3.2 Server action for request**

**Example (3) At the next page request for domain [www.example.com](http://www.example.com), all cookie data associated with this domain is sent too. page request cookie data**



**Fig. 3.3 Response for request**

## Parts of a Cookie

Each cookie is effectively a small lookup table containing pairs of (key, data) values - for example (firstname, John) (lastname, Peter)

Once the cookie has been read by the code on the server or client computer, the data can be retrieved and used to customise the web page appropriately

## Syntax for setting up a cookie

```
setcookie(name [,value [,expire [,path [,domain [,secure]]]])
```

- name = cookie
- namevalue = data to store (string)
- expire = UNIX timestamp when the cookie expires. Default is that cookie expires when browser is closed.
- path = Path on the server within and below which the cookie is available on.
- domain = Domain at which the cookie is available for.
- secure = If cookie should be sent over HTTPS connection only. Default false.

## Examples for setcookie

```
setcookie('name','Robert')
```

This command will set the cookie called name on the user's PC containing the data Robert. It will be available to all pages in the same directory or subdirectory of the page that set it (the default path and domain). It will expire and be deleted when the browser is closed (default expire)

## Examples for setcookie

```
setcookie('age','20',time()+60*60*24*30)
```

This command will set the cookie called age on the user's PC containing the data 20. It will be available to all pages in the same directory or subdirectory of the page that set it (the default path and domain). It will expire and be deleted after 30 days.

## Read cookie data

- All cookie data is available through the superglobal

```
$_COOKIE:
```

```
$variable = $_COOKIE['cookie_name'] or
```

```
$variable = $HTTP_COOKIE_VARS['cookie_name'];
```

```
Example: $age = $_COOKIE['age']
```

## Delete a cookie

- To remove a cookie, simply overwrite the cookie with a new one with an expiry time in the past

```
setcookie('cookie_name','',time()-6000)
```

Note that theoretically any number taken away from the time() function should do, but due to variations in local computer times, it is advisable to use a day or two.

### **Problems with Cookies**

- Browsers can refuse to accept cookies
- Additionally, it adds network overhead to send lots of information back and forth.
- There are also limits to the amount of information that can be sent
- Some information you just don't want to save on the client's computer.

### **Sessions**

- A Session allows to store user information on the server for later use (i.e. username, shopping cart items, etc)
- However, this session information is temporary and is usually deleted very quickly after the user has left the website that uses sessions
- Session variables hold information about one single user, and are available to all pages in one application.

### **Starting a PHP Session**

- Before you can store user information in your PHP session, you must first start up the session
- The session\_start() function must appear BEFORE the <html> tag.

```
<?php session_start(); ?>
```

```
<html>
```

```
<body>
```

```
..
```

```
..
```

```
</body>
```

```
</html>
```

### **Storing a Session Variable**

- The \$\_SESSION variable can be store and retrieved in PHP

```
<?php
```

```
session_start();
```

```
// store session data
$_SESSION[views]=1;
?>

<html>

<body>

</body>

</html>
```

### Retrieving a Session Variable

```
<html>

<body>

<?php

//retrieve session data

echo "Pageviews=". $_SESSION[views];

?>

</body>

</html>
```

Display:

Pageviews = 1

### Destroying a Session

- The unset() function is used to free the specified session variable.

```
<?php

unset($_SESSION[views]);

?>
```

- You can also completely destroy the session by calling the session\_destroy() function:

```
<?php

session_destroy();

?>
```

- session\_destroy() will reset your session and you will lose all your stored session data.

### Cookies vs. Sessions

## Cookies

- Cookies are stored on client side
- Cookies can only store strings
- Cookies can be set to a long lifespan
- 

## Sessions

Sessions are stored on server

Sessions can store objects

When users close their browser, lifespan. they also lost the session

## Filters- Introduction

- PHP Filter is an extension that filters the data by either sanitizing or validating it
- It plays a crucial role in security of a website, especially useful when the data originates from unknown or foreign sources, like user supplied input
- For example data from a TML form
- There are mainly two types of filters

### 1. Validation

### 2. Sanitization

## Types of filters

- **Validation**

Used to validate or check if the data meets certain qualifications or not

Example: passing in `FILTER_VALIDATE_URL` will determine if the data is a valid url, but it will not change the existing data by itself

- **Sanitization**

Unlike validation, sanitization will sanitize data so as to ensure that no undesired characters by removing or altering the data

Example passing in `FILTER_SANITIZE_EMAIL` will remove all the characters that are inappropriate for an email address to contain. That said, it does not validate the data

## Example PHP program to validate

Program to validate URL using `FILTER_VALIDATE_URL` filter

```
<?php
```

```
// PHP program to validate URL
```

```
// Declare variable and initialize it to URL
```

```
$url = "https://www.geeksforgeeks.org";
```



```
// Use filter function to validate URL
```

```
if (filter_var($url, FILTER_VALIDATE_URL)) {
```

```
 echo("valid URL");
```

```
}
```

```
else {
```

```
 echo("Invalid URL");
```

```
}
```

Program to validate email using FILTER\_VALIDATE\_EMAIL filter

```
<?php
```

```
// PHP program to validate email
```

```
// Declare variable and initialize it to email
```

```
$email = "xyz@gmail.com";
```

```
// Use filter function to validate email
```

```
if (filter_var($email, FILTER_VALIDATE_EMAIL)) { echo "Valid Email";
```

```
}
```

```
else {
```

```
 echo "Invalid Email";
```

```
}
```

```
?>
```

### **Filter Functions**

- The filter function is used to filter the data coming from insecure source

**filter\_var():** Filters a specific variable

**filter\_var\_array():**Filters multiple variable i.e. array of variable

**filter\_has\_var():** Check if the variable of specific input type exists or not

**filter\_id():**helps to get filter id of the specified filter name

**filter\_list():**Returns a list of supported filter name in the form of array

**filter\_input():**Gets an external variable and filters it if set to do so

**filter\_input\_array():**same as filter\_input() but here Gets multiple variables i.e. array of variable and filters them if set to do so

### **Predefined Filter Constants**

- **Validate filter constants**

**FILTER\_VALIDATE\_BOOLEAN:** Validates a boolean

**FILTER\_VALIDATE\_INT:** Validates an integer

**FILTER\_VALIDATE\_FLOAT:** Validates a float

**FILTER\_VALIDATE\_REGEXP:** Validates a regular expression

**FILTER\_VALIDATE\_IP:** Validates an IP address

**FILTER\_VALIDATE\_EMAIL:** Validates an e-mail address

**FILTER\_VALIDATE\_URL:** Validates an URL

#### **Predefined Filter Constants**

- **Sanitize filter constants**
- **FILTER\_SANITIZE\_EMAIL:** Removes all illegal characters from an e-mail address
- **FILTER\_SANITIZE\_ENCODED:** Removes/Encodes special characters
- **FILTER\_SANITIZE\_MAGIC\_QUOTES:** Apply addslashes() function
- **FILTER\_SANITIZE\_NUMBER\_FLOAT:** Remove all characters, except digits, +- and optionally ., eE
- **FILTER\_SANITIZE\_NUMBER\_INT:** Removes all characters except digits and + –
- **FILTER\_SANITIZE\_SPECIAL\_CHARS:** Removes special characters
- **FILTER\_SANITIZE\_FULL\_SPECIAL\_CHARS** Encoding quotes can be disabled by using FILTER\_FLAG\_NO\_ENCODE\_QUOTES.
- **FILTER\_SANITIZE\_STRING :** Removes tags/special characters from a string
- **FILTER\_SANITIZE\_STRIPPED :** Alias of FILTER\_SANITIZE\_STRING
- **FILTER\_SANITIZE\_URL:** Removes all illegal character from s URL

#### **Predefined Filter Constants**

- **Other filter constants**
- **FILTER\_UNSAFE\_RAW :**Do nothing, optionally strip/encode special characters
- **FILTER\_CALLBACK :**Call a user-defined function to filter data

#### **Exception Handling in PHP**

- An exception is unexpected program result that can be handled by the program itself. Exception Handling in PHP is almost similar to exception handling in all programming languages

PHP provides following specialized keywords for this purpose

- **try:** It represent block of code in which exception can arise
- **catch:** It represent block of code that will be executed when a particular exception has been thrown
- **throw:** It is used to throw an exception. It is also used to list the exceptions that a function throws, but doesn't handle itself
- **finally:** It is used in place of catch block or after catch block basically it is put for cleanup activity in PHP code

### Need for Exception Handling in PHP

Following are the main advantages of exception handling over error handling

- **Separation of error handling code from normal code:** In traditional error handling code there is always if else block to handle errors. These conditions and code to handle errors got mixed so that becomes unreadable. With try Catch block code becomes readable
- **Grouping of error types:** In PHP both basic types and objects can be thrown as exception. It can create a hierarchy of exception objects, group exceptions in namespaces or classes, categorize them according to types

### Exception handling in PHP

```
<?php
```

```
// PHP Program to illustrate normal
```

```
// try catch block code
```

```
function demo($var) {
```

```
 echo " Before try block";
```

```
 try {
```

```
 echo "\n Inside try block";
```

```
 // If var is zero then only if will be executed
```

```
 if($var == 0)
```

```
 {
```

```
 // If var is zero then only exception is thrown
```

```
 throw new Exception('Number is zero.');
```

```
 }

 // This line will never be executed
```

```

 echo "\n After throw (It will never be executed)";
 }
}

// Catch block will be executed only
// When Exception has been thrown by try block
catch(Exception $e) {
 echo "\n Exception Caught", $e->getMessage();
}

// This line will be executed whether
// Exception has been thrown or not
echo "\n After catch (will be always executed)";
}

// Exception will not be rised
demo(5)
// Exception will be rised here
demo(0);
?>

```

### Output

Before try block Inside try block  
 After catch (will be always executed) Before try block Inside try block  
 Exception CaughtNumber is zero. After catch (will be always executed)

### Using Custom Exception Class

```

<?php
class myException extends Exception {
 function get_Message() {

 // Error message
 $errorMsg = 'Error on line '.$this->getLine().
 ' in '.$this->getFile()
 }
}

```

```

 .$this->getMessage().' is number zero';
 return $errorMsg;
 }
}

```

```

function demo($a) {
 try {

 // Check if
 if($a == 0) {
 throw new myException($a);
 }
 }
}

```

```

catch (myException $e) {

 // Display custom message
 echo $e->get_Message();
}

// This will not generate any exception
demo(5);

```

```

// It will cause an exception
demo(0);

?>

```

### Output

```

Error on line 20 in
/home/45ae8dc582d50df2790517e9
12980806.php0 is number zero

```

### Set a Top Level Exception Handler

- The **set\_exception\_handler()** function set all user defined function to all uncaught exception
- Example

```

<?php
function exception_handler($exception) {
 echo "Uncaught exception: " . $exception->getMessage(), "\n";
}

set_exception_handler('exception_handler');

throw new Exception('Uncaught Exception');
echo "Not Executed\n";
?>

```

## Database Connectivity With MySQL

### BASIC DATABASE SERVER CONCEPTS

world's most popular open source database because of its consistent **fast performance, high reliability and ease of use**

### BASIC DATABASE SERVER CONCEPTS

Database runs as a server

Attaches to either a default port or an administrator specified port

Clients connect to database

For secure systems

authenticated connections

usernames and passwords

Clients make queries on the database

Retrieve content

Insert content

SQL (Structured Query Language) is the language used to insert and retrieve content

## MY SQL

- MySQL can be controlled through a simple command-line interface; however, we can use phpMyAdmin as an interface to MySQL
- phpMyAdmin is a very powerful tool; it provides a large number of facilities for customising a database management system

## CONNECTING to MY SQL

- In order for our PHP script to access a database we need to form a connection from the script to the database management system

```
resourceId = mysql_connect(server, username, password);
```

Server is the DBMS server

username is your username  
password is your password

## CONNECTING to MY SQL DBMS

- In order for our PHP script to access a database we need to form a connection from the script to the database management system

***resourceId = mysql\_connect(server, username, password);***

The function returns a resource-identifier type

- a PHP script can connect to a **DBMS anywhere in the world**, so long as it is connected to the internet

- We can also connect to multiple DBMS at the same time

## SELECTING A DATABASE

- **Once connected to a DBMS, we can select a database**

***mysql\_select\_db(databasename, resourceId);***

## Create HTML form for connecting to database

- you need to create a working folder first and then create a web page with the name “contact.html”
- If you install xampp your working folder is in folder this “E:\xampp\htdocs”
- You can create a new folder “contact” on your localhost working folder
- Create a “contact.html” file and paste the following code

## Code create HTML form for connecting to database

Create a PHP page to save data from HTML form to your MySQL database

- The contact HTML form action is on “contact.php” page. On this page, we will write code for inserting records into the database.
- For storing data in MySQL as records, you have to first connect with the DB. Connecting the code is very simple. The mysql\_connect in PHP is

***mysqli\_connect.***

Example:

```
$con = mysqli_connect("localhost","your_localhost_database_user",
"your_localhost_database_password","your_localhost_database_db");
```

## Local Host

You need to place value for your localhost username and password. Normally localhost MySQL database username is root and password blank or root

Example

```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Contact Form - PHP/MySQL Demo Code</title>

</head>

<body>

<fieldset>

<legend>Contact Form</legend>

<form name="frmContact" method="post" action="contact.php">

<p>

<label for="Name">Name </label>

<input type="text" name="txtName" id="txtName">

</p>

<p>

<label for="email">Email</label>

<input type="text" name="txtEmail" id="txtEmail">

</p>

<p>

<label for="phone">Phone</label>

<input type="text" name="txtPhone" id="txtPhone">

</p>

<p>

<label for="message">Message</label>

<textarea name="txtMessage" id="txtMessage"></textarea>

</p>

<p> </p>
```



```

<p>
<input type="submit" name="Submit" id="Submit" value="Submit">
</p>
</form>
</fieldset>
</body> </html>

```

### QUESTION BANK

S.No	Questions (2 Marks)	Competence	BT Level
1.	Contrast between echo and print in PHP?	Analyze	BTL4
2.	Interpret the correct and the most two common way to start and finish a PHP block of code?	Understand	BTL2
3.	Compare a session and cookies?	Analyze	BTL4
4.	Categorize the php data types.	Analyze	BTL4
5.	Illustrate how can PHP and Javascript interact?	Analyze	BTL4
6.	Construct the code to connect to a MySQL database from a PHP script?	Create	BTL6
7.	How can we access the data sent through the URL with the GET method?	Knowledge	BTL1
8.	Discover the statement to return a value from a function?	Analyze	BTL4
9.	How is a constant defined in a PHP script?	Knowledge	BTL1
10.	Criticize integer 12 and a string "13" in PHP?	Apply	BTL5
11.	Define Persistent Cookie?	Knowledge	BTL1
12.	List out the PHP String Functions.	Analyze	BTL4
13.	Solve and output the following code: <pre> &lt;?php declare(strict_types=1); //strict requirement function addNumbers(float \$a, float \$b) : float {     return \$a + \$b; }  echo addNumbers(1.2, 5.2); </pre>	Create	BTL6

	?>		
14.	What is htmlspecialchars() function?	Knowledge	BTL1
15.	Identify the use of preg_match() function	Apply	BTL3
16.	Identify the command for destroy a session	Apply	BTL3
17.	Explain the necessity of Cookies and sessions	Knowledge	BTL1
18.	Identify the situation for cookies creation	Apply	BTL3
19.	List out the parts of cookies	Knowledge	BTL1
20.	Identify the problems that a cookie causes	Apply	BTL3
21.	Justify session information is temporary	Create	BTL6
22.	Explain the need of filters in PHP	Understand	BTL2
23.	Differentiate cookies and Sessions	Understand	BTL2
24.	List the type of filters in PHP	Knowledge	BTL1
25.	Determine the usage of filter_list(): in PHP	Create	BTL6
26.	Classify and define the two advantages exception handling	Apply	BTL3
27.	Illustrate custom exception class	Apply	BTL3
<b>S.No.</b>	<b>Questions (16 Marks)</b>	<b>Competent</b>	<b>BT Level</b>
1.	Design a simple HTML form with two input fields and a submit button and send the form data for processing by php file. To get form data, use PHP superglobals \$_GET and \$_POST.	Create	BTL6
2.	Construct a HTML form with various input fields: required and optional text fields, radio buttons, and a submit button to get name, email id and gender details and validate the above data using PHP.	Create	BTL6
3.	Write a php function to sort an array.	Create	BTL6
4.	Develop HTML form for connecting to database	Evaluate	BTL5

5.	With neat sketch explain the database connectivity with my SQL	Apply	BTL3
6.	Describe top level exception handler with example program	Create	BTL6
7.	Develop a code for create a cookie and destroy the same	Create	BTL6
8.	Explain the comments in connecting MYSQL	Create	BTL6



**SATHYABAMA**

INSTITUTE OF SCIENCE AND TECHNOLOGY  
(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

[www.sathyabama.ac.in](http://www.sathyabama.ac.in)

## UNIT – IV -Customer Interface Design and Development– SITA1502

## UNIT 4 ANGULAR JS AND JQUERY

Angular JS Expression – Modules – Directives – Data Binding – Controllers – Scopes – Filters – Services – Tables – Events – Form – Validation. jQuery Syntax – Selects – Events – jQuery Effects – jQuery – jQuery HTML – jQuery Traversing.

### What is AngularJS

MVC Javascript Framework by Google for Rich Web Application Development

### Why AngularJS

Other frameworks deal with HTML's shortcomings by either abstracting away HTML, CSS, and/or JavaScript or by providing an imperative way for manipulating the DOM.

- Structure, Quality and Organization
- Lightweight ( < 36KB compressed and minified)
- Free Separation of concern
- Modularity
- Extensibility & Maintainability
- Reusable Components
- jQuery Allows for DOM Manipulation
- Does not provide structure to your code
- Does not allow for two-way binding

### Features of AngularJS

- Two-way Data Binding – Model as single source of truth
- Directives – Extend HTML
- MVC
- Dependency Injection
- Testing
- Deep Linking (Map URL to route Definition)
- Server-Side Communication
- Data Binding

```
<html ng-app>
<head>
 <script src='angular.js'></script>
</head>
<body>
 <input ng-model='user.name'>
 <div ng-show='user.name'>Hi {{user.name}}</div>
</body>
</html>
```

### Example

```
<html ng-app>
<head>
 <script src='angular.js'></script>
 <script src='controllers.js'></script>
</head>
<body ng-controller='UserController'>
 <div>Hi {{ user.name }}</div>
</body>
</html>
function XXXX($scope) {
 $scope.user = { name:'Larry' };
}
```

### Hello Javascript

```
<p id="greeting1"></p>
```

```
<script>
var isIE = document.attachEvent;
var addListener = isIE
 ? function(e, t, fn) {
 e.attachEvent('on' + t, fn);}
 : function(e, t, fn) {
 e.addEventListener(t, fn, false);};
addListener(document, 'load', function(){
 var greeting = document.getElementById('greeting1');
 if (isIE) {
 greeting.innerText = 'Hello World!';
 } else {
 greeting.textContent = 'Hello World!';
 }
});
</script>
```

### Hello JQuery

```
<p id="greeting2"></p>
```

```
<script>
$(function(){
 $('#greeting2').text('Hello World!');
});
</script>
```

### Sample Angular Powered View

```
<body ng-app="F1FeederApp" ng-controller="driversController">
 <table>
 <thead>
 <tr><th colspan="4">Drivers Championship Standings</th></tr>
 </thead>
 <tbody>
```

```

<tr ng-repeat="driver in driversList">
 <td>{{ $index + 1 }}</td>

 <td>

 {{ driver.Driver.givenName }} {{ driver.Driver.familyName }}

 </td>

 <td>{{ driver.Constructors[0].name }}</td>

 <td>{{ driver.points }}</td>

</tr>

</tbody>

</table>

</body>

```

## Expressions

Expressions allow you to execute some computation in order to return a desired value.

- {{ 1 + 1 }}
- {{ 946757880 | date }}
- {{ user.name }}

you shouldn't use expressions to implement any higher-level logic.

## Directives

Directives are markers (such as attributes, tags, and class names) that tell AngularJS to attach a given behaviour to a DOM element (or transform it, replace it, etc.)

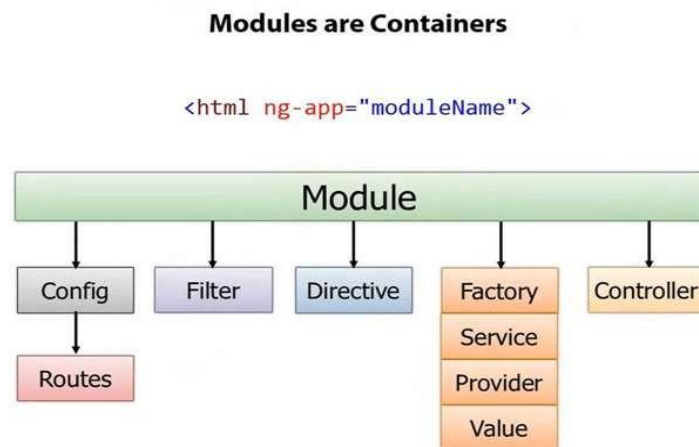
### Some angular directives

The ng-app - Bootstrapping your app and defining its scope.

- The ng-controller - defines which controller will be in charge of your view.
- The ng-repeat - Allows for looping through collections

## What is module ?

- The module is container for different parts of an application.  
I.e controllers,services,directives,filters.
- Controllers always belong to a module.
- Module is used as a main() method.angular is an object and module is a method. Module is which present in angular object Fig.4.1.



**Fig. 4.1. Modules**

## Why module

- Module is starting point of angularjs application.
- ng-app that have empty module make its controller function global.
- Controller function will become global if we don't register it with the module then there may be chances of overriding data. To avoid this we have register controller function with the module.
- So that can solve global value problem.

## How to create a module

Syntax:

```
Var app = angular.module("myModule", []);
```

## How to add controller to Module

```
var myApp=angular.module("myModule",[])
```



```
myApp.controller("myController",function($scope){ //Register the controller with the
module.
});
```

## kinds of directives in Angular

- ❑ There are **three** kinds of directives in Angular:
  - ❖ **Components**— directives with a template.
  - ❖ **Structural directives**— change the DOM layout by adding and removing DOM elements.
  - ❖ **Attribute directives**— change the appearance or behavior of an element, component, or another directive.

## Directives

- ❑ Structural Directives change the structure of the view. Two examples are **NgFor** and **NgIf**.
- ❑ Structural **directives are easy to recognize. An asterisk (\*) precedes the directive attribute name as in this.**
  - Attribute directives are used as **attributes of elements**. The built-in **NgStyle** directive in the Template
- ❑ Three of the common, built-in structural directives—**NgIf, NgFor, and NgSwitch**.

## NgIf

```
<div *ngIf="hero" class="name">{{hero.name}}</div>
```

- ❑ No brackets. No parentheses. Just **\*ngIf** set to a string.
- ❑ **The asterisk (\*) is a convenience** notation and the string is a micro syntax rather than the usual template

expression.

### Render to

```
<ng-template [ngIf]="hero">
<div class="name">{{ hero.name }}</div>
</ng-template>
```

- ❑ Angular desugars this notation into a marked-up **<ng-template>** that surrounds the host element and its

descendents. Each structural directive does something different with that template.

## NgSwitch

❑ The Angular **NgSwitch** is actually a

**set of cooperating directives:**

- a. NgSwitch,
- b. NgSwitchCase
- c. NgSwitchDefault

NgSwitch Example

```
<ul [ngSwitch]="person.country">
<li *ngSwitchCase="'UK'" class="text-success">
 {{ person.name }} ({{ person.country }})

<li *ngSwitchCase="'USA'" class="text-primary">
 {{ person.name }} ({{ person.country }})

<li *ngSwitchDefault class="text-warning">
 {{ person.name }} ({{ person.country }})


```

**NgClass**

❑ **Adds and removes CSS classes on an HTML element.**

❑ The CSS classes are updated as follows, depending on the type of the expression evaluation:

- a. **string** - the CSS classes listed in the string (space delimited) are added,
- b. **Array** - the CSS classes declared as Array elements are added,
- c. **Object** - keys are CSS classes that get added when the expression given in the value evaluates to a truthy value, otherwise they are removed.

```
<some-element [ngClass]="['first second']">...</some-element>
```

```
<some-element [ngClass]="['first', 'second']">...</some-element>
```

```
<some-element [ngClass]="{'first': true, 'second': true, 'third': false}">...</some-element>
```

```
<some-element [ngClass]="string: Exp|array: Exp|obj: Exp">...</some-element>
```

```
<some-element [ngClass]="{'class1 class2 class3' : true}">...</some-element>
```

## NgStyle

- ❑ An attribute directive that updates styles for the containing HTML element.
- ❑ Sets one or more style properties, specified as colon-separated key-value pairs.
- ❑ The key is a style name, with an optional .
- ❑ <unit> suffix (such as 'top.px', 'font-style.em').
- ❑ The value is an expression to be evaluated.
- ❑ The resulting non-null value, expressed in the given unit, is assigned to the given style property.
- ❑ If the result of evaluation is null, the corresponding style is removed.

<some-element [ngStyle]="{'font-style': styleExp}">...</some-element>

<some-element [ngStyle]="{'max-width.px': widthExp}">...</some-element>

<some-element [ngStyle]="objExp">...</some-element>

## Data Binding in Angular

### Types of Data Binding in Angular

- 1. **String Interpolation:**
- The type of one-way data binding where text is between a set of curly braces often uses the name of a component property. Angular replaces that name with the string value of the corresponding component property. The syntax of string interpolation is to use double curly braces.

{{ code }}

### 2: Property Binding:

- Property Binding allows us to bind the view of the template expression. Property binding in simple term is defined as updating the value of a certain variable in component (model) and displaying it in view (presentation layer).
- This is a one-way mechanism, thus it allows you to change the value whenever you want but only at the component level.
- binding.component.html

<h1>PropertyBinding</h1><img [src]="imagePath" class="image-adjustment"/><br>

### Sample Program

- binding.component.ts

- ```
import { Component, OnInit } from '@angular/core';@Component({ selector: 'app-binding', templateUrl: './binding.component.html', styleUrls: ['./binding.component.css']})export class BindingComponent implements OnInit { constructor() { } imagePath: string = 'assets/images/Databinding.png'; ngOnInit() { } }}
```
- Though, both doing the same thing, so what is the difference between Property Binding and Interpolation?

3: Event Binding

- Event binding is defined as the updating/sending of the value/information of a certain variable from the presentation layer (view) to the component (model). For example, clicking a button.

binding.component.html

```
<h1 Event Binding></h1><h1>{{ title }}</h1><button (click)="changeMyTitle()">Title is changed on click of this button.</button>
```

Sample Program

- binding.component.ts
- ```
export class BindingComponent implements OnInit { constructor() { } title = 'Learning string interpolation'; ngOnInit() { } changeMyTitle() { this.title = 'Learning Event Binding'; }}
```
- 
- **Output:**

## Learning string interpolation

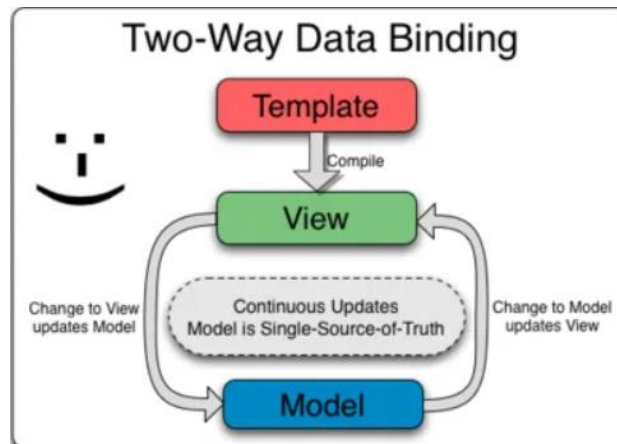
Title is changed on click of this button.

## Learning Event Binding

Title is changed on click of this button.

### 4: Two-Way Data Binding

- Two-way data binding is a combination of both Property and Event binding and it is a continuous synchronization of a data from view to the component and component to the view Fig.3.2.



**Fig.4.2 Two way data binding**

### Sample Program

- binding.component.html
- `<h1>Data Binding in Angular</h1><h2>Learning Two-Way DataBinding</h2><input type = "text" [(ngModel)]="userName"/><br><h4>Welcome: {{userName}}</h4>`
- **Output**

## Data Binding in Angular

### Learning Two-Way DataBinding

Two way  
Welcome: Two way

### Controllers

- AngularJS application mainly relies on controllers to control the flow of data in the application.
- A controller is defined using *ng-controller* directive.
- A controller is a JavaScript object that contains attributes/properties, and functions.
- Each controller accepts *\$scope* as a parameter, which refers to the application/module that the controller needs to handle.

### Example

- `<div ng-app = "" ng-controller = "studentController">`

...

`</div>`

Here, we declare a controller named *studentController*, using the *ng-controller* directive

## Scope in a Directive

- This scope object is used for accessing the variables and functions defined in the AngularJS controllers
- The controller and link functions of the directive. By default, directives do not create their own scope; instead they use the scope of their parent, generally a controller (within the scope of which the directive is defined).
- We can change the default scope of the directive using the scope field of the DDO (Data Definition Object).
- Note that scope is a property of the directive, just like restrict, template, etc. The value of the scope field defines how the scope of the directive will be created and used. These values are 'true', 'false,' and '{}'.

### scope: false

This is the default value of scope in the directive definition. In this case, the directive has the same scope as its parent controller.

## Creating an Angular app and controller

```
var movieApp = angular.module("movieApp",[]);
```

```
movieApp.controller("movieController",function($scope){
 $scope.movie = "Ice Age";
});
```

## Embedding the directive in HTML

```
<div ng-app="movieApp">
 <div ng-controller="movieController">
 <h2>Movie: {{ movie }}</h2>
 Change Movie Title : <input type='text' ng-model='movie'/>
 <movie-directive></movie-directive>
 </div>
</div>
```

## Output

Movie: Ice Age  
 Change Movie Title :   
 Movie title : Ice Age  
 Type a new movie title :

On changing the Movie Title in the text box above:

Movie: Ice Ag  
 Change Movie Title :   
 Movie title : Ice Ag  
 Type a new movie title :

### scope: true

- In this case, the movie title in both the header section and the directive is initialized to the same value defined in the controller's scope, *Ice Age*.

```
<div ng-app="movieApp">
```

```
 <div ng-controller="movieController">
```

```
 <h2>Movie: {{ movie }}</h2>
```

```
 Change Movie Title : <input type='text' ng-model='movie' />
```

```
 <movie-directive></movie-directive>
```

```
 </div>
```

```
</div>
```

### Output

Movie: Ice Age  
 Change Movie Title :   
 Movie title : Ice Ag  
 Type a new movie title :

Movie: Ice Age  
 Change Movie Title :   
 Movie title : Ice Ag  
 Type a new movie title :

### scope: true

- Let's change the scope field of the directive in the last example to true

```
<div ng-app="movieApp">
```

```
 <div ng-controller="movieController">
```

```

 <h2>Movie: {{movie}}</h2>

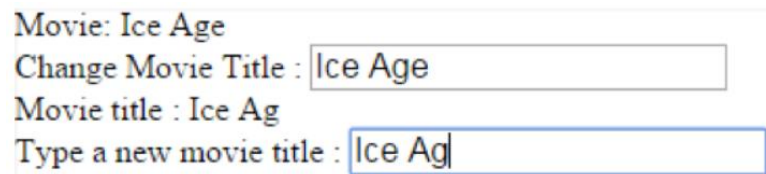
 Change Movie Title : <input type='text' ng-model='movie' />

 <movie-directive></movie-directive>

</div>
</div>

```

## Output



Movie: Ice Age  
 Change Movie Title :   
 Movie title : Ice Ag  
 Type a new movie title :

## scope: {}

This is how the directive gets its isolated scope. We pass an object for the scope field in this case. This way, the scope of the directive is not inherited from the parent and is instead completely detached from it. Thus, the directive has an isolated scope.

```

movieApp.directive("movieDirective", function(){
 return {
 restrict: "E",
 scope: {},
 template: "<div>Movie title : {{movie}}</div>" +
 "Type a new movie title : <input type='text' ng-model='movie' />"
 };
});

```

## Isolated scope – Prefixes

The directive scope uses prefixes to achieve that. Using prefixes helps establish a two-way or one-way binding between parent and directive scopes, and also make calls to parent scope methods. To access any data in the parent scope requires passing the data at two places – the directive scope and the directive tag.

### 1. Passing data to the directive scope

```

movieApp.directive("movieDirective", function(){
 return {
 restrict: "E",

```



```

scope: {
 movie: '=',
},
template: "<div>Movie title : {{ movie }}</div>" +
 "Type a new movie title : <input type='text' ng-model='movie' />"
};
});

```

## 2. Passing data in the directive tag

```
<movie-directive movie="movie"></movie-directive>
```

There are 3 types of prefixes in AngularJS:

1. '@' - Text binding / one-way binding
2. '=' - Direct model binding / two-way binding
3. '&' - Behavior binding / Method binding

## Filters

Filters are used to modify the data. They can be clubbed in expression or directives using pipe (|) character. The following list shows the commonly used filters Table.4.1.

Table.4.1 Filter Types

Sr.No.	Name & Description
1	<b>uppercase</b> converts a text to upper case text.
2	<b>lowercase</b> converts a text to lower case text.
3	<b>currency</b> formats text in a currency format.
4	<b>filter</b> filter the array to a subset of it based on provided criteria.
5	<b>orderBy</b> orders the array based on provided criteria.

## 1. Uppercase Filter

Add uppercase filter to an expression using pipe character. Here we've added uppercase filter to print student name in all capital letters.

Enter first name:<input type = "text" ng-model = "student.firstName">

Enter last name: <input type = "text" ng-model = "student.lastName">

Name in Upper Case: `{{ student.fullName() | uppercase }}`

## 2. Lowercase Filter

Add lowercase filter to an expression using pipe character. Here we've added lowercase filter to print student name in all lowercase letters.

### Syntax

`{{ string | lowercase }}`

## 3. Currency Filter

Add currency filter to an expression returning number using pipe character. Here we've added currency filter to print fees using currency format.

`{{ number | currency : symbol : fractionsize }}`

## 4. Filter

- To display only required subjects, we use `subjectName` as filter.

```
<div ng-app="myApp" ng-controller="namesCtrl">
```

```

 <li ng-repeat="x in names | filter : 'i'">
 {{ x }}

</div>
```

## 5. OrderBy Filter

To order subjects by marks, we use `orderBy` marks.

`{{ array | orderBy : expression : reverse }}`

## Service

- In AngularJS, a service is a function, or object, that is available for, and limited to, your AngularJS application.
- AngularJS has about 30 built-in services.
- Syntax

\$location

## Example

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
 $http.get("welcome.htm").then(function (response) {
 $scope.myWelcome = response.data;
 });
});
```

## The \$timeout Service

The \$timeout service is AngularJS' version of the window.setTimeout function.

## 9.Tables

Tables are one of the common elements used in HTML when working with web pages.

Tables in HTML are designed using the HTML tag

1. <table> tag – This is the main tag used for displaying the table.
2. <tr> - This tag is used for segregating the rows within the table.
3. <td> - This tag is used for displaying the actual table data.
4. <th> - This is used for the table header data

Using the above available HTML tags along with AngularJS, we can make it easier to populate table data. In short, the ng-repeat directive is used to fill in table data. The structure of the table is still created using the normal HTML tags of <table>, <tr>, <td> and <th>. However, the data is populated by using the ng-repeat directive in AngularJS. Let's look a simple example of how we can implement Angular tables.

The ng-repeat directive is perfect for displaying tables.

### 9.1 Displaying Data in a Table

Displaying tables with angular is very simple:

## Example

```
<div ng-app="myApp" ng-controller="customersCtrl">
```

```
<table>
 <tr ng-repeat="x in names">
 <td>{{ x.Name }}</td>
 <td>{{ x.Country }}</td>
 </tr>
</table>
```

</div>

<script>

```
var app = angular.module('myApp', []);
app.controller('customersCtrl', function($scope, $http) {
 $http.get("customers.php")
 .then(function (response) {$scope.names = response.data.records;});
});
</script>
```

## Output

|                                    |             |
|------------------------------------|-------------|
| Alfreds Futterkiste                | Germany     |
| Ana Trujillo Emparedados y helados | Mexico      |
| Antonio Moreno Taquería            | Mexico      |
| Around the Horn                    | UK          |
| B's Beverages                      | UK          |
| Berglunds snabbköp                 | Sweden      |
| Blauer See Delikatessen            | Germany     |
| Blondel père et fils               | France      |
| Bólido Comidas preparadas          | Spain       |
| Bon app'                           | France      |
| Bottom-Dollar Marketse             | Canada      |
| Cactus Comidas para llevar         | Argentina   |
| Centro comercial Moctezuma         | Mexico      |
| Chop-suey Chinese                  | Switzerland |
| Comércio Mineiro                   | Brazil      |

## 9.2 Displaying with CSS Style

To make it nice, add some CSS to the page: Example

<style>

```
table, th, td {
 border: 1px solid grey;
 border-collapse: collapse;
 padding: 5px;
}
```

```
table tr:nth-child(odd) {
 background-color: #f1f1f1;
```

```

}

table tr:nth-child(even) {
 background-color: #ffffff;
}
</style>

```

## Display with orderBy Filter

To sort the table, add an **orderBy** filter: Example

```

<table>
 <tr ng-repeat="x in names | orderBy : 'Country'">
 <td>{{ x.Name }}</td>
 <td>{{ x.Country }}</td>
 </tr>
</table>

```

## OUTPUT:

|                                    |             |
|------------------------------------|-------------|
| Cactus Comidas para llevar         | Argentina   |
| Comércio Mineiro                   | Brazil      |
| Bottom-Dollar Marketse             | Canada      |
| Blondel père et fils               | France      |
| Bon app'                           | France      |
| Alfreds Futterkiste                | Germany     |
| Blauer See Delikatessen            | Germany     |
| Ana Trujillo Emparedados y helados | Mexico      |
| Antonio Moreno Taquería            | Mexico      |
| Centro comercial Moctezuma         | Mexico      |
| Bólido Comidas preparadas          | Spain       |
| Berglunds snabbköp                 | Sweden      |
| Chop-suey Chinese                  | Switzerland |
| Around the Horn                    | UK          |
| B's Beverages                      | UK          |

### 9.3 Display with uppercase Filter

To display uppercase, add an **uppercase** filter: Example

```
<table>
 <tr ng-repeat="x in names">
 <td>{{ x.Name }}</td>
 <td>{{ x.Country | uppercase }}</td>
 </tr>
</table>
```

**OUTPUT:**

Alfreds Futterkiste	GERMANY
Ana Trujillo Emparedados y helados	MEXICO
Antonio Moreno Taquería	MEXICO
Around the Horn	UK
B's Beverages	UK
Berglunds snabbköp	SWEDEN
Blauer See Delikatessen	GERMANY
Blondel père et fils	FRANCE
Bólido Comidas preparadas	SPAIN
Bon app'	FRANCE
Bottom-Dollar Marketse	CANADA
Cactus Comidas para llevar	ARGENTINA
Centro comercial Moctezuma	MEXICO
Chop-suey Chinese	SWITZERLAND
Comércio Mineiro	BRAZIL

### 9.4 Display the Table Index (\$index)

To display the table index, add a <td> with **\$index**:

```
<table>
 <tr ng-repeat="x in names">
 <td>{{ $index + 1 }}</td>
```

```

 <td>{{ x.Name }}</td>
 <td>{{ x.Country }}</td>
</tr>
</table>

```

### Output:

1	Alfreds Futterkiste	Germany
2	Ana Trujillo Emparedados y helados	Mexico
3	Antonio Moreno Taquería	Mexico
4	Around the Horn	UK
5	B's Beverages	UK
6	Berglunds snabbköp	Sweden
7	Blauer See Delikatessen	Germany
8	Blondel père et fils	France
9	Bólido Comidas preparadas	Spain
10	Bon app'	France
11	Bottom-Dollar Marketse	Canada
12	Cactus Comidas para llevar	Argentina
13	Centro comercial Moctezuma	Mexico
14	Chop-suey Chinese	Switzerland
15	Comércio Mineiro	Brazil

### 9.5 Using \$even and \$odd

```

<table>
 <tr ng-repeat="x in names">
 <td ng-if="$odd" style="background-color:#f1f1f1">{{ x.Name }}</td>
 <td ng-if="$even">{{ x.Name }}</td>
 <td ng-if="$odd" style="background-color:#f1f1f1">{{ x.Country }}</td>
 <td ng-if="$even">{{ x.Country }}</td>
 </tr>
</table>

```

Alfreds Futterkiste	Germany
Ana Trujillo Emparedados y helados	Mexico
Antonio Moreno Taquería	Mexico
Around the Horn	UK
B's Beverages	UK
Berglunds snabbköp	Sweden
Blauer See Delikatessen	Germany
Blondel père et fils	France
Bólido Comidas preparadas	Spain
Bon app'	France
Bottom-Dollar Marketse	Canada
Cactus Comidas para llevar	Argentina
Centro comercial Moctezuma	Mexico
Chop-suey Chinese	Switzerland
Comércio Mineiro	Brazil

## 10. AngularJS Events

AngularJS includes certain directives which can be used to provide custom behavior on various DOM events, such as click, dblclick, mouseenter etc.

The following table lists AngularJS event directives.

You can add AngularJS event listeners to your HTML elements by using one or more of these directives:

- ng-blur
- ng-change
- ng-click
- ng-copy
- ng-cut
- ng-dblclick
- ng-focus
- ng-keydown
- ng-keypress
- ng-keyup
- ng-mousedown
- ng-mouseenter
- ng-mouseleave



- ng-mousemove
- ng-mouseover
- ng-mouseup
- ng-paste

The event directives allow us to run AngularJS functions at certain user events.

An AngularJS event will not overwrite an HTML event, both events will be executed.

## 10.1 Mouse Events

Mouse events occur when the cursor moves over an element, in this order:

1. ng-mouseover
2. ng-mouseenter
3. ng-mousemove
4. ng-mouseleave

Or when a mouse button is clicked on an element, in this order:

1. ng-mousedown
2. ng-mouseup
3. ng-click

You can add mouse events on any HTML element

### Example

Increase the count variable when the mouse moves over the H1 element:

```
<div ng-app="myApp" ng-controller="myCtrl">

<h1 ng-mousemove="count = count + 1">Mouse over me!</h1>

<h2>{{ count }}</h2>

</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
 $scope.count = 0;
});
</script>
```

## OUTPUT:

### Mouse Over Me!

3

## 10.2 The ng-click Directive

The ng-click directive defines AngularJS code that will be executed when the element is being clicked.

### Example

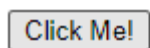
```
<div ng-app="myApp" ng-controller="myCtrl">

<button ng-click="count = count + 1">Click me!</button>

<p>{{ count }}</p>

</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
 $scope.count = 0;
});
</script>
```

## OUTPUT:



0

## 10.3 Toggle, True/False

If you want to show a section of HTML code when a button is clicked, and hide when the button is clicked again, like a dropdown menu, make the button behave like a toggle switch:

Click Me

### Example

```
<div ng-app="myApp" ng-controller="myCtrl">

<button ng-click="myFunc()">Click Me!</button>
```

```

<div ng-show="showMe">
 <h1>Menu:</h1>
 <div>Pizza</div>
 <div>Pasta</div>
 <div>Pesce</div>
</div>

</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
 $scope.showMe = false;
 $scope.myFunc = function() {
 $scope.showMe = !$scope.showMe;
 }
});
</script>

```

The showMe variable starts out as the Boolean value false.

The myFunc function sets the showMe variable to the opposite of what it is, by using the ! (not) operator.

## OUTPUT:

Click Me!

Click the button to show/hide the menu.

## 10.4 \$event Object

You can pass the \$event object as an argument when calling the function.

The \$event object contains the browser's event object:

Example

```

<div ng-app="myApp" ng-controller="myCtrl">

 <h1 ng-mousemove="myFunc($event)">Mouse Over Me!</h1>

 <p>Coordinates: {{ x + ', ' + y }}</p>

</div>
<script>

```

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
 $scope.myFunc = function(myE) {
 $scope.x = myE.clientX;
 $scope.y = myE.clientY;
 }
});
</script>
```

**OUTPUT:**

## Mouse Over Me!

Coordinates: ,

Mouse over the heading to display the value of clientX and clientY from the event object.

Forms in AngularJS provides data-binding and validation of input controls.

### 10.5 Input Controls

Input controls are the HTML input elements:

- input elements
- select elements
- button elements
- text area elements

#### Data-Binding

Input controls provides data-binding by using the ng-model directive.

```
<input type="text" ng-model="firstname">
```

The application does now have a property named firstname.

The ng-model directive binds the input controller to the rest of your application.

The property firstname, can be referred to in a controller:

#### Example

```
<script>
var app = angular.module('myApp', []);
app.controller('formCtrl', function($scope) {
```

```
$scope.firstname = "John";
});
</script>
```

## OUTPUT:

First Name:

## You entered:

Change the name inside the input field, and you will see the name in the header changes accordingly.

## Example

```
<form>
 First Name: <input type="text" ng-model="firstname">
</form>
```

```
<h1>You entered: {{firstname}}</h1>
```

First Name:

## Checkbox

A checkbox has the value true or false. Apply the ng-model directive to a checkbox, and use its value in your application.

## Example

Show the header if the checkbox is checked:

```
<form>
 Check to show a header:
 <input type="checkbox" ng-model="myVar">
</form>
```

```
<h1 ng-show="myVar">My Header</h1>
```

## OUTPUT:

Check to show a header: ☐

The header's ng-show attribute is set to true when the checkbox is checked.

## Radiobuttons

Bind radio buttons to your application with the ng-model directive.

Radio buttons with the same ng-model can have different values, but only the selected one will be used.

### Example

Display some text, based on the value of the selected radio button:

```
<form>
 Pick a topic:
 <input type="radio" ng-model="myVar" value="dogs">Dogs
 <input type="radio" ng-model="myVar" value="tuts">Tutorials
 <input type="radio" ng-model="myVar" value="cars">Cars
</form>
```

The value of myVar will be either dogs, tuts, or cars.

Pick a topic: ☐ Dogs ☐ Tutorials ☐ Cars

The ng-switch directive hides and shows HTML sections depending on the value of the radio buttons.

## Selectbox

Bind select boxes to your application with the ng-model directive.

The property defined in the ng-model attribute will have the value of the selected option in the selectbox.

### Example

Display some text, based on the value of the selected option:

```
<form>
 Select a topic:
 <select ng-model="myVar">
 <option value="">
 <option value="dogs">Dogs
 <option value="tuts">Tutorials
 <option value="cars">Cars
 </select>
</form>
```

### OUTPUT:

Select a topic:

The `ng-switch` directive hides and shows HTML sections depending on the value of the dropdown list.

The value of `myVar` will be either `dogs`, `tuts`, or `cars`.

## 11. An AngularJS Form Example

Top of Form

FirstName:

LastName:

RESET

Bottom of Form

```
form = { "firstName": "John", "lastName": "Doe" }
```

```
master = { "firstName": "John", "lastName": "Doe" }
```

Application Code

```
<div ng-app="myApp" ng-controller="formCtrl">
 <form novalidate>
 First Name:

 <input type="text" ng-model="user.firstName">

 Last Name:

 <input type="text" ng-model="user.lastName">

 <button ng-click="reset()">RESET</button>
 </form>
 <p>form = { {user} }</p>
 <p>master = { {master} }</p>
</div>
```

```
<script>
var app = angular.module('myApp', []);
app.controller('formCtrl', function($scope) {
 $scope.master = { firstName: "John", lastName: "Doe" };
 $scope.reset = function() {
 $scope.user = angular.copy($scope.master);
 };
 $scope.reset();
});
```

```
});
</script>
```

## OUTPUT:

First Name:

Last Name:

```
form = {"firstName":"John","lastName":"Doe"}
```

```
master = {"firstName":"John","lastName":"Doe"}
```

## 11.1 AngularJS Form Validation

AngularJS can validate input data

### Form Validation

AngularJS offers client-side form validation.

AngularJS monitors the state of the form and input fields (input, textarea, select), and lets you notify the user about the current state.

AngularJS also holds information about whether they have been touched, or modified, or not.

You can use standard HTML5 attributes to validate input, or you can make your own validation functions

### Required

Use the HTML5 attribute required to specify that the input field must be filled out:

### Example

The input field is required:

```
<form name="myForm">
 <input name="myInput" ng-model="myInput" required>
</form>
```

```
<p>The input's valid state is:</p>
<h1>{{ myForm.myInput.$valid }}</h1>
```



Try writing an E-mail address in the input field:

The input's valid state is:

**true**

Note that the state of the input field is "true" before you start writing in it, even if it does not contain an e-mail address.

### 11.3 Form State and Input State

AngularJS is constantly updating the state of both the form and the input fields.

Input fields have the following states:

- `$untouched` The field has not been touched yet
- `$touched` The field has been touched
- `$pristine` The field has not been modified yet
- `$dirty` The field has been modified
- `$invalid` The field content is not valid
- `$valid` The field content is valid

They are all properties of the input field, and are either true or false.

Forms have the following states:

- `$pristine` No fields have been modified yet
- `$dirty` One or more have been modified
- `$invalid` The form content is not valid
- `$valid` The form content is valid
- `$submitted` The form is submitted

They are all properties of the form, and are either true or false.

You can use these states to show meaningful messages to the user. Example, if a field is required, and the user leaves it blank, you should give the user a warning:

#### **Example**

Show an error message if the field has been touched AND is empty:

```
<input name="myName" ng-model="myName" required>
The name is
required.
```

Try leaving the first input field blank:

Name:

Address:

We use the `ng-show` directive to only show the error message if the field has been touched AND is empty.

## 11.4 CSS Classes

AngularJS adds CSS classes to forms and input fields depending on their states.

The following classes are added to, or removed from, input fields:

- `ng-untouched` The field has not been touched yet
- `ng-touched` The field has been touched
- `ng-pristine` The field has not been modified yet
- `ng-dirty` The field has been modified
- `ng-valid` The field content is valid
- `ng-invalid` The field content is not valid
- `ng-valid-key` One key for each validation. Example: `ng-valid-required`, useful when there are more than one thing that must be validated
- `ng-invalid-key` Example: `ng-invalid-required`

The following classes are added to, or removed from, forms:

- `ng-pristine` No fields has not been modified yet
- `ng-dirty` One or more fields has been modified
- `ng-valid` The form content is valid
- `ng-invalid` The form content is not valid
- `ng-valid-key` One key for each validation. Example: `ng-valid-required`, useful when there are more than one thing that must be validated
- `ng-invalid-key` Example: `ng-invalid-required`

The classes are removed if the value they represent is false.

Add styles for these classes to give your application a better and more intuitive user interface.

### Example

Apply styles, using standard CSS:

```
<style>
input.ng-invalid {
 background-color: pink;
}
input.ng-valid {
 background-color: lightgreen;
}

</style>
```

Try writing in the input field:

The input field requires content, and will therefore become green when you write in it.

Forms can also be styled:

### Example

Apply styles for unmodified (pristine) forms, and for modified forms:

```
<style>
form.ng-pristine {
 background-color: lightblue;
}
form.ng-dirty {
 background-color: pink;
}

</style>
```

Try writing in the input field:

The form gets a "ng-dirty" class when the form has been modified, and will therefore turn pink.

## 11.5 Custom Validation

To create your own validation function is a bit trickier; You have to add a new directive to your application, and deal with the validation inside a function with certain specified arguments.

## Example

Create your own directive, containing a custom validation function, and refer to it by using my-directive.

The field will only be valid if the value contains the character "e":

```
<form name="myForm">
<input name="myInput" ng-model="myInput" required my-directive>
</form>
```

```
<script>
```

```
var app = angular.module('myApp', []);
app.directive('myDirective', function() {
 return {
 require: 'ngModel',
 link: function(scope, element, attr, mCtrl) {
 function myValidation(value) {
 if (value.indexOf("e") > -1) {
 mCtrl.$setValidity('charE', true);
 } else {
 mCtrl.$setValidity('charE', false);
 }
 return value;
 }
 mCtrl.$parsers.push(myValidation);
 }
 };
});
```

Try writing in the input field:

The input's valid state is:

**false**

The input field must contain the character "e" to be consider valid.

```
</script>
```

In HTML, the new directive will be referred to by using the attribute my-directive.

In the JavaScript we start by adding a new directive named myDirective.

Remember, when naming a directive, you must use a camel case name, my Directive, but when invoking it, you must use - separated name, my-directive.

Then, return an object where you specify that we require ngModel, which is the ngModelController.

Make a linking function which takes some arguments, where the fourth argument, mCtrl, is the ngModelController,

Then specify a function, in this case named myValidation, which takes one argument, this argument is the value of the input element.

Test if the value contains the letter "e", and set the validity of the model controller to either true or false.

At last, mCtrl. \$parsers. Push(myValidation); will add the myValidation function to an array of other functions, which will be executed every time the input value changes.

### Validation Example

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<h2>Validation Example</h2>

<form ng-app="myApp" ng-controller="validateCtrl"
name="myForm" novalidate>

<p>Username:

 <input type="text" name="user" ng-model="user" required>

 Username is required.

</p>

<p>Email:

 <input type="email" name="email" ng-model="email" required>

 Email is required.
 Invalid email address.

</p>

<p>
```

```

<input type="submit"
ng-disabled="myForm.user.$dirty && myForm.user.$invalid ||
myForm.email.$dirty && myForm.email.$invalid">
</p>

</form>

<script>
var app = angular.module('myApp', []);
app.controller('validateCtrl', function($scope) {
 $scope.user = 'John Doe';
 $scope.email = 'john.doe@gmail.com';
});
</script>

</body>
</html>

```

## Validation Example

Username:

Email:



## 12.Jquery

- 1.jquery is a JavaScript Library.
- 2.jquery greatly simplifies JavaScript programming.
- 3.jquery is easy to learn.

With our online editor, you can edit the code, and click on a button to view the result.

### Example

```

$(document).ready(function(){
 $("p").click(function(){
 $(this).hide();
 });
});

```

## Output:

If you click on me, I will disappear.

Click me away!

Click me too!

**jQuery** There are lots of other JavaScript libraries out there, but jQuery is probably the most popular, and also the most extendable.

Many of the biggest companies on the Web use jQuery, such as:

- Google
- Microsoft
- IBM
- Netflix

jQuery is a lightweight, "write less, do more", JavaScript library.

The purpose of jQuery is to make it much easier to use JavaScript on your website.

jQuery takes a lot of common tasks that require many lines of JavaScript code to accomplish, and wraps them into methods that you can call with a single line of code.

jQuery also simplifies a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation.

The jQuery library contains the following features:

- HTML/DOM manipulation
- CSS manipulation
- HTML event methods
- Effects and animations
- AJAX
- Utilities

### 12.1 Adding jQuery to Your Web Pages

There are several ways to start using jQuery on your web site. You can:

- Download the jQuery library from [jquery.com](http://jquery.com)
- Include jQuery from a CDN, like Google

### 12.2 Downloading jQuery

There are two versions of jQuery available for downloading:

- Production version - this is for your live website because it has been minified and compressed
- Development version - this is for testing and development (uncompressed and readable code)

Both versions can be downloaded from [jQuery.com](https://jquery.com).

The jQuery library is a single JavaScript file, and you reference it with the HTML `<script>` tag (notice that the `<script>` tag should be inside the `<head>` section):

```
<head>
<script src="jquery-3.5.1.min.js"></script>
</head>
```

### 12.3 jQuery CDN

If you don't want to download and host jQuery yourself, you can include it from a CDN (Content Delivery Network).

Google is an example of someone who host jQuery:

#### Google CDN:

```
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
</head>
```

## This is a heading

This is a paragraph.

This is another paragraph.

Click me

With jQuery you select (query) HTML elements and perform "actions" on them.

### jQuery Syntax

The jQuery syntax is tailor-made for **selecting** HTML elements and performing some **action** on the element(s).

Basic syntax is: `$(selector).action()`

- A \$ sign to define/access jQuery



- A (*selector*) to "query (or find)" HTML elements
- A jQuery *action()* to be performed on the element(s)

Examples:

`$(this).hide()` - hides the current element.

`$("p").hide()` - hides all `<p>` elements.

`$(".test").hide()` - hides all elements with `class="test"`.

`$("#test").hide()` - hides the element with `id="test"`.

### The Document Ready Event

You might have noticed that all jQuery methods in our examples, are inside a document ready event:

```
$(document).ready(function(){
 //jQuery methods go here...
});
```

This is to prevent any jQuery code from running before the document is finished loading (is ready).

It is good practice to wait for the document to be fully loaded and ready before working with it. This also allows you to have your JavaScript code before the body of your document, in the head section.

Here are some examples of actions that can fail if methods are run before the document is fully loaded:

- Trying to hide an element that is not created yet
- Trying to get the size of an image that is not loaded yet
- `$(function(){`

```
 //jQuery methods go here...
});
```

- Use the syntax you prefer. We think that the document ready event is easier to understand when reading the code.

### jQuery Selectors

jQuery selectors are one of the most important parts of the jQuery library.

## 12.4jQuery Selectors

jQuery selectors allow you to select and manipulate HTML element(s).

jQuery selectors are used to "find" (or select) HTML elements based on their name, id, classes, types, attributes, values of attributes and much more. It's based on the existing CSS Selectors, and in addition, it has some own custom selectors.

All selectors in jQuery start with the dollar sign and parentheses: `$()`.

The element Selector

The jQuery element selector selects elements based on the element name.

You can select all `<p>` elements on a page like this:

```
$("p")
```

### Example

When a user clicks on a button, all `<p>` elements will be hidden:

Example:

```
$(document).ready(function(){
 $("button").click(function(){
 $("p").hide();
 });
});
```

**This is a heading**

This is a paragraph.

This is another paragraph.

Click me to hide paragraphs

The #id Selector

The jQuery #id selector uses the id attribute of an HTML tag to find the specific element.

An id should be unique within a page, so you should use the #id selector when you want to find a single, unique element.

To find an element with a specific id, write a hash character, followed by the id of the HTML element:

```
$("#test")
```

### Example

When a user clicks on a button, the element with id="test" will be hidden:

#### Example

```
$(document).ready(function(){
 $("button").click(function(){
 $("#test").hide();
 });
});
```

**This is a heading**

This is a paragraph.

This is another paragraph.

Click me

The .class Selector

The jQuery .class selector finds elements with a specific class.

To find elements with a specific class, write a period character, followed by the name of the class:

```
$(".test")
```

### Example

When a user clicks on a button, the elements with class="test" will be hidden:

```
$(document).ready(function(){
 $("button").click(function(){
 $(".test").hide();
 });
});
```

# This is a heading

This is a paragraph.

This is another paragraph.

Click me

## 12.6 Functions In a Separate File

If your website contains a lot of pages, and you want your jQuery functions to be easy to maintain, you can put your jQuery functions in a separate .js file.

When we demonstrate jQuery in this tutorial, the functions are added directly into the <head> section. However, sometimes it is preferable to place them in a separate file, like this (use the src attribute to refer to the .js file):

### Example

```
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script src="my_jquery_functions.js"></script>
</head>
```

jQuery is tailor-made to respond to events in an HTML page.

What are Events?

All the different visitors' actions that a web page can respond to are called events.

An event represents the precise moment when something happens.

Examples:

- moving a mouse over an element
- selecting a radio button
- clicking on an element

The term "**fires/fired**" is often used with events. Example: "The keypress event is fired; the moment you press a key".

Here are some common DOM events:

Table 4.2:DOM Events

Mouse Events	Keyboard Events	Form Events	Document/Window Events
click	keypress	submit	load
dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave		blur	unload

### jQuery Syntax For Event Methods

In jQuery, most DOM events have an equivalent jQuery method.

To assign a click event to all paragraphs on a page, you can do this:

```
$("#p"). click();
```

The next step is to define what should happen when the event fires. You must pass a function to the event:

```
$("#p").click(function(){
 // action goes here!!
});
```

### Commonly Used jQuery Event Methods

#### **\$(document). ready()**

The `$(document).ready()` method allows us to execute a function when the document is fully loaded. This event is already explained in the jQuery Syntax chapter.

## click()

The click() method attaches an event handler function to an HTML element.

The function is executed when the user clicks on the HTML element.

The following example says: When a click event fires on a <p> element; hide the current <p> element:

### Example

```
$("#p").click(function(){
 $(this).hide();
});
```

If you click on me, I will disappear.

Click me away!

Click me too!

## dblclick()

The dblclick() method attaches an event handler function to an HTML element.

The function is executed when the user double-clicks on the HTML element:

### Example

```
$("#p").dblclick(function(){
 $(this).hide();
});
```

If you double-click on me, I will disappear.

Click me away!

Click me too!

## mouseenter()

The mouseenter() method attaches an event handler function to an HTML element.

The function is executed when the mouse pointer enters the HTML element:

### Example

```
$("#p1").mouseenter(function(){
 alert("You entered p1!");
});
```

### OUTPUT

Enter this paragraph.

An embedded page on this page says

You entered p1!

OK

### mouseleave()

The mouseleave() method attaches an event handler function to an HTML element.

The function is executed when the mouse pointer leaves the HTML element:

### Example

```
$("#p1").mouseleave(function(){
 alert("Bye! You now leave p1!");
});
```

### OUTPUT:

This is a paragraph.

An embedded page on this page says

Bye! You now leave p1!

OK

### mousedown ()

The mousedown () method attaches an event handler function to an HTML element.

The function is executed, when the left, middle or right mouse button is pressed down, while the mouse is over the HTML element:

#### Example

```
$("#p1").mousedown(function(){
 alert("Mouse down over p1!");
});
```

#### OUTPUT:

This is a paragraph.

#### mouseup ()

The mouseup () method attaches an event handler function to an HTML element.

The function is executed, when the left, middle or right mouse button is released, while the mouse is over the HTML element:

#### Example

```
$("#p1").mouseup(function(){
 alert("Mouse up over p1!");
});
```

#### OUTPUT:

This is a paragraph.

#### hover()

The hover () method takes two functions and is a combination of the mouseenter () and mouseleave () methods.

The first function is executed when the mouse enters the HTML element, and the second function is executed when the mouse leaves the HTML element:

#### Example

```
$("#p1").hover(function(){
 alert("You entered p1!");
},
```



```
function(){
 alert("Bye! You now leave p1!");
};
```

## OUTPUT:

Enter this paragraph.

An embedded page on this page says

You entered p1!

OK

## focus()

The focus() method attaches an event handler function to an HTML form field.

The function is executed when the form field gets focus:

### Example

```
$("input").focus(function(){
 $(this).css("background-color", "#cccccc");
});
```

Name:   
Email:

## blur()

The blur() method attaches an event handler function to an HTML form field.

The function is executed when the form field loses focus:

### Example

```
$("input").blur(function(){
 $(this).css("background-color", "#ffffff");
});
```

Name:   
Email:

## The on() Method

The on() method attaches one or more event handlers for the selected elements.

Attach a click event to a <p> element:

### Example

```
$("#p").on("click", function(){
 $(this).hide();
});
```

### OUTPUT:

If you click on me, I will disappear.

Click me away!

Click me too!

Attach multiple event handlers to a <p> element:

### Example

```
$("#p").on({
 mouseenter: function(){
 $(this).css("background-color", "lightgray");
 },
 mouseleave: function(){
 $(this).css("background-color", "lightblue");
 },
 click: function(){
 $(this).css("background-color", "yellow");
 }
});
```

### OUTPUT:

Click or move the mouse pointer over this paragraph.

## 13.JQuery Effects

Hide, Show, Toggle, Slide, Fade, and Animate.

Examples

jQueryhide()

Demonstrates a simple jQuery hide() method.

jQueryhide()

Another hide() demonstration. How to hide parts of text.

### 13.1 jQuery hide() and show()

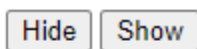
With jQuery, you can hide and show HTML elements with the hide() and show() methods:

#### Example

```
$("#hide").click(function(){
 $("p").hide();
});
```

```
$("#show").click(function(){
 $("p").show();
});
```

If you click on the "Hide" button, I will disappear.



#### Syntax:

```
$(selector).hide(speed,callback);
```

```
$(selector).show(speed,callback);
```

The optional speed parameter specifies the speed of the hiding/showing, and can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the hide() or show() method completes (you will learn more about callback functions in a later chapter).

The following example demonstrates the speed parameter with hide():

#### Example

```
$("#button").click(function(){
 $("p").hide(1000);
});
```

Hide

This is a paragraph with little content.

This is another small paragraph.

### 13.2 jQuery toggle()

You can also toggle between hiding and showing an element with the toggle() method.

Shown elements are hidden and hidden elements are shown:

#### Example

```
$("#button").click(function(){
 $("#p").toggle();
});
```

Toggle between hiding and showing the paragraphs

This is a paragraph with little content.

This is another small paragraph.

#### Syntax:

```
$(selector). toggle(speed,callback);
```

The optional speed parameter can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after toggle () completes.

### jQuery Effects - Fading

With jQuery you can fade elements in and out of visibility.

#### Examples

##### jQueryfadeIn()

Demonstrates the jQuery fadeIn() method.

##### jQueryfadeOut()

Demonstrates the jQuery fadeOut() method.

jQueryfadeToggle()

Demonstrates the jQuery fadeToggle() method.

jQueryfadeTo()

Demonstrates the jQuery fadeTo() method.

### 13.3 jQuery Fading Methods

With jQuery you can fade an element in and out of visibility.

jQuery has the following fade methods:

- fadeIn()
- fadeOut()
- fadeToggle()
- fadeTo()

### 13.4 jQuery fadeIn() Method

The jQuery fadeIn() method is used to fade in a hidden element.

#### **Syntax:**

`$(selector).fadeIn(speed,callback);`

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the fading completes.

The following example demonstrates the fadeIn() method with different parameters:

#### **Example**

```
$("#button").click(function(){
 $("#div1").fadeIn();
 $("#div2").fadeIn("slow");
 $("#div3").fadeIn(3000);
});
```

Demonstrate fadeIn() with different parameters.

Click to fade in boxes

### 13.5 jQuery fadeOut() Method

The jQuery fadeOut() method is used to fade out a visible element.

#### Syntax:

```
$(selector).fadeOut(speed,callback);
```

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the fading completes.

The following example demonstrates the fadeOut() method with different parameters:

#### Example

```
$("#button").click(function(){
 $("#div1").fadeOut();
 $("#div2").fadeOut("slow");
 $("#div3").fadeOut(3000);
});
```

Demonstrate fadeOut() with different parameters.

Click to fade out boxes



### 13.6 jQuery fadeToggle() Method

The jQuery fadeToggle() method toggles between the fadeIn() and fadeOut() methods.

If the elements are faded out, fadeToggle() will fade them in.

If the elements are faded in, fadeToggle() will fade them out.

**Syntax:**

`$(selector).fadeToggle(speed,callback);`

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the fading completes.

The following example demonstrates the fadeToggle() method with different parameters:

**Example**

```
$("#button").click(function(){
 $("#div1").fadeToggle();
 $("#div2").fadeToggle("slow");
 $("#div3").fadeToggle(3000);
});
```

Demonstrate fadeToggle() with different speed parameters.

Click to fade in/out boxes



### 13.7 jQuery fadeTo() Method

The jQuery fadeTo() method allows fading to a given opacity (value between 0 and 1).

**Syntax:**

`$(selector).fadeTo(speed,opacity,callback);`

The required speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The required opacity parameter in the fadeTo() method specifies fading to a given opacity (value between 0 and 1).

The optional callback parameter is a function to be executed after the function completes.

The following example demonstrates the fadeTo() method with different parameters:

### Example

```
$("#button").click(function(){
 $("#div1").fadeTo("slow", 0.15);
 $("#div2").fadeTo("slow", 0.4);
 $("#div3").fadeTo("slow", 0.7);
});
```

Demonstrate fadeTo() with different parameters.

Click to fade boxes



## 13.8 jQuery Effects - Sliding

The jQuery slide methods slide elements up and down.

### Examples

jQueryslideDown()

Demonstrates the jQuery slideDown() method.



jQueryslideUp()

Demonstrates the jQuery slideUp() method.

jQueryslideToggle()

Demonstrates the jQuery slideToggle() method.

## 13.9 jQuery Sliding Methods

With jQuery you can create a sliding effect on elements.

jQuery has the following slide methods:

- slideDown()
- slideUp()
- slideToggle()

### 13.10 jQuery slideDown() Method

The jQuery slideDown() method is used to slide down an element.

#### **Syntax:**

```
$(selector).slideDown(speed,callback);
```

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the sliding completes.

The following example demonstrates the slideDown() method:

#### **Example**

```
$("#flip").click(function(){
 $("#panel").slideDown();
});
```

#### **OUTPUT:**

Click to slide down panel

### 13.11 jQuery slideUp() Method

The jQuery slideUp() method is used to slide up an element.

**Syntax:**

```
$(selector).slideUp(speed,callback);
```

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

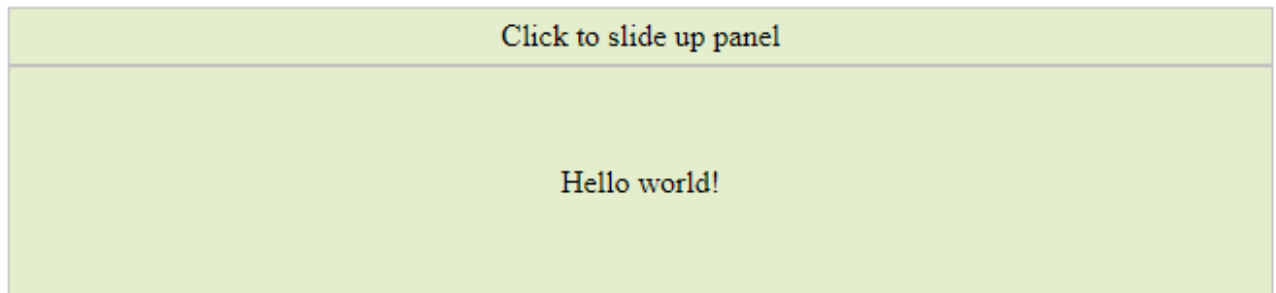
The optional callback parameter is a function to be executed after the sliding completes.

The following example demonstrates the slideUp() method:

**Example**

```
$("#flip").click(function(){
 $("#panel").slideUp();
});
```

OUTPUT:



### 13.12jQuery slideToggle() Method

The jQuery slideToggle() method toggles between the slideDown() and slideUp() methods.

If the elements have been slid down, slideToggle() will slide them up.

If the elements have been slid up, slideToggle() will slide them down.

```
$(selector).slideToggle(speed,callback);
```

The optional speed parameter can take the following values: "slow", "fast", milliseconds.

The optional callback parameter is a function to be executed after the sliding completes.

The following example demonstrates the slideToggle() method:

**Example**

```
$("#flip").click(function(){
 $("#panel").slideToggle();
});
```

## OUTPUT:

Click to slide the panel down or up

## 14.Jquery Animation:

With jQuery, you can create custom animations.

jQuery Animations - The animate () Method

The jQuery animate () method is used to create custom animations.

### Syntax:

```
$(selector).animate({params},speed,callback);
```

The required params parameter defines the CSS properties to be animated.

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the animation completes.

The following example demonstrates a simple use of the animate () method; it moves a <div> element to the right, until it has reached a left property of 250px:

### Example

```
$("#button").click(function(){
 $("#div").animate({left: '250px'});
});
```

## OUTPUT:

Start Animation

By default, all HTML elements have a static position, and cannot be moved. To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!



## jQuery animate() - Manipulate Multiple Properties

Notice that multiple properties can be animated at the same time:

### Example

```
$("#button").click(function(){
 $("#div").animate({
 left: '250px',
 opacity: '0.5',
 height: '150px',
 width: '150px'
 });
});
```

### OUTPUT:

Start Animation

By default, all HTML elements have a static position, and cannot be moved. To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!



## jQuery animate() - Using Relative Values

It is also possible to define relative values (the value is then relative to the element's current value). This is done by putting += or -= in front of the value:

### Example

```
$("#button").click(function(){
 $("#div").animate({
 left: '250px',
 height: '+=150px',
 width: '+=150px'
 });
});
```

## OUTPUT:

Start Animation

By default, all HTML elements have a static position, and cannot be moved. To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!



## jQuery animate() - Using Pre-defined Values

You can even specify a property's animation value as "show", "hide", or "toggle":

### Example

```
$("#button").click(function(){
 $("#div").animate({
 height: 'toggle'
 });
});
```

## OUTPUT:

Start Animation

By default, all HTML elements have a static position, and cannot be moved. To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!



## jQuery Stop Animations

The jQuery stop() method is used to stop animations or effects before it is finished.

jQuerystop()sliding

Demonstrates the jQuery stop() method.

jQuerystop()animation(withparameters)  
Demonstrates the jQuery stop() method.

## jQuery stop() Method

The jQuery stop() method is used to stop an animation or effect before it is finished.

The stop() method works for all jQuery effect functions, including sliding, fading and custom animations.

### Syntax:

```
$(selector).stop(stopAll,goToEnd);
```

The optional stopAll parameter specifies whether also the animation queue should be cleared or not. Default is false, which means that only the active animation will be stopped, allowing any queued animations to be performed afterwards.

The optional goToEnd parameter specifies whether or not to complete the current animation immediately. Default is false.

So, by default, the stop() method kills the current animation being performed on the selected element.

The following example demonstrates the stop() method, with no parameters:

### Example

```
$("#stop").click(function(){
 $("#panel").stop();
});
```

### OUTPUT:



## 16.jQuery Callback Functions

A callback function is executed after the current effect is 100% finished.

### jQuery Callback Functions

JavaScript statements are executed line by line. However, with effects, the next line of code can be run even though the effect is not finished. This can create errors.

To prevent this, you can create a callback function.

A callback function is executed after the current effect is finished.

Typical syntax: `$(selector).hide(speed,callback);`

## Examples

The example below has a callback parameter that is a function that will be executed after the hide effect is completed:

### Example with Callback

```
$("#button").click(function(){
 $("#p").hide("slow", function(){
 alert("The paragraph is now hidden");
 });
});
```

### OUTPUT:



This is a paragraph with little content.

The example below has no callback parameter, and the alert box will be displayed before the hide effect is completed:

### Example without Callback

```
$("#button").click(function(){
 $("#p").hide(1000);
 alert("The paragraph is now hidden");
});
```

### OUTPUT:



This is a paragraph with little content.

## jQuery - Chaining

With jQuery, you can chain together actions/methods.

Chaining allows us to run multiple jQuery methods (on the same element) within a single statement.

### jQuery Method Chaining

Until now we have been writing jQuery statements one at a time (one after the other).

However, there is a technique called chaining, that allows us to run multiple jQuery commands, one after the other, on the same element(s).

**Tip:** This way, browsers do not have to find the same element(s) more than once.

To chain an action, you simply append the action to the previous action.

The following example chains together the `css()`, `slideUp()`, and `slideDown()` methods. The "p1" element first changes to red, then it slides up, and then it slides down:

#### Example

```
$("#p1").css("color", "red").slideUp(2000).slideDown(2000);
```

**OUTPUT:**

jQuery is fun!!

Click me

We could also have added more method calls if needed.

When chaining, the line of code could become quite long. However, jQuery is not very strict on the syntax; you can format it like you want, including line breaks and indentations.

This also works just fine:

#### Example

```
$("#p1").css("color", "red")
 .slideUp(2000)
 .slideDown(2000);
```

jQuery throws away extra whitespace and executes the lines above as one long line of code.

**OUTPUT:**

jQuery is fun!!

Click me

## jQuery HTML

### jQuery - Get Content and Attributes

jQuery contains powerful methods for changing and manipulating HTML elements and attributes.

jQuery DOM Manipulation



One very important part of jQuery is the possibility to manipulate the DOM.

jQuery comes with a bunch of DOM related methods that make it easy to access and manipulate elements and attributes.

DOM = Document Object Model

The DOM defines a standard for accessing HTML and XML documents:

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

Get Content - text(), html(), and val()

Three simple, but useful, jQuery methods for DOM manipulation are:

- text() - Sets or returns the text content of selected elements
- html() - Sets or returns the content of selected elements (including HTML markup)
- val() - Sets or returns the value of form fields

The following example demonstrates how to get content with the jQuery text() and html() methods:

#### Example

```
$("#btn1").click(function(){
 alert("Text: " + $("#test").text());
});
$("#btn2").click(function(){
 alert("HTML: " + $("#test").html());
});
```

#### OUTPUT:

This is some **bold** text in a paragraph.

Show Text

Show HTML

The following example demonstrates how to get the value of an input field with the jQuery val() method:

### Example

```
$("#btn1").click(function(){
 alert("Value: " + $("#test").val());
});
```

OUTPUT:

Name:

Get Attributes - attr()

The jQuery attr() method is used to get attribute values.

The following example demonstrates how to get the value of the href attribute in a link:

### Example

```
$("#button").click(function(){
 alert($("#w3s").attr("href"));
});
```

[W3Schools.com](https://www.w3schools.com)

## jQuery - Set Content and Attributes

Set Content - text(), html(), and val()

We will use the same three methods from the previous page to **set content**:

- text() - Sets or returns the text content of selected elements
- html() - Sets or returns the content of selected elements (including HTML markup)
- val() - Sets or returns the value of form fields

The following example demonstrates how to set content with the jQuery text(), html(), and val() methods:

### Example

```
$("#btn1").click(function(){
 $("#test1").text("Hello world!");
});
$("#btn2").click(function(){
```

```
$("#test2").html("Hello world!");
});
$("#btn3").click(function(){
 $("#test3").val("Dolly Duck");
});
```

OUTPUT:

This is a paragraph.

This is another paragraph.

Input field:

A Callback Function for text(), html(), and val()

All of the three jQuery methods above: text(), html(), and val(), also come with a callback function. The callback function has two parameters: the index of the current element in the list of elements selected and the original (old) value. You then return the string you wish to use as the new value from the function.

The following example demonstrates text() and html() with a callback function:

### Example

```
$("#btn1").click(function(){
 $("#test1").text(function(i, origText){
 return "Old text: " + origText + " New text: Hello world!
 (index: " + i + ")";
 });
});

$("#btn2").click(function(){
 $("#test2").html(function(i, origText){
 return "Old html: " + origText + " New html: Hello world!
 (index: " + i + ")";
 });
});
```

This is a **bold** paragraph.

This is another **bold** paragraph.

## Set Attributes - attr()

The jQuery attr() method is also used to set/change attribute values.

The following example demonstrates how to change (set) the value of the href attribute in a link:

### Example

```
$("#button").click(function(){
 $("#w3s").attr("href", "https://www.w3schools.com/jquery/");
});
```

### OUTPUT:

[W3Schools.com](https://www.w3schools.com/jquery/)

Change href Value

Mouse over the link (or click on it) to see that the value of the href attribute has changed.

The attr() method also allows you to set multiple attributes at the same time.

The following example demonstrates how to set both the href and title attributes at the same time:

### Example

```
$("#button").click(function(){
 $("#w3s").attr({
 "href" : "https://www.w3schools.com/jquery/",
 "title" : "W3Schools jQuery Tutorial"
 });
});
```

### OUTPUT:

[W3Schools.com](https://www.w3schools.com/jquery/)

Change href and title

Mouse over the link to see that the href attribute has changed and a title attribute is set.

## A Callback Function for attr()

The jQuery method attr(), also comes with a callback function. The callback function has two parameters: the index of the current element in the list of elements selected and the original (old) attribute value. You then return the string you wish to use as the new attribute value from the function.

The following example demonstrates attr() with a callback function:

### Example

```
$("#button").click(function(){
 $("#w3s").attr("href", function(i, origValue){
 return origValue + "/jquery/";
 });
});
```

[W3Schools.com](https://www.w3schools.com/jquery/jquery_attr.asp)

Change href Value

Mouse over the link (or click on it) to see that the value of the href attribute has changed.

### jQuery - Add Elements

With jQuery, it is easy to add new elements/content.

#### Add New HTML Content

We will look at four jQuery methods that are used to add new content:

- append() - Inserts content at the end of the selected elements
- prepend() - Inserts content at the beginning of the selected elements
- after() - Inserts content after the selected elements
- before() - Inserts content before the selected elements

#### jQuery append() Method

The jQuery append() method inserts content AT THE END of the selected HTML elements.

### Example

```
$("#p").append("Some appended text.");
```

### OUTPUT:

This is a paragraph.

This is another paragraph.

1. List item 1
2. List item 2
3. List item 3

Append text

Append list items

### jQuery prepend() Method

The jQuery prepend() method inserts content AT THE BEGINNING of the selected HTML elements.

#### Example

```
$("p").prepend("Some prepended text.");
```

OUTPUT:

This is a paragraph.

This is another paragraph.

1. List item 1
2. List item 2
3. List item 3

Append text

Append list items

### Add Several New Elements With append() and prepend()

In both examples above, we have only inserted some text/HTML at the beginning/end of the selected HTML elements.

However, both the append() and prepend() methods can take an infinite number of new elements as parameters. The new elements can be generated with text/HTML (like we have done in the examples above), with jQuery, or with JavaScript code and DOM elements.

In the following example, we create several new elements. The elements are created with text/HTML, jQuery, and JavaScript/DOM. Then we append the new elements to the text with the append() method (this would have worked for prepend() too) :

### Example

```
function appendText() {
 var txt1 = "<p>Text.</p>"; // Create element with HTML
 var txt2 = $("<p></p>").text("Text."); // Create with jQuery
 var txt3 = document.createElement("p"); // Create with DOM
 txt3.innerHTML = "Text.";
 $("body").append(txt1, txt2, txt3); // Append the new elements
}
```

### jQuery after() and before() Methods

The jQuery after() method inserts content AFTER the selected HTML elements.

The jQuery before() method inserts content BEFORE the selected HTML elements.

### Example

```
$("#img").after("Some text after");
```

```
$("#img").before("Some text before");
```

### Add Several New Elements With after() and before()

Also, both the after() and before() methods can take an infinite number of new elements as parameters. The new elements can be generated with text/HTML (like we have done in the example above), with jQuery, or with JavaScript code and DOM elements.

In the following example, we create several new elements. The elements are created with text/HTML, jQuery, and JavaScript/DOM. Then we insert the new elements to the text with the after() method (this would have worked for before() too) :

### Example

```
function afterText() {
 var txt1 = "I "; // Create element with HTML
 var txt2 = $("<i></i>").text("love "); // Create with jQuery
 var txt3 = document.createElement("b"); // Create with DOM
 txt3.innerHTML = "jQuery!";
 $("#img").after(txt1, txt2, txt3); // Insert new elements after
}
```

### jQuery - Remove Elements

With jQuery, it is easy to remove existing HTML elements.

### Remove Elements/Content

To remove elements and content, there are mainly two jQuery methods:

- `remove()` - Removes the selected element (and its child elements)
- `empty()` - Removes the child elements from the selected element

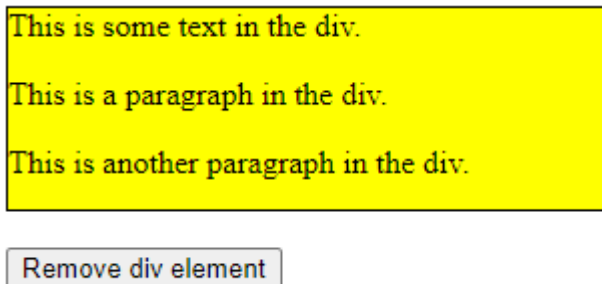
### jQuery `remove()` Method

The jQuery `remove()` method removes the selected element(s) and its child elements.

#### Example

```
$("#div1").remove();
```

#### OUTPUT:



### 17.jQuery Traversing

#### Traversing

jQuery traversing, which means "move through", are used to "find" (or select) HTML elements based on their relation to other elements. Start with one selection and move through that selection until you reach the elements you desire.

The image below illustrates an HTML page as a tree (DOM tree). With jQuery traversing, you can easily move up (ancestors), down (descendants) and sideways (siblings) in the tree, starting from the selected (current) element. This movement is called traversing - or moving through - the DOM tree.

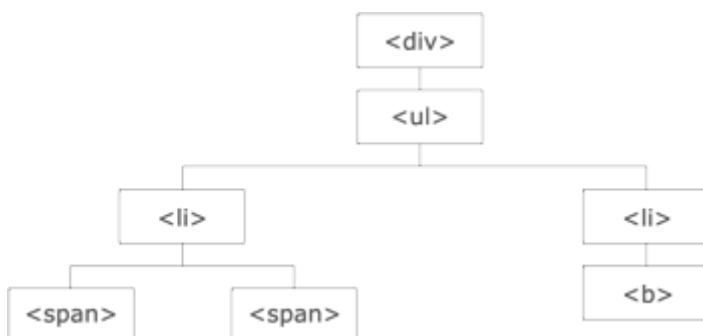


Fig.4.3. DOM Tree

#### Illustration explained:

- The `<div>` element is the **parent** of `<ul>`, and an **ancestor** of everything inside of it
- The `<ul>` element is the **parent** of both `<li>` elements, and a **child** of `<div>`



- The left <li> element is the **parent** of <span>, **child** of <ul> and a **descendant** of <div>
- The <span> element is a **child** of the left <li> and a **descendant** of <ul> and <div>
- The two <li> elements are **siblings** (they share the same parent)
- The right <li> element is the **parent** of <b>, **child** of <ul> and a **descendant** of <div>
- The <b> element is a **child** of the right <li> and a **descendant** of <ul> and <div>
- An ancestor is a parent, grandparent, great-grandparent, and so on.  
A descendant is a child, grandchild, great-grandchild, and so on.  
Siblings share the same parent.

## 17.1 Traversing the DOM

jQuery provides a variety of methods that allow us to traverse the DOM.

The largest category of traversal methods is tree-traversal.

### jQuery Traversing - Ancestors

With jQuery you can traverse up the DOM tree to find ancestors of an element.

An ancestor is a parent, grandparent, great-grandparent, and so on.

#### Traversing Up the DOM Tree

Three useful jQuery methods for traversing up the DOM tree are:

- `parent()`
- `parents()`
- `parentsUntil()`

#### jQuery `parent()` Method

The `parent()` method returns the direct parent element of the selected element.

This method only traverse a single level up the DOM tree.

The following example returns the direct parent element of each <span> elements:

Example

```
$(document).ready(function(){
 $("span").parent();
});
```

#### jQuery `parents()` Method

The `parents()` method returns all ancestor elements of the selected element, all the way up to the document's root element (`<html>`).

The following example returns all ancestors of all `<span>` elements:

Example

```
$(document).ready(function(){
 $("span").parents();
});
```

You can also use an optional parameter to filter the search for ancestors.

The following example returns all ancestors of all `<span>` elements that are `<ul>` elements:

Example

```
$(document).ready(function(){
 $("span").parents("ul");
});
```

### jQuery `parentsUntil()` Method

The `parentsUntil()` method returns all ancestor elements between two given arguments.

The following example returns all ancestor elements between a `<span>` and a `<div>` element:

Example

```
$(document).ready(function(){
 $("span").parentsUntil("div");
});
```

### jQuery Traversing - Descendants

With jQuery you can traverse down the DOM tree to find descendants of an element.

A descendant is a child, grandchild, great-grandchild, and so on.

#### Traversing Down the DOM Tree

Two useful jQuery methods for traversing down the DOM tree are:

- `children()`
- `find()`

### jQuery `children()` Method

The `children()` method returns all direct children of the selected element.

This method only traverses a single level down the DOM tree.

The following example returns all elements that are direct children of each `<div>` elements:

Example

```
$(document).ready(function(){
 $("div").children();
});
```

You can also use an optional parameter to filter the search for children.

The following example returns all `<p>` elements with the class name "first", that are direct children of `<div>`:

Example

```
$(document).ready(function(){
 $("div").children("p.first");
});
```

jQuery `find()` Method

The `find()` method returns descendant elements of the selected element, all the way down to the last descendant.

The following example returns all `<span>` elements that are descendants of `<div>`:

Example

```
$(document).ready(function(){
 $("div").find("span");
});
```

The following example returns all descendants of `<div>`:

Example

```
$(document).ready(function(){
 $("div").find("*");
});
```

## jQuery Traversing - Siblings

With jQuery you can traverse sideways in the DOM tree to find siblings of an element.

Siblings share the same parent.

## Traversing Sideways in The DOM Tree

There are many useful jQuery methods for traversing sideways in the DOM tree:

- `siblings()`
- `next()`
- `nextAll()`
- `nextUntil()`
- `prev()`
- `prevAll()`
- `prevUntil()`

### jQuery `siblings()` Method

The `siblings()` method returns all sibling elements of the selected element.

The following example returns all sibling elements of `<h2>`:

Example

```
$(document).ready(function(){
 $("h2").siblings();
});
```

You can also use an optional parameter to filter the search for siblings.

The following example returns all sibling elements of `<h2>` that are `<p>` elements:

Example

```
$(document).ready(function(){
 $("h2").siblings("p");
});
```

### jQuery `next()` Method

The `next()` method returns the next sibling element of the selected element.

The following example returns the next sibling of `<h2>`:

Example

```
$(document).ready(function(){
 $("h2").next();
});
```

### jQuery `nextAll()` Method

The `nextAll()` method returns all next sibling elements of the selected element.

The following example returns all next sibling elements of <h2>:

Example

```
$(document).ready(function(){
 $("h2").nextAll();
});
```

jQuery nextUntil() Method

The nextUntil() method returns all next sibling elements between two given arguments.

The following example returns all sibling elements between a <h2> and a <h6> element:

Example

```
$(document).ready(function(){
 $("h2").nextUntil("h6");
});
```

jQuery prev(), prevAll() & prevUntil() Methods

The prev(), prevAll() and prevUntil() methods work just like the methods above but with reverse functionality: they return previous sibling elements (traverse backwards along sibling elements in the DOM tree, instead of forward).

## jQuery Traversing - Filtering

The first(), last(), eq(), filter() and not() Methods

The most basic filtering methods are first(), last() and eq(), which allow you to select a specific element based on its position in a group of elements.

Other filtering methods, like filter() and not() allow you to select elements that match, or do not match, a certain criteria.

jQuery first() Method

The first() method returns the first element of the specified elements.

The following example selects the first <div> element:

Example

```
$(document).ready(function(){
 $("div").first();
});
```

jQuery last() Method

The last() method returns the last element of the specified elements.

The following example selects the last <div> element:

Example

```
$(document).ready(function(){
 $("div").last();
});
```

jQuery eq() method

The eq() method returns an element with a specific index number of the selected elements.

The index numbers start at 0, so the first element will have the index number 0 and not 1. The following example selects the second <p> element (index number 1):

Example

```
$(document).ready(function(){
 $("p").eq(1);
});
```

jQuery filter() Method

The filter() method lets you specify a criteria. Elements that do not match the criteria are removed from the selection, and those that match will be returned.

The following example returns all <p> elements with class name "intro":

Example

```
$(document).ready(function(){
 $("p").filter(".intro");
});
```

jQuery not() Method

The not() method returns all elements that do not match the criteria.

**Tip:** The not() method is the opposite of filter().

The following example returns all <p> elements that do not have class name "intro":

Example

```
$(document).ready(function(){
 $("p").not(".intro");
});
```

## QUESTION BANK

S.No	Questions (2 marks)	Competence	BT Level
1.	Explain the differences between Angular and jQuery	Understand	BTL 2
2.	Identify what is the Ahead of Time Compilation?	Analyze	BTL 4
3.	Explain ngOnInit () function	Understand	BTL 2
4.	List out the difference between Angular Component and Directive	Remember	BTL 1
5.	Categorize different phases of the AngularJS Scope lifecycle.	Create	BTL 5
6.	Determine the AngularJS components that can be injected as dependency	Create	BTL 5
7.	Brief the use of \$routeProvider	Create	BTL 5
8.	Distinguish between AngularJS and backbone.js	Understand	BTL 2
9.	Differentiate between a link and compile in Angular.js?	Analyze	BTL 4
10.	Brief the linking function and its type	Create	BTL 5
11.	Explain directives and their types	Analyze	BTL 4
12.	Define \$rootScope and how do we use it?	Apply	BTL 3
13.	Organize the step to initialize a select box with options on page load.	Analyze	BTL 4
14.	Define Dependency Injection.	Remember	BTL 1
15.	Create a program for to bootstrap process in Angular	Create	BTL 6
16.	Compare and Contrast prop() and attr().	Analyze	BTL 4
17.	jQuery a JavaScript or JSON library file justify.	Analyze	BTL 4
18.	Illustrate the clone and Object in jQuery	Apply	BTL 3
19.	Create a simple code to disable Browser's back button in jQuery.	Analyze	BTL 2

<b>S.No</b>	<b>Questions (16 marks)</b>	<b>Competence</b>	<b>BT Level</b>
1.	Describe the architecture of MVC	Remember	BTL 1
2.	Illustrate the Sample Angular Powered View with example	Analyze	BTL 4
3.	Develop a simple code fragment using NgFor. Create a file called mock-heroes.ts in the user defined folder and Define a HEROES constant as an array of ten heroes and export it and list the name of the heros.	Evaluate	BTL 5
4.	List and Explain the Different data binding methods in angularjs with syntax.	Apply	BTL 3
5.	Demonstrate passing data to directive type scope with sample coding fragment	Apply	BTL 3
6.	Develop a code for printing the fees using filters	Create	BTL 6
7.	Explain function scope and service in AngularJS	Understand	BTL 2
8.	Write a program to hide an HTML tag just by one button click in angular	Evaluate	BTL 5
9.	Discuss the jQuery function used to provide effects.	Analyze	BTL 4
10.	List the different slide effects available in jQuery.	Remember	BTL 1
11.	Illustrate the various CSS classes used to manipulate in HTML using jQuery	Apply	BTL 3
12.	Create the jQuery code to select all links inside the paragraph	Create	BTL 6





**SATHYABAMA**

INSTITUTE OF SCIENCE AND TECHNOLOGY  
(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

[www.sathyabama.ac.in](http://www.sathyabama.ac.in)

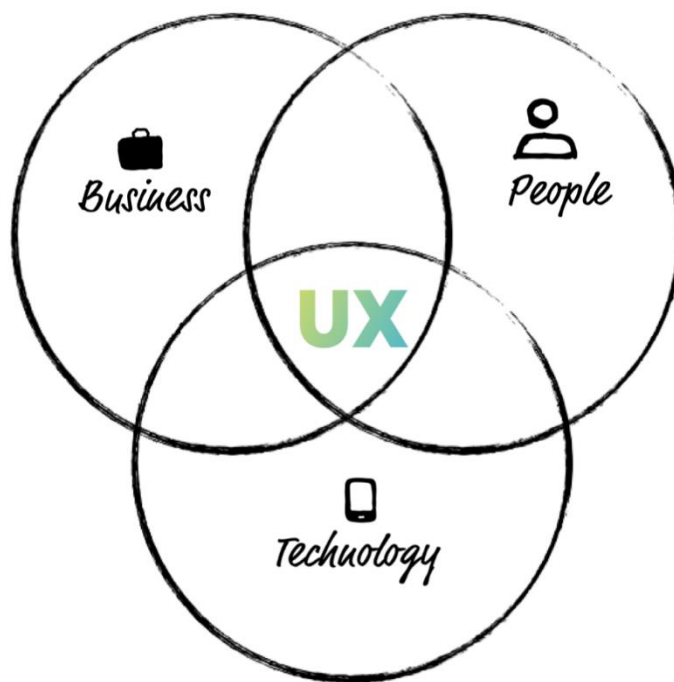
## UNIT –V - Customer Interface Design and Development– SITA1502

## UNIT 5 UX AND UI

UX Introduction -Elements of UX Design- UX Design Process- Research Methods and Tools- Understanding User Needs and Goals. UX Design Process: Visual Design Principles-Information Design and Visualization-Interaction Design- Prototyping Tools-Usability Test. UI Introduction- User Interface Components -Tools and Processes.

### UX Introduction

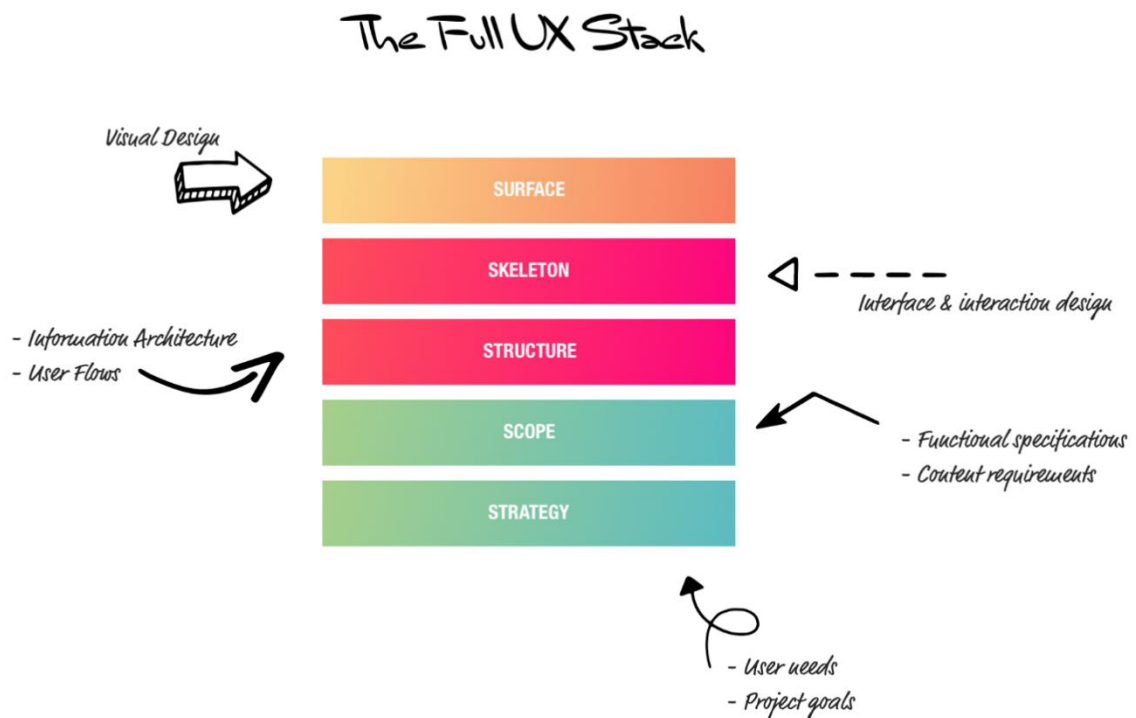
UX Design is studying user behaviour and understanding user motivations with the goal of designing better digital experiences. UX designers do far more than sketch out where a button should appear on a web page. Systems that confuse, intimidate or infuriate their users don't have flawed users, but flawed designs that need to be fixed.



**Fig. 5.1 Balancing business, people and technology**

#### Balancing business requirements

While focusing on user needs it is also important for a UX Designer to be aware of balancing business goals with technology constraints (or opportunities). While it is true that a product cannot succeed without a healthy business, a business cannot succeed without a happy customer — and it is the UX Designer's job to be the customer advocate. Customers don't often get invited to meetings, so don't be afraid to speak up on their behalf.



**Fig. 5.2 The Elements of the User Experience**

**X Designers are part of the full product design process.** *UX Design goes a lot deeper than just the user interface. Testing one user early in the project is better than testing 50 near the end.* UX is a mindset that should be shared by the whole team. It is the team's way of empathising with your users and being inquisitive about what they want. In this way UX isn't a single step in a process but a skill that must be applied at each stage. Let's walk through the full UX Stack to see what questions UX can help the team answer, starting with the initial strategy and scope phases where the concept is taking shape. UXD can answer some fundamental business questions:

1. Do users need the product you are making?
2. Do they want it enough that they will either pay for it or if it is free, spend time looking for it and learning to use it?
3. Are you missing a key feature they will need?
4. Are you spending time building features they will never use?

Next, once we have decided *what* to build we need to decide *how*. It is in the structure and skeleton phases where the project really takes shape and a good UXD can help answer some critical implementation questions:

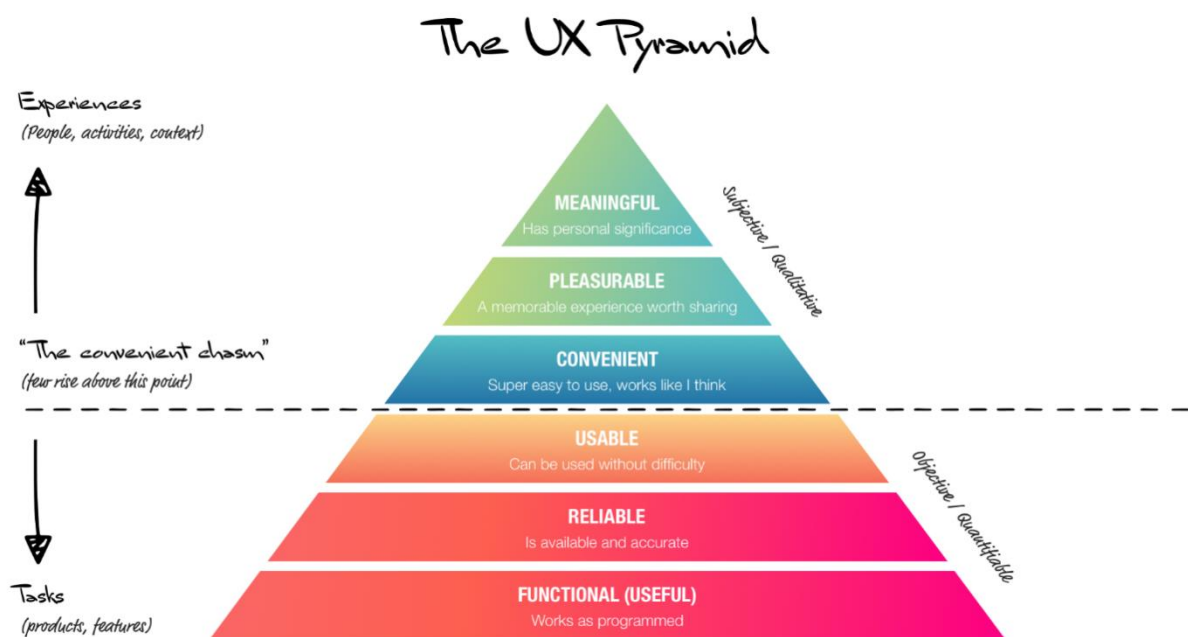
1. How should the content be organised so that users can easily find it?

2. Will users find your app easy to use? Where do they get confused or lost?
3. What content is needed and how should it be written to be most engaging?

Lastly, we need to focus on the surface of the product. What is the product going to look like visually? This is an important step because a user's first impression is critical. UXD can help with the following:

1. What should the visual tone of the product be?
2. How do users feel when they see your product? Do they trust it?
3. Is the product visually appealing and does it spark joy?
4. Is the visual design usable and accessible?

Lastly, always remember that users use products; they don't use documentation. Design artefacts (wireframes, prototypes, strategy documents) are a means to an end, so don't lose sight on improving the product.



**Fig. 5.3 UX Pyramid**

### The UX Pyramid

With such a broad and varied definition, it can be difficult to find ways to benchmark or measure User Experience. The UX Pyramid is an excellent framework for categorising UX effort and tracking progress. Based on Maslow's hierarchy of needs, the base of the UX Pyramid lays the foundation with fundamentals (breathing, in Maslow's case), before advancing to higher, more enriching user experiences. Levels 1 to 3 of the Pyramid concentrate on a user's ability to achieve a desired task. Can they use the system to achieve a beneficial outcome? Levels 4 to 6 go on to focus on the user's

experiences while using the product or service. Do they enjoy using it? Does it make their life better? Many budget-focused businesses only see value in achieving up to level 3, thereby missing out on improved customer loyalty, customer advocacy, customer spend and many other incredibly beneficial outcomes that stem from an engaged customer.

### **Level 1: Functional**

Does it work?

#### **Characteristics:**

- No bugs, errors and outages
- Uses current technologies (doesn't rely on old technologies like Flash that don't work on phones or tablets)
- It has some purpose; someone has a need for it
- Includes all key features
- Works in all modern browsers
- Passes basic accessibility

### **Level 2: Reliable**

*Is it available and accurate?*

#### **Characteristics:**

- Loads in reasonable time, even in peak periods
- Content is current and accurate
- Data is clean and reliable
- Password resets are sent/received promptly
- It can be used effectively on mobile devices and standard device types

### **Level 3: Usable**

*Can it be used without difficulty?*

#### **Characteristics**

- Users don't get lost or confused
- Users can easily find the content or products they are interested in
- The site doesn't rely on constant help messages or long instruction manuals
- It has a short learning curve
- Users don't rely on 'hacks' or workarounds to use it
- Call centres aren't swamped with basic enquiries
- Meets basic UX heuristics and best practice

### **Level 3: Usable**

*Can it be used without difficulty?*

#### **Characteristics**

- Users don't get lost or confused
- Users can easily find the content or products they are interested in
- The site doesn't rely on constant help messages or long instruction manuals
- It has a short learning curve
- Users don't rely on 'hacks' or workarounds to use it
- Call centres aren't swamped with basic enquiries
- Meets basic UX heuristics and best practice

### **Level 5: Pleasurable**

*Is it an enjoyable experience that's worth sharing?*

#### **Characteristics**

- Users invest themselves into it
- Users promote, share and evangelise it

- It becomes part of the user's regular routine

### **Level 6: Meaningful**

*Does it have personal or social significance?*

### **Characteristics**

- Users it brings meaning to their life

### **Diagnosing and solving UX issues**

There are three main strategies for improving the UX of a design or system.

#### **Firstly, by using UX patterns and best practice.**

When working on a project, there is no need to reinvent the wheel each time. There are many standards, templates, checklists and case studies available, providing solutions to many common design problems. Users love consistency, so even though it might be tempting to design something 'cool' and 'new' it is often a better experience to use a commonly accepted way of doing things. It is important to note that this needs to be done mindfully and doesn't mean just blindly copying. Make sure you have explored the rationale behind a particular design choice and if in doubt, user test the design before release.

#### **The second strategy is to conduct a heuristic evaluation.**

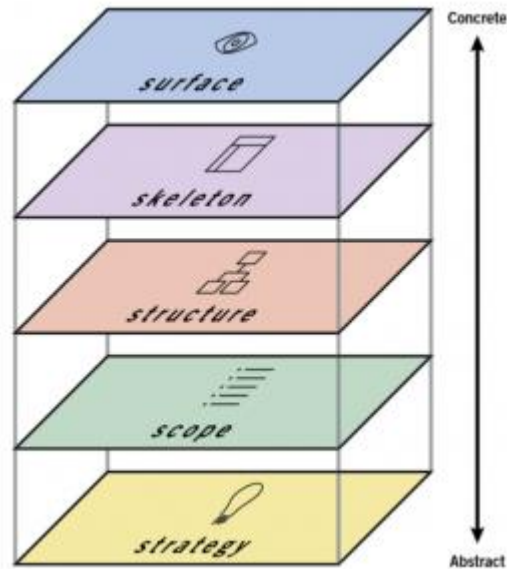
A heuristic evaluation is where you review a design against a set of UX principles. There are many widely accepted lists of UX heuristics you can use that have stood the test of time. A heuristic evaluation is not a total replacement for getting out into the field with users it is a good way of addressing the 'low hanging fruit' (or easy to find usability issues) before spending time and money testing designs out in the field.

#### **Thirdly, there is no substitute for researching and observing your users' behaviour.**

## **2.Elements of UX Design**

**When you want to make or buy something while on a website, you make decisions. Analysing the key elements of UX, you will better understand how these decisions are made. By meeting the needs of users, we motivate them to stay on the website, engage in interaction and be more satisfied.**

There are five key elements of UX. All 5 steps are dependent on each other and the whole process goes from bottom to top.



**Fig. 5.4 Elements**

### **1. Strategy**

The strategy defines the reason why an application or product exists, why you are doing the whole business, who you are doing and why people would use it. The main goal here is to define the needs of users and business goals. This can be done through a strategic research where potential users would be interviewed on one side, and business needs would be adjusted on the other.



**Fig. 5.5 Strategy**

### **2. Volume**

The scope defines the functional and content requirements. What features and content should contain the product. Requirements should meet strategic goals.

Functional requirements – related to the functioning of the entire side, as certain parts work together. These are the characteristics that the user needs to store to achieve a specific goal. Content requirements – the information we need to give value to what we do (text, images, video). Without defining the content, we will not know how much time it takes for the completion of the project.



**Fig. 5.6 volume**

### **3. Structure**

The structure defines the user's interaction with the product, the behaviour of the whole system, how it is organized, and how much it should be displayed at a given moment. There are two structural components: interactive design & information architecture. Interactive design – when functional features are already defined, defines the user's relationship with the product as a system that needs to respond to given user requests.

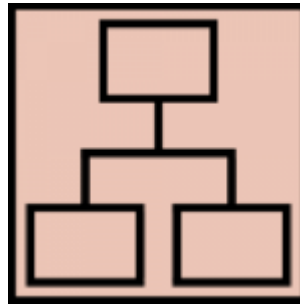
Information architecture (IA) – When requirements are given, defines how the content should be organized and where it is set, in order to facilitate the user's understanding of the product.

#### **Excellent Interactive design:**

- A. Helps users meet their goals;
- B. Has effective interactivity and functionality (what the user can do);
- C. It informs the user about changing conditions while on the page (file has been saved, feedback when the wrong email has been entered, etc.);
- D. Prevents errors when the application requires confirmation from the user for a potentially harmful effect (eg deletion).



- E. Excellent Information Architecture: Organizes content by categories and priority information based on user needs and business goals; Making easily understandable content and switching from one to another content is simple; It is suitable for your audience; It is flexible for adaptation and adapts to changes.



**Fig. 5.7 Structure**

#### **4. Skeleton**

The skeleton determines the visual form of the screen and presents all the elements that need to be interacted. Shows how the user moves through information and how the information is presented to be clear, effective, and obvious. Wireframes are widely used to create a visual format. It is actually a static diagram that presents the visual format of the product, which includes content, navigation, and all forms of interaction. The skeleton is divided into three parts: Design of the interface, design of navigation and information design.

Interface design – presenting and editing elements so they can act with the functionality of the application; Navigation Design – the way of navigating through the information; Information Design – presentation of the information in an easily understandable way.

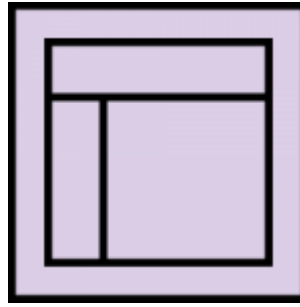
The skeleton should answer the following questions:

What visual forms should be presented on the screen?

How can interactions be presented and divided?

How can a user move around the site or application?

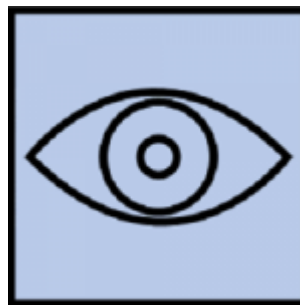
How can the content be clearly presented?



**Figure: 5.8 Skelton**

## **5. Area**

The last step. It refers to how your product, typography, colours, the actual layout, and so on will look. The goal is to simplify things, be easy to understand and the user to absorb the necessary information. It is necessary to visually present the entire content and buttons, for the user to know what can do and how to communicate with them.



**Fig. 5.9 Area**

Everything is connected. If you do not have a clear strategy, you pay the price over the entire project. An ordinary decision can completely change the course of the project. It's okay to make your decisions change from time to time and of course you can "punch". It's not good to make fixed decisions, because in the end it can turn out to be a product that no one wants. Changes are inevitable. The root of every successful design is a strategy, which later becomes the scope where the goals of the user and the business are set. Below is the structure where the interactions and the distribution of information are defined. Through wireframes "scanners are made" to present the interactions and structure defined in the structure. The skeleton allows the information to be clearly presented. Finally, on the surface, all the decisions made up to then are considered and a final visual presentation is made. There are many research activities that allow you to understand your users and how they interact with your sites, apps and services.

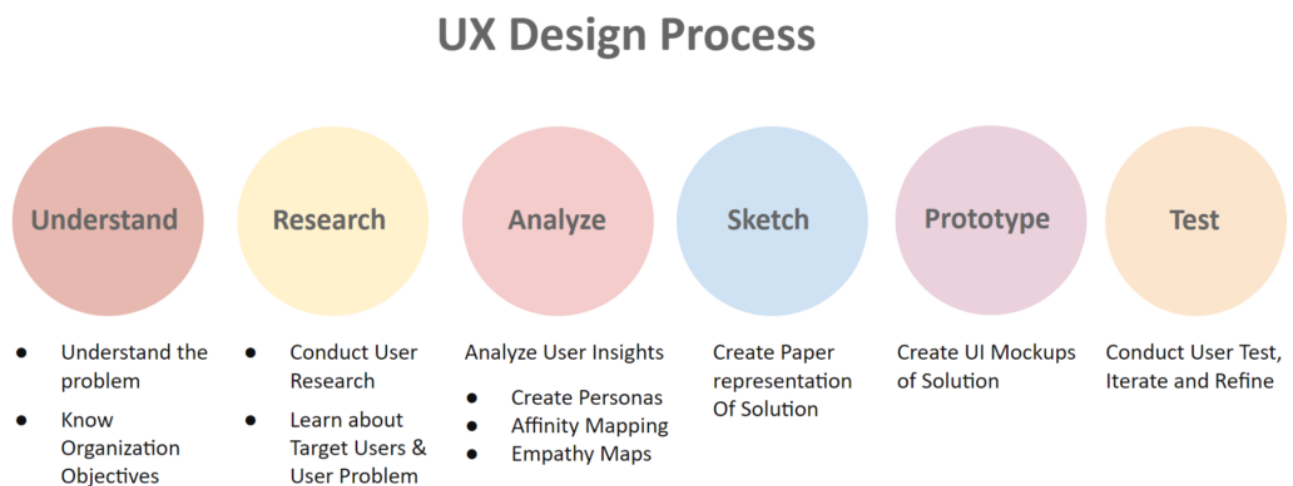
### 3.UX Design Process

**User experience (UX)** refers to the interaction a user has with a product or service and the experience they have with it. It is in fact a person's feelings, attitude and emotions they feel while using any product or service.

User experience (UX) design is an Empathy driven process of designing a product or service that are useful, easy to use, and delightful to interact with. **UX Design process is a process of solving User Problem. It is an iterative method which helps a UX Designer to continuously improve and polish designs based on User Feedback.**

It is important that you know that a UX Design process feeds on User Feedback at every step. Without User Feedback you would be shooting in the dark. UX Design Process makes it possible for designers to craft amazing experiences for users. Also, UX process followed by one Designer can be different from another .

While the majority may maintain the same steps however, It usually depends on the resources the organization has and also sometimes depend on the project.



**Fig. 5.10 Design process**

#### 6Steps of UX Design Process

##### **Step 1: Understand**

UX design is the process of solving a problem for user so that they can achieve their goals easily. In order to do this, the first step is to **understand the problem** you would solve and **the objectives of the organization** as well.



**Fig. 5.11 understand**

There can be multiple ways to do this

- If you working for an agency then ask your clients
- Working for an organization then ask the stakeholders
- Ask for previous research conducted which can include market research, User research, competitor analysis, etc.)
- Speak to the Product Managers as well
- Analyse requirements to understand and clarify them

Getting understanding about two elements is crucial

1. User
2. Brand

Understanding these Key elements would help you create a Design Strategy that would align **the goals and objectives of User and business**. And forms a base for your next step where you would conduct user research to dig deep into the User Problem.

**Outcome: -**

**By the end of Step 1, you would get a good understanding of the Design Strategy and objectives of the Organization. This would guide you on how-to carry-on User Research.**

### **Step 2: Research**

User Experience (UX) research—serves many purposes throughout the design process. It not just helps us to get a clear picture of about users, but also answers key questions like **what users think and why they do what they do**.



**Fig. 5.12 Research**

And in order to do so you need to ‘walk their shoes’. You need to learn about the Target Audience Hence, it is extremely essential that user experience research and Design teams conduct user research regularly. Additionally, it also helps us identify and prove or disprove our assumptions. In Bigger organization, research is conducted by a UX Researcher. However, in smaller organizations/start-ups, a UX Designer has to wear multiple hats to perform multiple jobs. Primarily, there are two UX Research methods

### Qualitative Research

Exploratory form of the research where the researcher collects verbal, behavioural or observational data which is interpreted to get insights. Most common methods are

#### **1. Focus Groups**

Focus Group brings together 6-9 Participant users. The Goal of the Test is to discover what users want from the Product.

Furthermore, conducting Focus Groups allows you to learn about their attitude, opinion and reactions to concepts that you are testing with Users.

#### **2. Contextual Interview**

A contextual interview involves one-on-one interaction between user and researcher. And the interaction involves the researcher to watch and observe the user work in their environment; and then discuss those activities with them.

#### **3. User Interview**

User interview is one of the most common User research methods. In fact, it provides you the rich information and insights of what your target users think about your new product, site or service.

A User Interview is typically conducted by 2 UX researchers, one to conduct the interview and other to record the interview and take notes.

#### **4. Ethnography Study**

Ethnography is a kind of social research. It is type of qualitative research which provides a detailed and in-depth description of everyday life and practice taking a wider picture of culture.

### Quantitative Research

Structured way of collecting and analysing data in numeric form. Analysis, interpretation and presentation of numerical data is done by using Statistical techniques.

#### **Survey**

Surveys consists of a set of questions to gather wide information on a wide range of topics. It is one of the most common types of quantitative research methods. Survey is an easy way of collecting quantitative data from a large number of users within lesser amount of time.

The questionnaire, or survey is completed by the person being surveyed which may be

- an online questionnaire
- a face-to-face interview
- or a telephone interview.

**Outcome: -**

**By the end of Step 2, you would get lot of User insights. This information needs to be then analysed in order to make a sense out of it and learn about key User issues.**

### **Step 3:Analyse**

After you have conducted your Research, you would a plethora of insights which can be quantitative or qualitative. In the next step you have to analyse the information gathered and make connections around it so that you draw some conclusions.

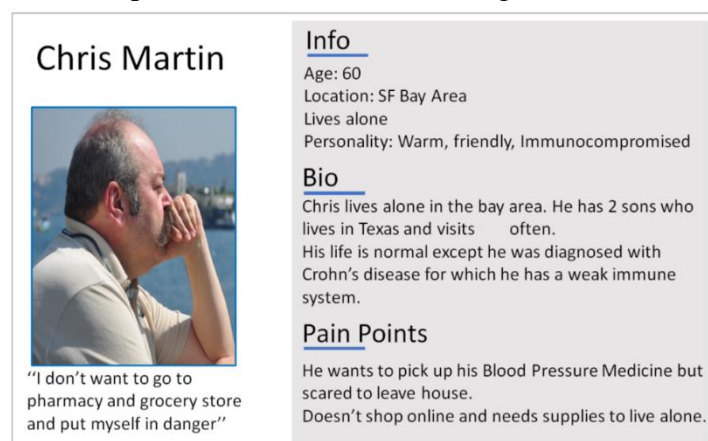
Framing the right problem is the only way to create the right solution. Moreover, only after you have detailed information of the wants, needs, and pain-points, you can synthesize the information into an actionable problem statement.

**“If I had an hour to solve a problem, I’d spend 55 minutes thinking about the problem and 5 minutes thinking about solutions.”– Albert Einstein**

There are multiple UX Techniques that you can use to Analyse the information

### **1. Creating User Personas**

User Persona is a **fictional yet realistic representation of Target user of the product**. Creating User Personas helps you to identify what the user requirements by understanding their needs, experiences, behaviours and goals.



**Fig. 5.13 Creating user personas**

### **2. Affinity Mapping**

Affinity Mapping is about finding the user needs from the observations gathered. The goal is to synthesize information gathered into common themes and patterns to discover interesting findings which will help in defining user focused Problem and creating design solution.



**Fig. 5.14 Affinity Mapping**

### 3. Empathy Mapping

An Empathy Map is another method to synthesize the observations to uncover unexpected insights around user needs. Moreover, it can be drawn on a board, paper or table and has four quadrants representing the four key user traits.

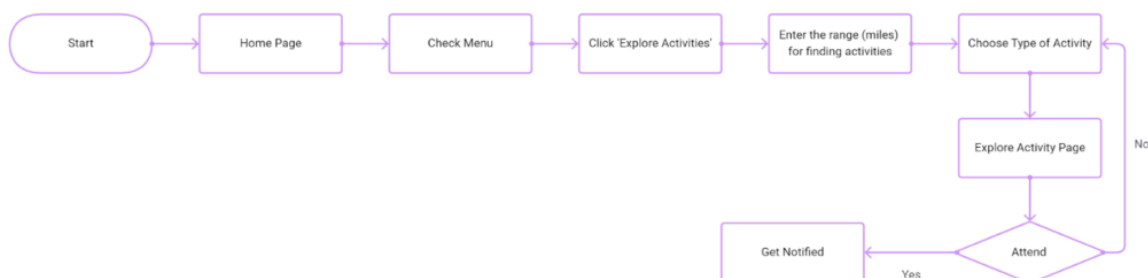
By Analysing and Synthesizing the information, you would be able to do. Defining your Point-of-view statement which would Define the Right Problem or simply define the Challenge to address.

Based on the information we go ahead and depict paths a user would take to solve the identified problem. Creating these paths help us understand what user will be going through when using product or service. Infact, only by identifying and depicting such path then you build the best product for them.

Some of the ways you can do it are

### 4. User Flows

Creating User flow is creating Visual representation of specific routes that a user might take while navigating a website or app in order to achieve a goal. The route starts at an entry point and then covers all the steps the user must take to reach a certain outcome.



**Fig. 5.15 User flows**

## 5. User Journey Map

User Journey map talks about the complete path a User takes while interacting with your company which starts from the awareness stage when they realize they have a need, through all the points of interaction with your brand, and the moment they are satisfied (or not).

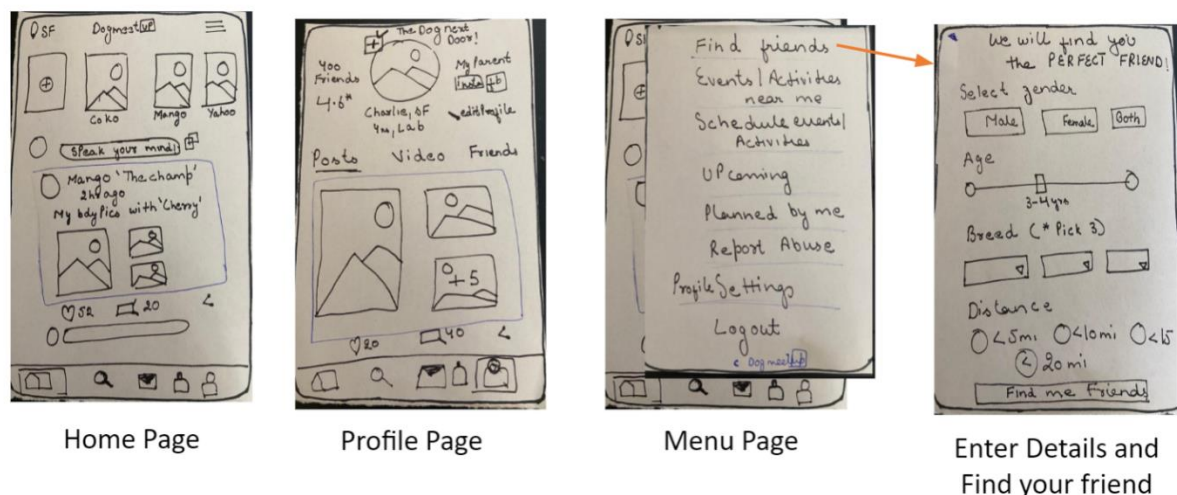
**Outcome: -**

**By the end of Step 3, you would get a clear understanding of User Issues that would help you to define your Problem Statement. All of this would help you to brainstorm and come up with solutions to solve the User Problem**

### Step 4: Sketch Designs

Now, it's time to actually define how the content on each page should be organized. Furthermore, you have to define how these pages would work together in a way that for user finds that they find the design it intuitive and easy to navigate.

Easiest way to do this is by creating Paper Sketches. Infact creating paper sketches and Prototyping on paper is a quick and cost-effective way to test ideas in the early stages of Product development.



**Fig. 5.16 User journey map**

## UX Case Study | Yukti.io

It is very common for designers to move ahead and create Actual UI screens, however it is the best practice to stick the lo-fi first. See, you must spend time on **exploring the designs and not creating the designs**. And Paper Prototypes lets you do just that.

Creating a Digital App or showing multiple steps to use a physical product- Draw different sketches of the user interface on different piece of paper and simulate interactivity by linking and moving series of paper on table. This is quick and easy, best in the early stages of Product Development. Make sure you review the design with team members and then move ahead to design it.

**Tools Used – Pen and Paper**



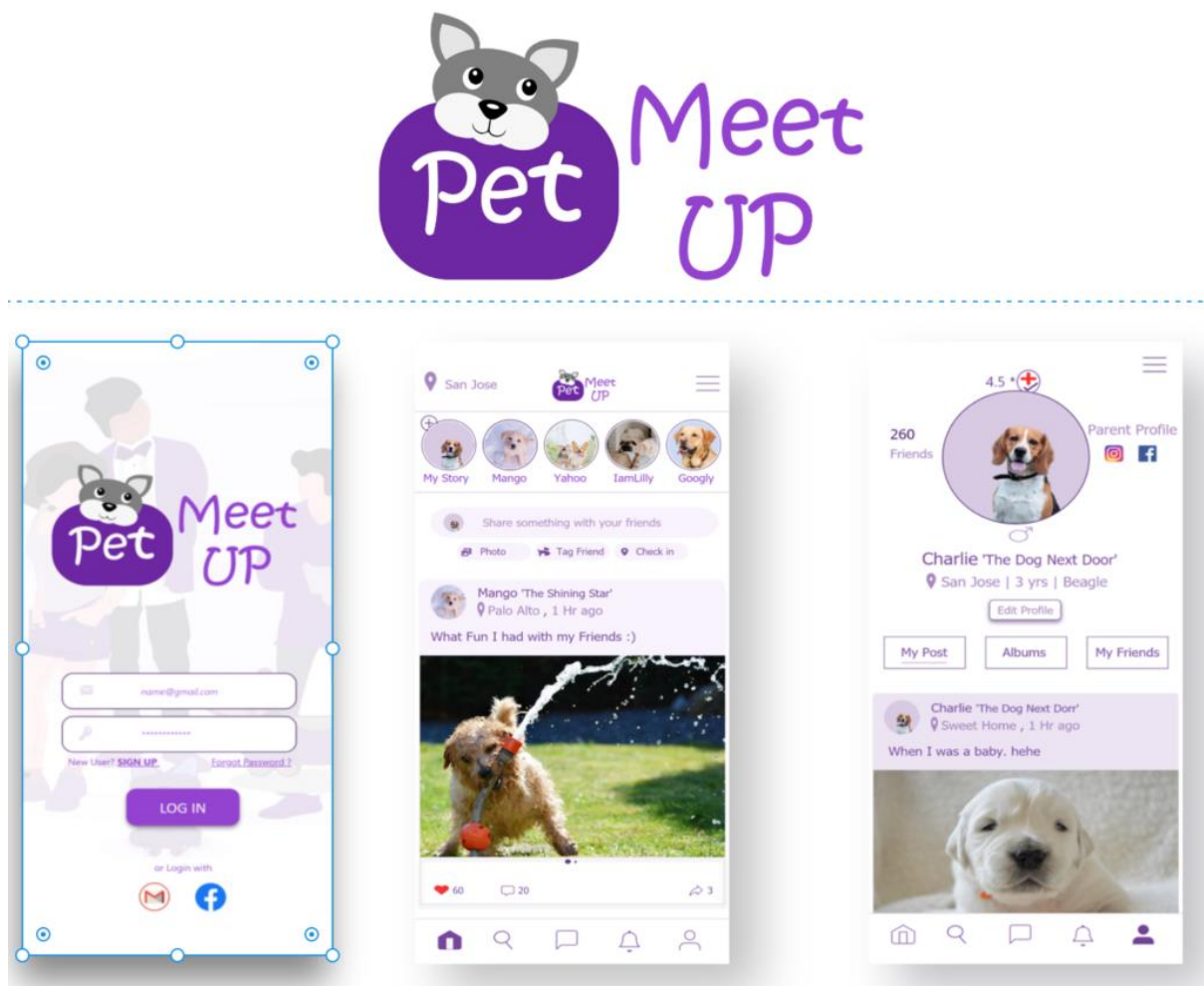
**Outcome: -**

**By the end of Step 4, you would have a Paper representation of your solutions, validated by your Team members and stakeholders. The solution would be then designed in the next step.**

### **Step 5: Wireframes and Prototype**

A wireframe is a static representation or blueprint of the initial product concept however, prototype is a working model of an app or a webpage. Wireframes and prototypes can be low/medium as well as high fidelity.

Wireframing is the stage where you take a concept or design and shape it into something tangible so that you can thoroughly review your work with users and stakeholders and ensure it makes sense.



**Fig. 5.17 UI Mock-ups**

Creating Wireframes gives Designers the flexibility to play around and do lot of Experimentation. Prototyping helps to review and refine it with the help of User Feedback to turn it into a Polished version that can be Developed to an End Product.

This is often when UI designer comes into the picture to design the high-fidelity mock-ups of the product. It is important to mention here that though UX and UI design needs different skill sets, however Sometimes UX and UI are done by the same person.

**Tools Used-** There are a lot of tools in the market. some are free and for some you have to pay. Check out this article where you would find 8 important UI tools that you can use to create Designs.

**Outcome: -**

**By the end of Step 5, you would get Mock-ups of your solution to tested with Target Users.**

### **Step 6: Test Design and Iterate**

The next step is to **go out and test the Design with end users** to gather feedback on it. This feedback will form the basis for further iterations and refinement. You must learn about how they feel and think about it. Learn how they interact with the prototype. Pay attention on the User interactions.

So, Test the Design, get a constructive feedback and iterate it. Get as much critical feedback you can as it will help you to move faster in the Design Process. Additionally, it saves time, effort and money by catching bugs errors usability issues, that you might not have anticipated.



**Fig. 5.18 Testing**

You'll get to learn if your solution has been validated or if it has to be improved. It might be possible that end user will invalidate the solution and you will have to redefine the problem by empathizing with user. And repeat the entire process again.

### **Some of the ways of conducting User Test are**

#### **1. Usability Testing**

Usability testing evaluates the degree to which the system can be used by specified users with effectiveness, efficiency and satisfaction in a specified context of use- ISO 9241 Ergonomics of human-system interaction

It is a great method which allows you to improve the Usability of the Product. **Usability** refers to the ease with which a user can use a product in order to achieve his/her goal and how useful the Product is. A product which is high on usability makes it easy for user to accomplish his/her goals.

#### **2. Concept testing**

When you have a big idea, it becomes important to check and evaluate whether or not it would be accepted by people when launched in the market. Concept testing is done to **evaluate consumer acceptance** for a new product idea.

Concept testing helps to **validate as well as refine a product concept** by getting feedback directly from target market.

### 3. First Click Testing

First-click testing is the best way to improve your App/website designs. It allows you to analyse where the User clicks on the screen when the website or app is shown to them.

### 4. Tree Testing

Tree testing is a **usability technique** for evaluating the ease with which information can be found in a website. However, unlike Usability testing, Tree Testing is done on the simple version of site structure.

### 5. Beta Testing

In Beta testing, you test a near-complete product/software/application with end users (called **Beta Testers**). Before Launching the product in the market, enough tests need to be carried out to test the functionalities and reporting bugs.

Outcome: -

By the end of Step 6, you would get a lot of User Inputs whether the solution solved user problem and uncover Usability Issues. All of this information would be then used to iterate the solution further.

UX Design process is both. It is iterative in nature and can be entirely unique to your business and product. What it means is that the stages are going to look a lot different for you as compared to somebody else. However, the goal remains the same which is to solve User problems in the most effective and efficient manner. What is extremely important is to comprehend the psychology of a user and using best UX practices as described in the article. From Team members to stakeholders to Product managers to developers, everybody contributes in the UX Design process by performing their tasks and duties. Hence, as a UX Designer one of the most important skill that you must have is collaboration.

Furthermore, Strong communication and presentation skills; and to be able to articulate and discuss your design decisions with team is important as well.

## 4. Research Methods and Tools

Gathering user data and feedback is a hectic and maddening process, especially if you're in the middle of pushing out new builds. But UX research methods serve as a way of streamlining the task a bit and making the data you get more readable.

### *Interviews*

Is sitting down with users and asking them extensive questions an old-fashioned and time-consuming method to get information? Yes, it surely is. But is it also highly effective at getting answers that the user wouldn't provide to an automated test-box? You know it is or I wouldn't be mentioning these as a viable method.

It is important to mention that getting solid information requires the skills of a trained interviewer. I'm no good at these, for example, but some of my colleagues are a godsend when it comes to handling in code Group's users. Without interviews, we wouldn't know about some small issues with our early designs because users often tend to omit things when filling out automated response forms. Some do

it out of laziness, some don't want to harp on the problems too much, and some are just plain bored by the unending survey form.

### ***Automated***

### ***Surveys***

Oh, uhm, so about that time when I said survey forms are unending and not always as effective... They are, sure, but they can also be a great way of getting info on things that an interviewer simply wouldn't think to ask. If a user is willing to engage with a lengthy survey, chances are you'll get some excellent data out of them. And even a short one can target the particular questions that are rote and usually omitted in interviews. It might seem like an extra step (and, technically, sure, it is) but you never know which question will bring you that 9gold nugget of info that you need for a breakthrough in your design.

Now, it's still pretty time-consuming to compile a survey but it's pretty much a one-and-done task. A good survey will get all the info you need and will only require small tweaks as the projects change. Besides, you don't necessarily have to do it all by hand... But more on that in the tools section.

### ***Card***

### ***Sorting***

This is a fun one, though it's mostly a tool for the early stages of your design. It's a technique that hinges on most people having the same idea of "what goes where". Basically, we all "know" that the Contact page should be placed at the end of your menu, not at the beginning. There's no commandment that forces us to do so, we just feel kind of icky when we see that page at the beginning of a menu. This trained knowledge of placement is what the card testing relies on.

You present the user with several cards and ask them to arrange those in the "correct" or "optimal order". Odds are, with a large enough base, you'll see some differences in the arrangement. However, most will have similar setups and that's how you glean information about placement. Ask users where they'd want the most important elements to be, see how they "rank" them etc.

### ***User***

### ***Testing***

There are two approaches to user testing — moderated and unmoderated. In one case you'll have the users in a controlled environment and will have a moderator who helps the users if necessary as well as collects data. It's good if your testers are new to the process or if you want them to have a more intimate understanding of your design.

As for unmoderated user testing, it's good for users that might feel more comfortable using the product without any oversight. It's not going to be optimal if your project is complicated and the user can't get the hang of it, of course. However, if you're just trying to do design research for a website, unmoderated might be the way to go.

There's also the third approach, which makes traditionalists mad and works wonders if your budget for research is lower than the temperatures in the Antarctic. Guerrilla testing will have you running into coffee shops and asking people to test your site or app, be it for free or for a cup of coffee. On the one hand, this will give a clear opinion from your average bystander. On the other hand, that average bystander won't know much about industry-specific projects so don't expect a random person in the

park to solve the UX issues in your accounting software. Still, if the budget isn't giving you much to work with, UX guerrilla testing is a viable option.

### ***Screen***

Since users often don't consciously track their actions on a page, it can be hard to pinpoint which elements they struggled with or what didn't work as intended. By recording their interactions with the website, you can see the faults for yourself. From time spent navigating to the length of a visit, this data can help you work out some metrics that might not be available in the purest form otherwise. It also protects you from forgetful users who might misremember their interactions.

### ***Recording***

### **Tools to Use**

Now, these methods might be a great way to spend your research budget and, with enough time, get your UX researcher to bring you perfection. But you don't have to do handle these manually or beg for a higher budget if you can't afford skilled moderators or a UX researcher. With an array of tools such as these at your disposal, you can be your own researcher.

### ***Crazy***

With the Crazy Egg service, you can record every single thing a visitor does on the website, evaluating their actions and seeing how well your sitemap works. By tracking random visitors to the site, you can tweak things and try out different configurations with no bias that comes with moderated testing.

### ***Egg***

### ***Wufoo***

Remember when I said that you don't have to do surveys by hand? Well, what else is there to say except — Wufoo! This service allows you to create surveys really quickly, though it's, perhaps, not serving up the most appealing design (ironic, I know). If you'd prefer a second option, Type Form is a good choice as well.

### ***User***

If you want to run some tests but don't have any users on hand, the User Testing surface lets you run the test with an online userbase and get the results in a matter of minutes. Similar to it is the Verify App and User Zoom. Ethnic is also an invaluable platform for getting the users on board.

### ***Testing***

### ***UXPunk***

Remember card sorting? It's that thing I mentioned a long time ago, almost five paragraphs up, I'd assume. Well, that one is available online as well. Just use UXPunk to start off!

## **5. Understanding User Needs and Goals:**

How to recognise user needs Traditionally in government, user needs are identified when policy is written. Services are designed based on this policy. This results in a large gap between the understanding of user needs for the design of policy, and the understanding of those needs for the design of services.

Our way of working asks government to look at our services from the users' perspective.

For example, people experience several touchpoints with different areas of government when they travel overseas. They may need to get a passport, check travel warnings or register their travel details. All these add up to an entire service from the user's perspective, despite being accessed from multiple different government agencies.

When doing user research, we recognise user needs by focusing on people's goals rather than their preferences. Goals are the things that they need to do. Preferences are things they may want or like, such as website style or colour.

A common misconception is that people are coming to interact with a website, when in fact they are often coming to access a service.

People interact with services, not websites. They use our digital services when those services are simple and fast.

### **Design to meet user needs**

Services designed around user needs:

- are more likely to allow the user to complete the task they are trying to do, without support
- help more people get the right outcome, achieving government's policy intent
- cost less to operate by reducing time and money spent on resolving problems

The current experience of a service might involve interacting with different products that may be owned by different parts of government. The service you are designing and building needs to meet the whole user experience with those products.

This makes the whole experience clearer and simpler for the user. Often you won't meet their needs if you work on a single product in isolation.

Think about the whole service, not just a product.

### **How to learn about user needs**

You can learn about users and their needs by doing the following types of activities:

- interviewing and observing actual or likely users of the product
- talking to people who work with actual or likely users
- reviewing existing evidence

Treat any opinions or suggestions that don't come from users as assumptions. Explore these in your research.

Keep researching through each stage of the design and delivery process to make sure your service continues to meet user needs.

### **Who to include in research**

Groups to consider including in user research are people who:

- currently use the service
- don't currently use the service but may need it in the future
- have problems using the service
- work in the service (for example, call centre staff)
- help others use the service (for example, caseworkers, legal professionals or charity workers)

When researching, focus on users who have problems using the existing service and its products. This will help you create a simpler, clearer and faster service that more people can use.

### **Research wide then go narrow**

For government services, it's best to start wide and then narrow down your user research.

Begin by looking at as many different users of your service as possible. That way you'll see the entire spectrum of users, from the mainstream to the edges. You should aim to speak with as many of your user groups as possible in the Discovery stage.

The mainstream users will show you what business as usual is for your service. You'll see why they are using products and how they are managing the processes.

At the extremes, you'll see the groups of people who may be finding it difficult to access your service. You'll find out their current pain points as well as what is working well about the service. These pain points might include things that are pushing them out of your service. For example, they may find the service too difficult to navigate.

Other users might have workarounds for accessing your service. For example, they might use websites to translate content into their preferred language.

Build the service so that it works for the people with the most barriers to accessing it. That way, you can be sure you're building it for everyone.

Define your participant criteria

First, brainstorm the different groups of people you need to include in your research by using information from:

- existing research
- subject experts
- front line staff
- service data
- analytics
- general population statistics
- user groups that may have different experiences when using your product
- user personas (if available)

When you've defined your overall criteria, decide which groups you will include in each round of research.

## Specify target groups

Depending on your research objectives, your criteria might be:

- a particular demographic (for example, women under 30 years of age)
- a specific user group (for example, small business owners or job centre staff)
- specific life events (for example, users who have recently moved home or are looking for a job)
- specific access needs (for example, people who use assistive technology)
- a specific level of digital skill (for example, users who have basic online skills)

Ask subject experts for information about target groups. They may know about groups that you haven't already included. They may also help you to get in touch with people who need extra support to take part in your research.

Review your participant criteria to make sure they are relevant to your research questions. Do a gap analysis to make sure you don't miss important groups.

Outside of any specific criteria, always try to recruit a representative spread of:

- age
- gender
- social and economic status
- cultural and linguistic background
- education level

The research methods you use will determine the number of participants that you need. For example, you'll need:

- 4–8 participants per round of user research
- more than 250 participants for benchmarking

## **Make your research inclusive**

You need to learn about all your users to build a good service. Find out about people with access needs or who don't currently use digital services.

Make sure you don't exclude any users in the way you do research. Recruit participants that reflect the population and choose accessible research locations.

### Recruiting participants with access needs

You must conduct research to understand the experience of people with disability. Make sure you find out their needs ahead of time so you can make sure they can take part. For example, they might need to:

- bring someone with them
- use assistive technology
- bring a service dog



Caption: A research dashboard showing which user groups research has been conducted with.

## **Researching with people who don't use digital services**

### Who to include in your research

You should conduct research with people who have a mix of digital skills. It is important to speak with people who have a low level of skill as they will help you understand support needs that are the most difficult to meet. For example, some users may not be able to leave their homes, may not have a computer or may live in an area with very poor internet. It is also important to conduct research with people who have a high level of skills. They may still need support with an online service or may have concerns about privacy and security.

When you're researching with users who don't or can't use digital services you should focus on the people who are current users of your service or who are likely to use it in future.

Include people in your research who get support to use digital services. For example, they may get help from carers or support workers.

### Things to remember when doing research

When carrying out your research, remember to:

- explore users' digital support needs across all channels (including online and offline)
- hold the research in places where users use the service (for example, if they need to go to the post office to get support, do it there)
- talk to users rather than relying on the opinions of third parties
- don't rely on self-assessment of digital skills — wherever possible, get them assessed by an expert user researcher
- show the on-screen service to all users so they can give you accurate feedback on the support they would need
- find out the places where users seek support, including government contact centres, libraries, friends and family, trade bodies, paid intermediaries, charities or colleagues

## **Share what you know about users**

Once you've learned about the different kinds of users of your service, present your findings. Do it in a way that's easy for others to understand and share.

You can present your findings by creating:

- experience maps that show how users interact with existing services
- user profiles that describe groups of users with similar behaviours and needs
- case studies that explore individual experiences

## Writing user needs

Once you have a good understanding of your users' needs, write them down. You can then add them to your descriptions of users.

User needs are usually written in the format:

- I need/want/expect to ... (what does the user want to do?)
- so that ... (why does the user want to do this?)

You can add:

- as a/an ... (which type of user has this need?)
- when ... (what triggers the user's need?)
- because ... (is the user limited by any circumstances?)

Write user needs from the user's perspective. Use words that people would recognise and use themselves. For example:

'As an Australian I need a passport so that I can go overseas and prove who I am.'

Prioritise your user needs by focusing on what's most important for your users so you don't create an unmanageable list of user needs.

## Validating user needs

User needs should:

- be described in the user's words
- be based on evidence from user research, not assumptions
- focus on the user's problem rather than possible solutions

As you progress through the service design and delivery stages, confirm and refine user needs.

## Show the user needs

You should share your users' needs with everyone in your service including colleagues and stakeholders.

When presenting user needs, it can be helpful to group them by:

- audience, user type or persona (for example, new parent, caseworker or small business)
- stage in a process (for example, register, apply or interview)
- function (for example, identity, payments or licensing)

The more you share, the more people will learn about your users and what they need from your service. They'll also ask questions, spot gaps and comment on what you're doing. All of these will help you iterate your user needs and ultimately design a better service.

User research dashboard

‘We have a user research dashboard because it’s a big priority for our team. To make sure everyone is out there doing research regularly. It helps us make sure we are focusing on meeting user needs.’  
— Developer.

## 6. UX DESIGN PROCESS

We apply design thinking to product design, we would follow a UX process with the following five key phases:

- Product definition
- Research
- Analysis
- Design
- Validation



**Fig. 5.19 UX Design Process**

The UX design process consists of five key phases: product definition, research, analysis, design, and validation. Image credit Nick Babich.

### 1. Product definition

One of the most important phases in UX design is actually done before the product team creates anything. Before you can build a product, you need to understand its *context for existence*. The product definition phase sets the foundation for the final product. During this phase, UX designers brainstorm around the product at the highest level (basically, the concept of the product) with stakeholders.

This phase usually includes:

- **Stakeholder interviews:** interviewing key stakeholders to gather insights about business goals.
- **Value proposition mapping:** thinking about the key aspects and value propositions of the product: what it is, who will use it, and why they will use it. Value propositions help the team and stakeholders create consensus around what the product will be and how to match user and business needs.
- **Concept sketching:** creating an early mockup of the future product (can be low-fidelity paper sketches of the product's architecture).

This phase typically ends up with a project kick-off meeting. The kick-off meeting brings all the key players together to set proper expectations both for the product team and stakeholders. It covers the high-level outline of the product purpose, team structure (who will design and develop the product), communication channels (how they will work together), and what stakeholders' expectations are (such as KPIs and how to measure the success of the product).

## 2. Product research

Once you've defined your idea, the product team moves to the research phase. This phase typically includes both [user research](#) and market research. Seasoned product designers think of research as a good investment—good research informs design decisions and investing in research early in the process can save a lot of time and money down the road.

The product research phase is probably the most variable between projects—it depends on the complexity of the product, timing, available resources, and many other factors. This phase can include:

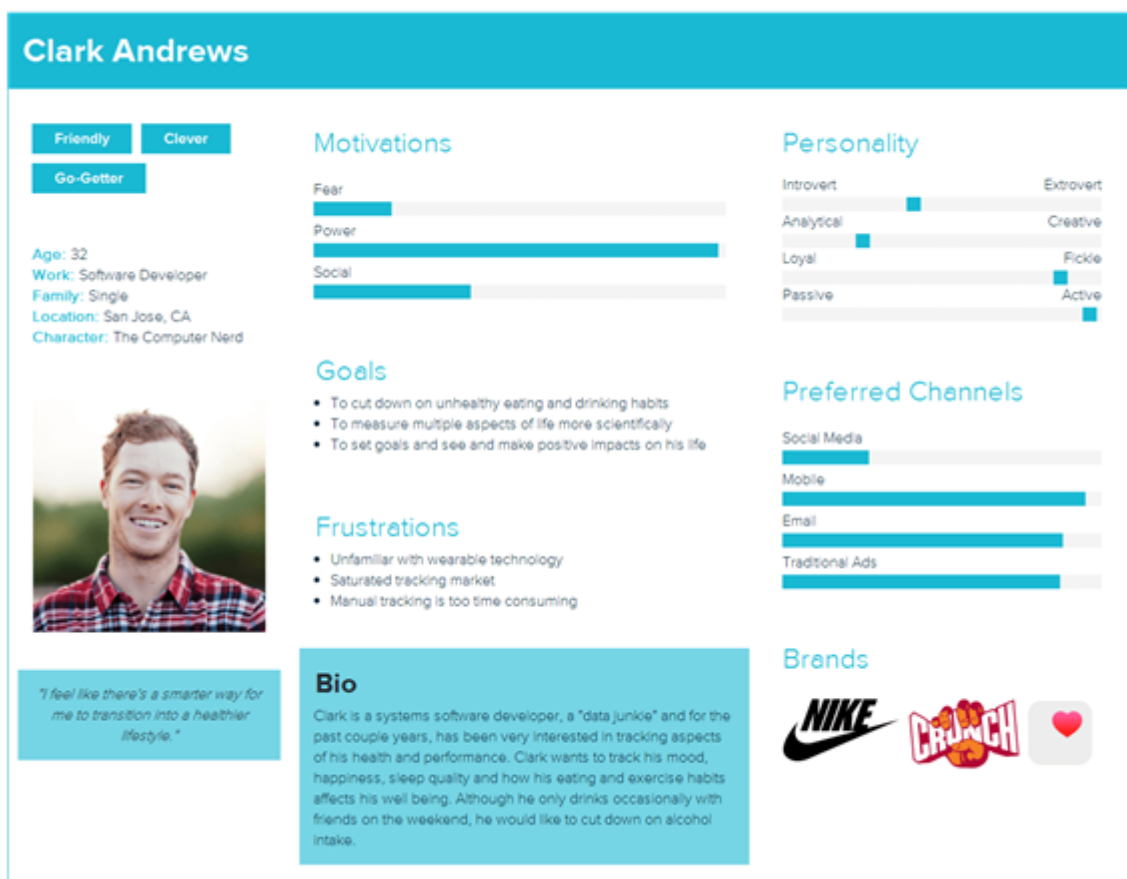
- **Individual in-depth interviews (IDI).** A great product experience starts with a good understanding of the users. In-depth interviews provide qualitative data about the target audience, such as their needs, wants, fears, motivations, and behavior.
- **Competitive research.** Research helps UX designers understand industry standards and identify opportunities for the product within its particular niche.

### 3. Analysis

The aim of the analysis phase is to draw insights from data collected during the research phase, moving from “what” users want/think/need to “why” they want/think/need it. During this phase, designers confirm that the team’s most important assumptions are correct.

This phase of the UX process usually includes:

- **Creating user personas.** Personas are fictional characters that represent the different user types for your product. As you design your product, you can reference these personas as realistic representations of your target audience.



**Fig. 5.20 Example of user persona**

Example of a user persona, showcasing the person’s gender, age, motivations, and more. Image credit [Xtensio](#).

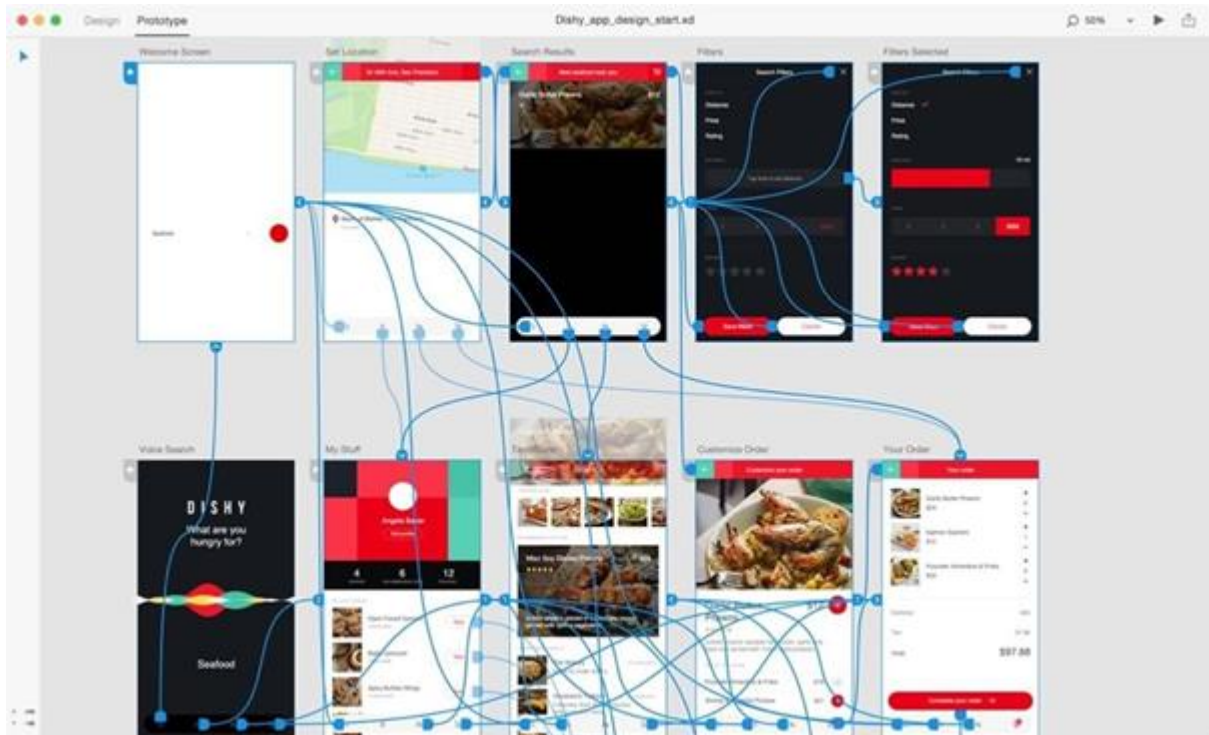
- **Creating user stories.** A user story is a tool that helps designers understand the product/service interactions from the user's point of view. It's usually defined with the following structure: "*As a [user] I want to [goal to achieve] so that [motivation].*"
- **Storyboarding.** Storyboarding is a tool that helps designers *connect* user personas and user stories. As the name suggests, it's essentially a story about a user interacting with your product.

#### 4. Design

When users' wants, needs, and expectations from a product are clear, product designers move to the design phase. At this step, product teams work on various activities, from creating information architecture ([IA](#)) to the actual [UI design](#). An effective design phase is both highly collaborative (it requires active participation from all team players involved in product design) and iterative (meaning that it cycles back upon itself to validate ideas).

The design phase usually includes:

- **Sketching.** Sketching is the easiest and fastest way to visualize our ideas. You can do this by drawing by hand on a piece of paper, on a whiteboard, or in a digital tool. It's very useful during brainstorming sessions because it can help the team visualize a broad range of design solutions before deciding which one to go with.
- **Creating wireframes.** A wireframe is a tool that helps designers visualize the basic structure of a future page, including the key elements and how they fit together. [Wireframing](#) acts as the backbone of the product, and designers often use them as a foundation for mockups and prototypes.
- **Creating prototypes.** While wireframes are mostly about structure and visual hierarchy (the look), prototypes are about the actual interaction experience (the look and feel). A prototype is like a simulation of the product and may be low-fidelity (clickable wireframes) to high-fidelity (coded prototypes).



**Fig. 5.21 Adobe XD**

With Adobe XD, you can turn static mockups into a prototype by creating a connection between individual screens. Image credit [Adobe XD](#).

- **Creating a design specification.** Design specifications contain all of the visual design assets required for developers to turn prototypes into a working product.
- **Creating design systems.** For large projects, designers typically create a system of components, patterns, and styles that help both designers and developers stay on the same page regarding the design.

## 5. Validation (Testing)

Validation is an essential step in the design process because it helps teams understand whether their design works for their users. Usually, the validation phase starts after the high-fidelity design is ready, since testing with high-fidelity designs provides more valuable feedback from end-users). During a series of [user testing](#) sessions, the team validates the product with both stakeholders and end-users.

The validation phase of the UX process may include the following activities:

- **“Eat your own dogfood.”** Once the design team has iterated the product to the point where it’s usable, it’s time to test the product in-house. Team members should try using the product on a regular basis, completing routine operations to uncover any major usability flaws.
- **Testing sessions.** User testing sessions with people who represent your target audience are very important. There are many different formats to try, including moderated/unmoderated usability testing, focus groups, beta testing, and A/B testing.
- **Surveys.** Surveys are a great tool for capturing both quantitative and qualitative information from real-world users. UX designers can add open-ended questions like “What part of the product you dislike?” to get user opinions on specific features.
- **Analytics.** Quantitative data (clicks, navigation time, search queries, etc.) from an analytics tool can be very helpful to uncover how users interact with your product.

### **How to improve the UX design process**

Now that you’ve seen how each phase connects, let’s consider some helpful tips for improving the UX design process:

Embrace the iterative nature of the design process

UX design isn’t a linear process; it’s an iterative process. The phases of the UX process have considerable overlap and usually there is a lot of back-and-forth. Take research and design as an example: as the UX designer learns more about the problem and the users, he or she might want to rethink some design decisions. It’s important to accept the fact that your design will never be perfect, so take your time to research the needs of your users and make your product a bit better for them.

## **7. VISUAL DESIGN PRINCIPLES**

**Visual-design principles** inform us how design elements such as line, shape, color, grid, or space go together to create well-rounded and thoughtful visuals.



This article defines 5 visual-design principles that impact UX:

1. **Scale**
2. **Visual hierarchy**
3. **Balance**
4. **Contrast**
5. **Gestalt**

## 5 Visual-Design Principles in UX

Visual-design principles inform us how design elements go together to create well-rounded and thoughtful visuals. Graphics that take advantage of the principles of good visual design can drive engagement and increase usability.

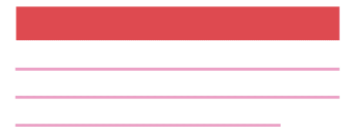
### SCALE

The principle of scale refers to using relative size to signal importance and rank in a composition.



### VISUAL HIERARCHY

The principle of visual hierarchy refers to guiding the eye on the page so that it attends to design elements in the order of their importance.



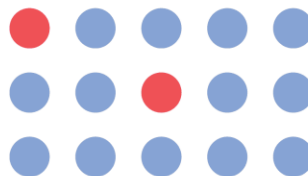
### BALANCE

Balance occurs when there is an equally distributed amount of visual signal on both sides of an imaginary axis.



### CONTRAST

The principle of contrast refers to the juxtaposition of visually dissimilar elements in order to convey the fact that these elements are different.



### GESTALT PRINCIPLES

Gestalt principles capture our tendency to perceive the whole as opposed to the individual elements.



NNGROUP.COM **NN/g**

**Fig. 5.22 Five visual-design principles can drive engagement and increase usability.**

### 1. Scale

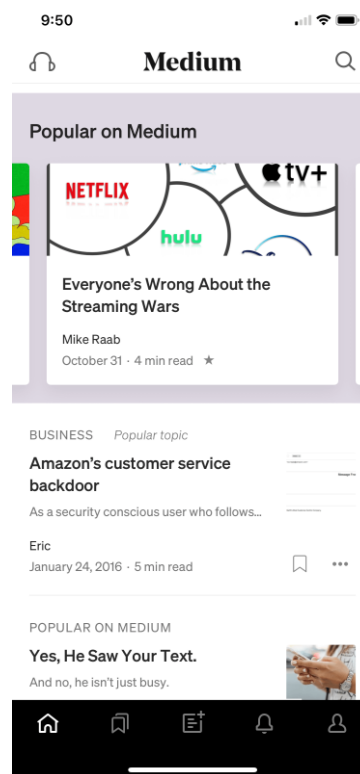
This principle is commonly used: almost every good visual design takes advantage of it.

**Definition:** The principle of **scale** refers to using relative size to signal importance and rank in a composition.

In other words, when this principle is used properly, the most important elements in a design are bigger than the ones that are less important. The reason behind this principle is simple: when something is big, it's more likely to be noticed.

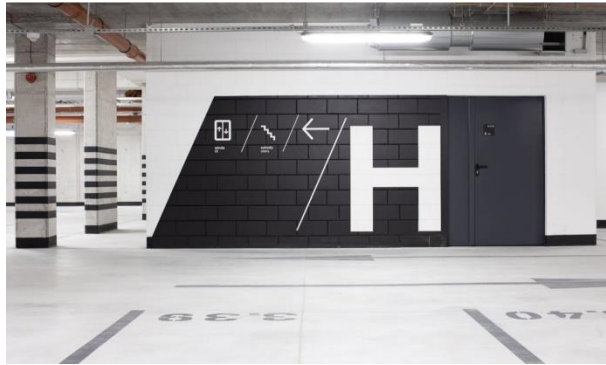
A visually pleasing design generally uses no more than 3 different sizes. Having a range of differently sized elements will not only create variety within your layout, but it will also establish a visual hierarchy (see next principle) on the page. Be sure to emphasize the most important aspect of your design by making them biggest.

When the principle of scale is used properly and the right elements are emphasized, users will easily parse the visual and know how to use it.



**Fig. 5.23 Medium for iphone**

Popular articles are visually larger than other articles. The scale directs users to potentially more-interesting article.



**Fig. 5.24** In this parking garage in Cracow, the most important piece of information (zone H—which is where you currently are in the parking garage), is the largest in size. (Image source: [www.behance.com](http://www.behance.com))

## 2. Visual Hierarchy

A layout with a good visual hierarchy will be easily understood by your users.

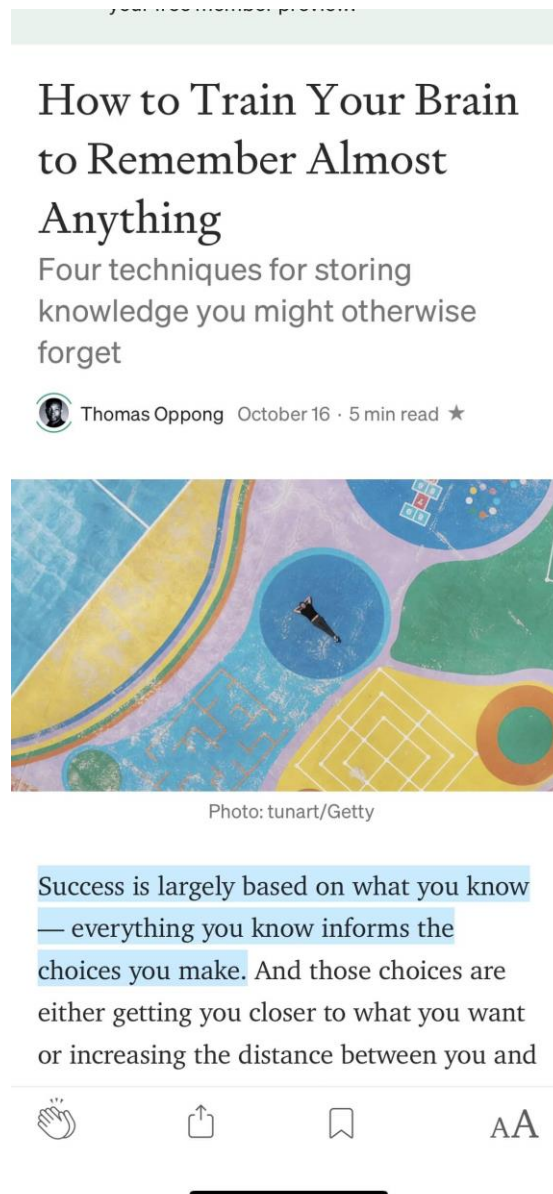
**Definition:** The principle of **visual hierarchy** refers to guiding the eye on the page so that it attends to different design elements in the order of their importance.

Visual hierarchy can be implemented through variations in scale, value, color, spacing, placement, and a variety of other signals.

Visual hierarchy controls the delivery of the experience. If you have a hard time figuring out where to look on a page, it's more than likely that its layout is missing a clear visual hierarchy.

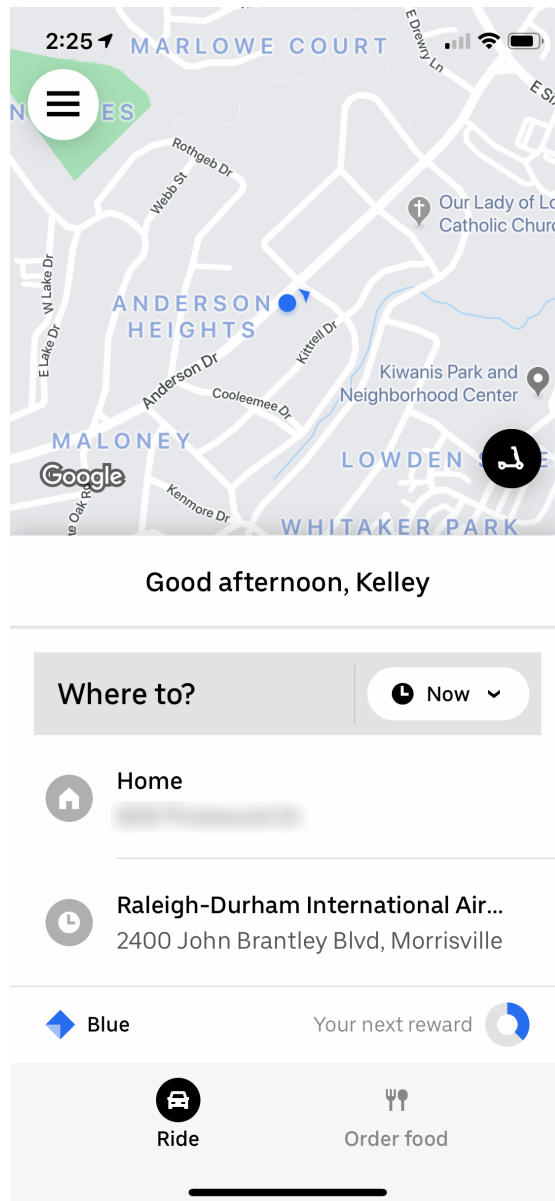
To create a clear visual hierarchy, use 2–3 typeface sizes to indicate to users what pieces of content are most important or at the highest level in the page's mini information architecture. Or, consider using bright colors for important items and muted colors for less important ones.

Scale can also help define the visual hierarchy, so incorporate various scales for your different design elements. A general rule of thumb is to include small, medium, and large components in the design.



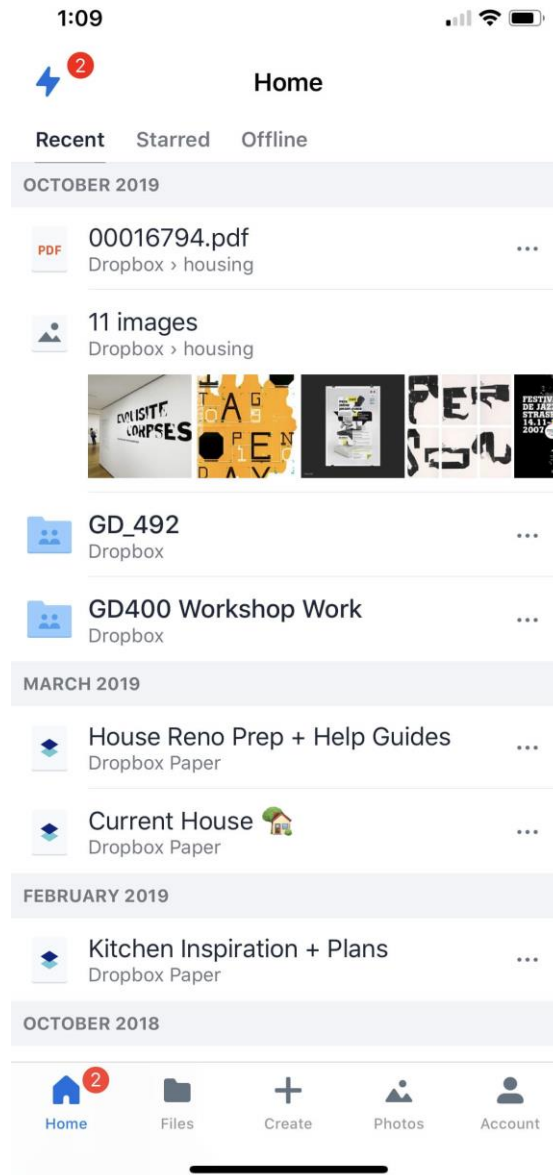
**Fig. 5.24 Medium mobile app**

There is a clear visual hierarchy of title, subtitle, and body text. Each component of the article is in a type size equal to its importance.



**Fig. 5.25 Uber mobile app**

The visual hierarchy is clear in Uber's mobile app. The screen is split in half between the map and input form (bottom half of screen), which enforces the thought that these components are equally important to the user. The eye is immediately drawn to the Where to? field because of its gray background, then to the recent locations below it, which are slightly smaller in font size.



**Fig. 5.26 Dropbox mobile app**

The visual hierarchy is less clear in Drop box’s mobile app. Even though the explanatory text is lower in size than the file name, it’s hard to distinguish among the different files. Thumbnails provide an additional layer to the hierarchy, but their presence depends on the available file types. Users ultimately have to do a lot of parsing and reading to find the folder or file they are looking for.

### 3. Balance

Balance is like a seesaw: instead of weight, you are balancing design elements.

**Definition:** The principle of **balance** refers to a satisfying arrangement or proportion of design elements. Balance occurs when there is an equally distributed (but not necessarily symmetrical)

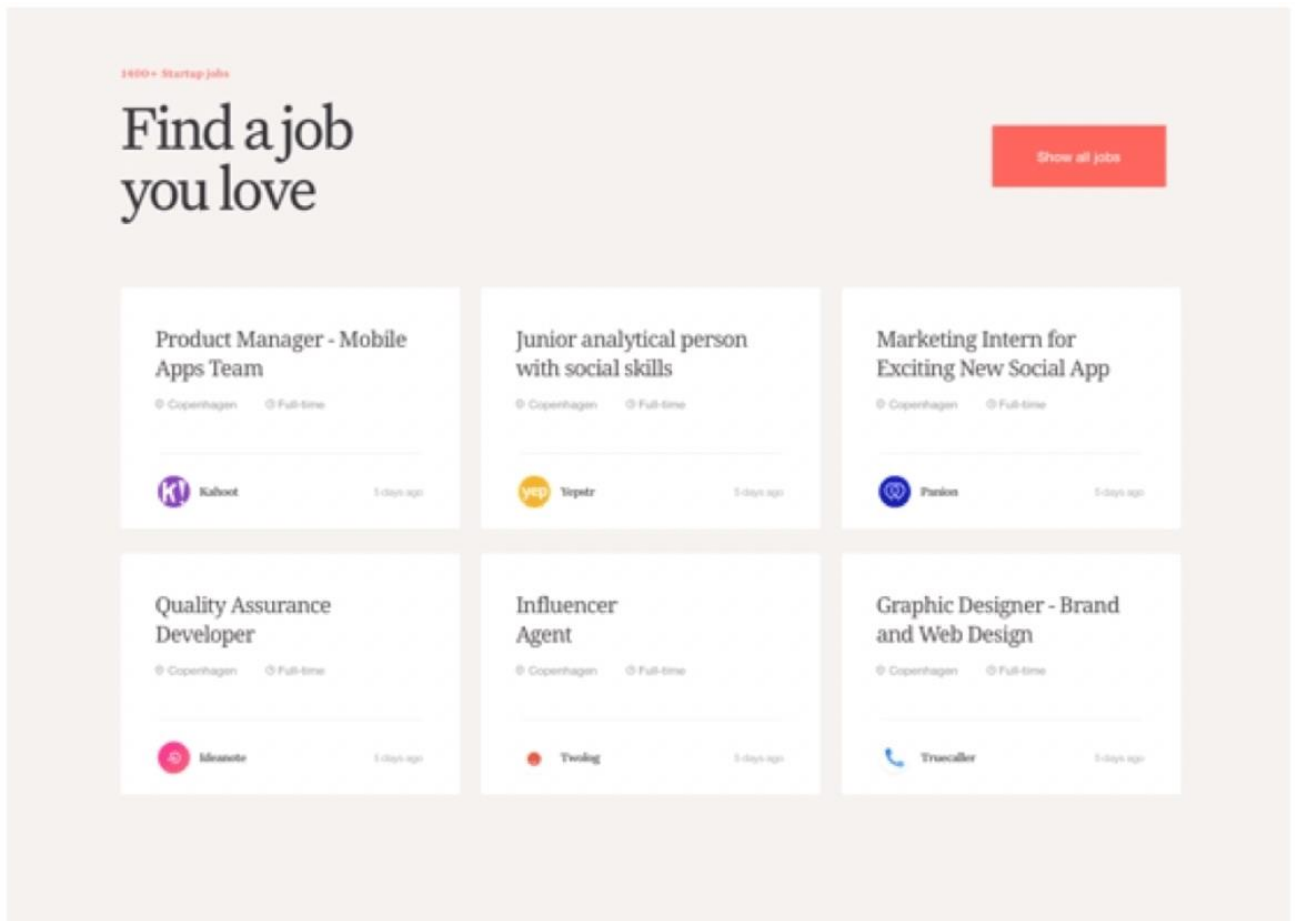
amount of visual signal on both sides of an imaginary axis going through the middle of the screen. This axis is often vertical but can also be horizontal.

Just like when balancing weight, if you were to have one small design element and one large design element on the two sides of the axis, the design would feel a bit unbalanced. The area taken by the design element matters when creating balance, not just the number of elements.

The imaginary axis you establish on your visual will be the reference point for how to organize your layout and will help you understand the current state of balance on your visual. In a balanced design, no one area draws your eye so much that you can't see the other areas (even though some elements might carry more visual weight and be focal points). Balance can be:

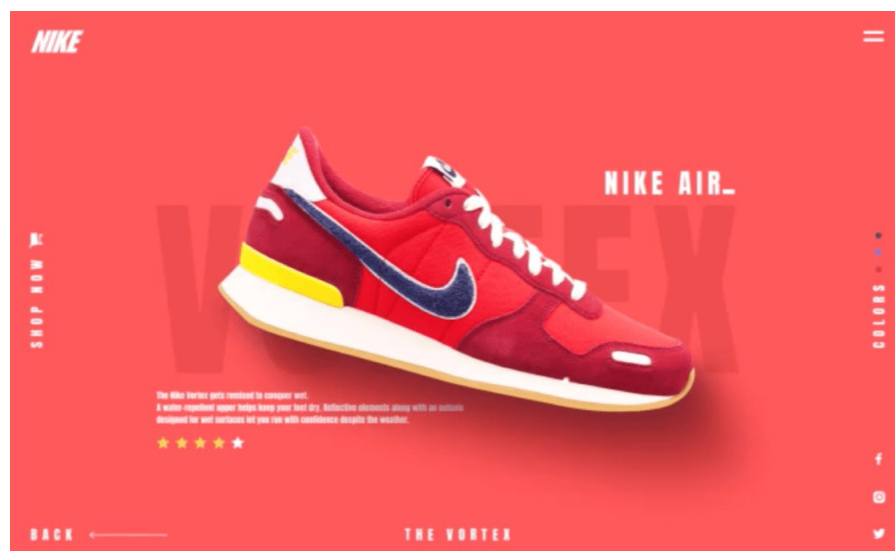
- Symmetrical: elements are symmetrically distributed relative to the central imaginary axis
- Asymmetrical: elements are asymmetrically distributed relative to the central axis
- Radial: elements radiate out from a central, common point in a circular direction.

The kind of balance you use in your visual depends on what you want to convey. Asymmetry is dynamic and engaging. It creates a sense of energy and movement. Symmetry is quiet and static. Radial balance will always lead the eye to the center of the composition.



**Fig. 5.27 the Hub Style Exploration**

The composition feels stable, which is especially appropriate when you're looking for a job you love. The balance here is symmetrical. If you were to draw an imaginary vertical axis down the center of the website, elements are distributed equally on both sides of the axis. (Image source: [dribbble.com](https://dribbble.com))



**Fig. 5.28 Nike**

This page is asymmetrically balanced, giving a sense of energy and movement that is fitting to Nike's brand. If you were to draw a vertical axis down the center on this visual, the number of elements is

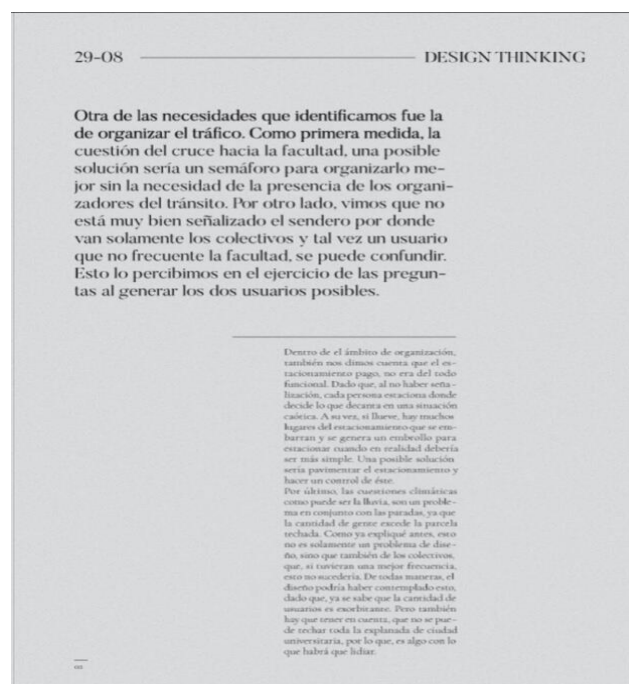


about the same on both sides of the axis. However, the difference is that they are not identical and in the same exact locations. Even though there is technically a bit more text on the left side of the shoe, it is balanced out with the larger text on the right that takes up more space and visual weight, thus making them appear pretty similar.



**Fig. 5.29 Brathwait wrist watch**

This classic watch is balanced radially. The eye is immediately drawn to the center of the watch face and all visual weight is distributed equally, regardless of where the imaginary axis is drawn.



**Fig. 5.30 This editorial spread is not balanced. If you were to draw a vertical axis down the page, elements are not equally distributed on both sides of the axis. (Image source: [www.behance.net](http://www.behance.net))**

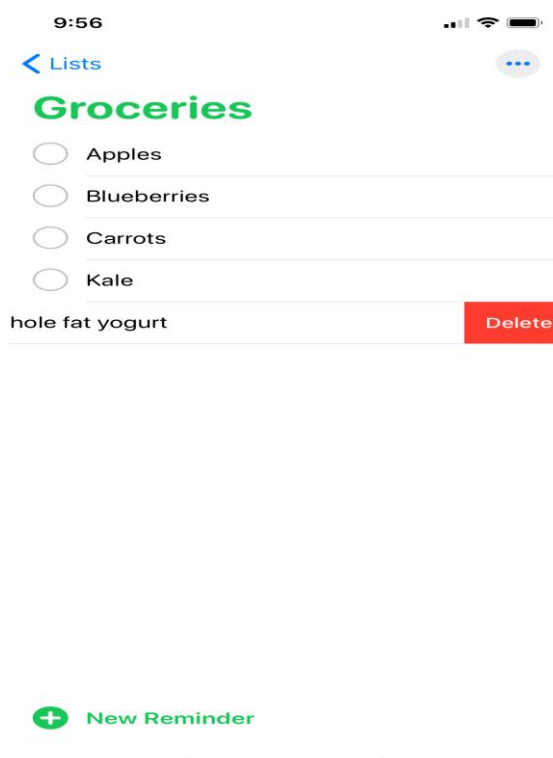
#### **4. Contrast**

This is another commonly used principle that makes certain parts of your design stand out to your users.

**Definition:** The principle of **contrast** refers to the juxtaposition of visually dissimilar elements in order to convey the fact that these elements are different (e.g., belong in different categories, have different functions, behave differently).

In other words, contrast provides the eye with a noticeable difference (e.g., in size or color) between two objects (or between two sets of objects) in order to emphasize that they are distinct.

The principle of contrast is often applied through color. For example, red is frequently used in UI designs, especially on iOS, to signify deleting. The bright color signals that a red element is different from the rest.



**Fig. 5.31 Reminders app on iOS**

The color red, which has high contrast to its surrounding context, is reserved for deleting.

Often, in UX the word “contrast” brings to mind the contrast between text and its background. Sometimes designers deliberately decrease the text contrast in order to deemphasize less important text. But this approach is dangerous — reducing text contrast also reduces legibility and may make your content inaccessible. Use a color-contrast checker to ensure that your content can still be read by all your target users.



**Fig. 5.32 Greenhouse Juice Co: The legibility of the text on the bottle relies on the color of juice. Although the contrast works beautifully for some juices, labels for bottles with light colored juices are nearly impossible to read. (Image source: [www.instagram.com](http://www.instagram.com))**

## 5. Gestalt Principles

These are a set of principles that were established in the early twentieth century by the Gestalt psychologists. They capture how humans make sense of images.

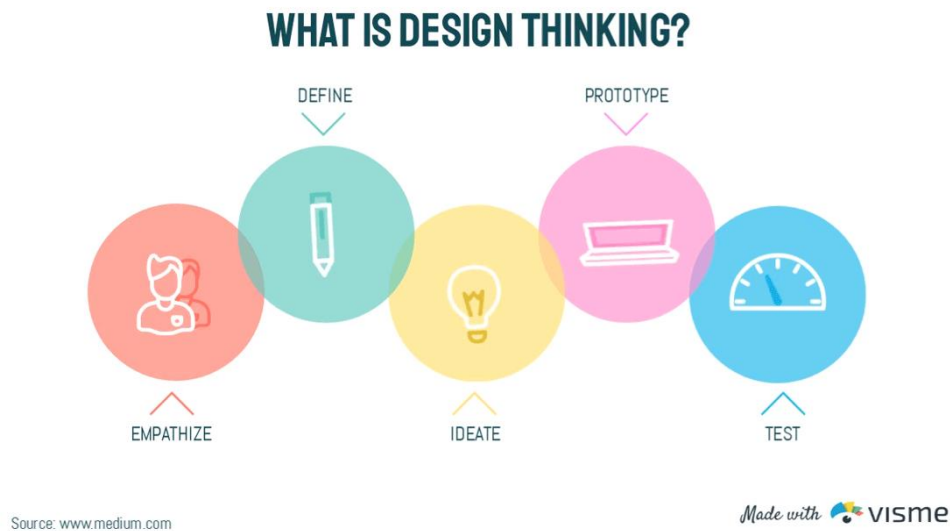
**Definition: Gestalt principles** explain how humans simplify and organize complex images that consist of many elements, by subconsciously arranging the parts into an organized system that creates a whole, rather than interpreting them as a series of disparate elements. In other words, Gestalt principles capture our tendency to perceive the whole as opposed to the individual elements.

There are several Gestalt principles, including similarity, continuation, closure, proximity, common region, figure/ground, and symmetry and order. Proximity is especially important for UX — it refers to the fact that items that are visually closer together are perceived as part of the same group.

## 8. INFORMATION DESIGN AND VISUALIZATION

**Information design** is data used as a storytelling tool. It's data with a purpose. Therefore, Information visualizations are more about informing the viewer about a data set and it's specific parts. Conclusions have already been made for that data, and it's being presented in a snackable design. Data visualizations are raw data visualized in a way that permits the viewer to make their own conclusions. Data visualizations can be ever-evolving visuals with new data and information being added regularly.

They can also include data from a specific point in time and can be organized in a way that inspires a distinctive reaction. Nevertheless, it can still be analyzed and direct viewers to their own conclusions. Not only must information be presented in a clear manner, but users also need to navigate the information without it being overwhelming or confusing.



**Fig. 5.33 Design process of Project**

Let's take a look at how even a simple information design project follows the user experience design process.

Imagine for example, a large set of safety posters to be displayed on a factory floor.

## User Research

The user research step is all about getting to know the user and what their needs are. It's all about getting to know them in their environment and figuring out the best way to help.

### 1. Empathize

The first step of user research is to empathize with the user. In our example, the information designer visits the factory floor where the posters will be displayed. They will meet the workers and talk to them about what their days and movements in the factory look like.

In cases where it's not possible to do such deep research, the designer asks more questions to get a sense of what the factory floor is like. If they are only talking to the floor manager, the designer insists on talking to the union leader of the workforce.

## **2. Define**

This is the stage where the problem to be solved is defined. It's likely that the problem as a whole has been presented from the beginning.

In this case, "we need safety posters on the factory floor."

But only after the designer communicates with the client and the factory workers, will the real problems show themselves.

For example, a few problems that could arise are:

- The previous posters weren't laminated so the information faded fast under the lights.
- They didn't have enough visuals and nobody took the time to read all the instructions.
- The posters were posted in a section of the factory floor where few workers walked past so not everyone saw them.

### **Ideate**

After the user research, it's time to come up with some ideas for the safety posters project. The questions to answer could be:

- How big should the posters be?
- How should the information be visualized so it's attractive and easy to read?
- Where should the posters be placed so that they are not missed?
- Can we think of any other design techniques to make this project a success for all parties involved

### **Prototype**

Once the questions have been answered, the designer puts together a first draft or prototype. It's displayed in a specific place, for example, the hall outside the bathrooms. They are placed at a general eye-level, laminated, and full of visual instructions.

### **Test**

No information design project is finished until it has been tested by the users. After a few days with prototypes on the floor, the information designer will ask for feedback.

One possibility is that the posters are placed on the wall next to the doors of the bathroom, therefore when the workers pass by them on their way to the bathroom they are usually in a hurry and don't stop to look.

If the posters are put on the opposite wall, they will see them on the way out and will be in a more relaxed state to stop and study the posters. Of course after a while, employees will already know the information and stop looking.

Nevertheless, the posters must stay full of color and attractive for newly hired workers.

This process is followed by all information design projects big or small. It's the number one proven way to make sure the visualizations do their job well.

Safety posters are not the only type of information design. In fact, the possibilities are far-reaching. Let's take a look at some of the most common types.



**Fig. 5.34**Types of Information Design

Information and data are all around us. Everything we do collects data.

For example, our devices are constantly collecting data about how we shop, communicate, what we like to do, how our health is, and what our life, in general, is like.

Information design takes on an important role in this flow of data and information. It's essentially a way of putting together chunks of relevant information to make it easy to understand for users.

Information design is often visual, but can also be sensory. Some types are even physically interactive through sound and smell. The best examples also take accessibility into account.

## **9. INTERACTION DESIGN**

The purpose of interaction design is to create a great user experience. That's why most of the UI disciplines require understanding and hands-on experience of interaction design principles. After all, it's about designing for the entire interconnected system: the device, interface, context, environment, and people. Interaction designers strive to create meaningful relationships between people and the products and services they use. It may include computers, mobile devices, gadgets, appliances, and more.

It is important to understand ux design best practices while developing complex web and mobile applications. These are the key elements that product designers should not neglect while creating an interface for the user.

The 10 most important interaction design principles are-

1. **UX:** Match user experience and expectations
2. **Consistent design:** Maintain consistency throughout the application
3. **Functionality:** Follow functional minimalism
4. **Cognition:** Reduce cognitive loads/mental pressure to understand the application
5. **Engagement:** Design interactively such that it keeps the user engaged.
6. **User control:** Allow the user to control, trust, and explore
7. **Perceivability:** Invite interactions through intuitions and interactive media
8. **Learnability:** Make user interactions easy to learn and remember
9. **Error handling:** Take care to prevent errors, if they occur make sure to detect and recover them.
10. **Affordability:** Simulate actions by taking inspiration from usual and physical world interactions.

## 10 Important Interaction Design Principles

### #1 Match user experience and expectations

By matching the sequence of steps, layout of information, and terminology used with the expectation and prior experiences of the users, designers can reduce the friction and discomfort of learning a new system.

You can match your audience's prior experiences and expectations by using common conventions or UI patterns, for example, [Hitee Chatbot](#).

### #2 Consistency

Along with matching people's expectations through terminology, layout, and interactions, the design should be consistent throughout the process and between related applications.

By maintaining consistency, you are helping users learn more quickly. You can re-apply their prior experiences from one part of an application to another to maintain consistency throughout the design. Design consistency is also an aid to intuitive interfaces.

*Bonus – you can use the inconsistencies to indicate to users where things might not work the way they expect. Breaking consistency is similar to knowing when to be unconventional.*

### #3 Functional minimalism

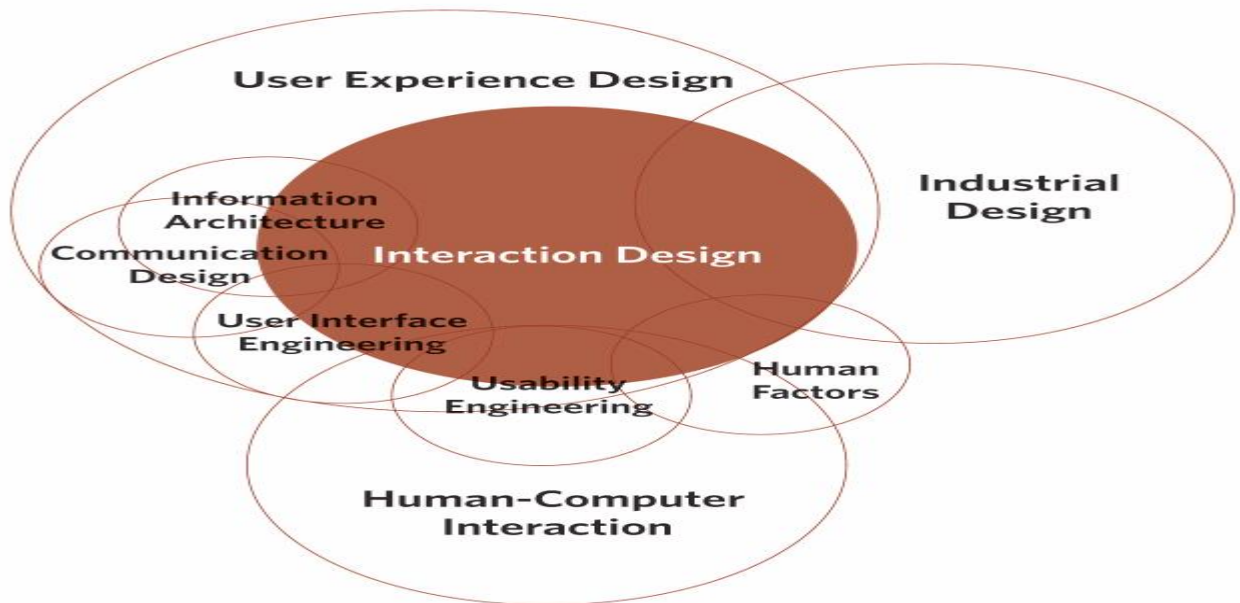
“Everything should be made as simple as possible, but no simpler.”

*Albert Einstein*

The range of possible actions should be no more than is absolutely necessary. Providing too many options will detract the primary function and reduce usability by overwhelming the user with choices. To achieve the Zen of functional minimalism, you should-



1. Avoid unnecessary features and functions
2. Break complex tasks into manageable sub-tasks
3. Limit functions rather than the user experience.



**Fig 5.35 Interaction design**

#### **#4 Cognitive loads**

Cognition refers to the “process of thoughts.” A good user interactive design minimizes the user’s “effort to think” to complete a task. Another way to put this is that a good assistant uses his skills to help the master focus on his skills.

For instance, while designing an interactive interface, we need to understand how much concentration a task requires to complete it. Accordingly, you can design the UI that reduces the cognitive load as much as possible.

Here’s a technique to reduce users’ “thinking work.” Focus on what the computer is good at and build a system that utilizes its abilities to the fullest. Remember, computers are good at-

- Maths
- Remembering things
- Keeping track of things
- Comparing things
- Spell Checking and spotting/correcting errors

The point is – by knowing the attributes of users and products, one can create a design for a better user experience.

#### **#5 Engagement**

In terms of user experience, engagement is the measure of the extent to which the user has a positive experience with your product. An engaging experience is not only enjoyable but also easier and productive. Engagement is subjective to the system. I.e. your design must engage with the desired



audience. For instance, what appeals to teenagers might be irrelevant to their grandparents. Apart from aligning your design for the appropriate audience, *achieving and creating control is the key*. The interaction design principles state that users should always feel like they're in control of the experience. They must constantly experience a sense of achievement through positive feedback/results or feel like they've created something.

In his book "[Flow](#)," [Mihaly Csikszentmihalyi](#) describes a state of optimal experience where people are so engaged in the activity that the rest of the world falls away. Flow is what we're looking to achieve through engaging interactions. We should *allow users to concentrate on their work* and not on the user interface. In short, stay out of the way!

### **#6 Control, trust, and explorability**

Good interaction design should incorporate control, trust, and explorability to any system. If users feel in control of the process, they'll be more comfortable using the system. If the user is comfortable and in control, they'll trust the system and believe that the application will prevent them from making an unrecoverable error or from feeling stupid. Trust inspires confidence and with confidence, the user is free to explore further. Intuitive interfaces are extremely good at stimulating users to navigate and explore the app.

### **#7 Perceivability**

People are aware of the opportunity to interact with interactive media. As interface designers, we must avoid developing hidden interactions, which decrease the usability, efficiency, and user experiences. In other words, people should not have to guess or look for opportunities to interact.

When developing interactive media, users should have the ability to review an interface and identify where they can interact. We must remember that not everyone experiences and interacts with interface in the same way others do. In the process of interaction design, make it a habit to provide hints and indicators like buttons, icons, textures, textiles, etc. Let the user see that these visual cues can be clicked or tapped with their fingers. Always consider the usability and accessibility of the interactive media and how the user sees and perceives the objects in the interface.

### **#8 Learnability**

Another important interaction design principle is inducing the ability to learn to use the interface easily. In other words, users should be able to learn to use the interface in the first attempt and should not face issues using it again. Please note that engaging interfaces allow users to easily learn and remember the interactions.

Even though simple interfaces may require a certain amount of experience to learn, learnability makes interaction intuitive. People tend to interact with an interface similar to other interfaces. This is the reason why we must understand the process of interaction design thoroughly and the importance of design patterns and consistency.

Intuitive interface design allows users to learn to use the interface without much effort and gives them a sense of achievement. They feel smart and capable of grasping and utilizing newer interfaces. In a nutshell, product designers should let the user feel confident while navigating through the interface.

### **#9 Error prevention, detection, and recovery**

The best way to reduce the number of errors a user makes is to anticipate possible mistakes and prevent them from happening in the first place. If the errors are unavoidable, we need to make them easy to spot and help the user to recover from them quickly and without unnecessary friction.

### ***Error prevention techniques-***

- Disabling functions that aren't relevant to the user
- Using appropriate controls to constrain inputs (e.g. radio buttons, dropdowns, etc.)
- Providing clear instructions and preemptive help
- As a last resort, provide clear warning messages.

### ***How to handle application errors through design?***

Anticipate possible errors and provide feedback that helps users verify that-

1. They've done what they intended to do.
2. What they intended to do was correct.

Please note that providing feedback by changing the visuals of the object is more noticeable than a written message.

### ***Error recovery techniques –***

If the error is unavoidable, provide direction to the user to recover from it. For example, you can provide “back,” “undo,” or “cancel” buttons.

If a specific action is irreversible, you should flag it “critical” and make the user confirm first to prevent slip-ups. Alternatively, you can create a system that naturally defaults to a less harmful state. For example, closing a document without saving it should be intelligent enough to know the unlikely behavior of the user. It can either auto-save or display a warning.

## **#10 Affordance**

Affordance is the quality of an object that allows an individual to perform an action. For example, a standard household light switch appears innately clickable.

The point is – users should get a clue about how to use an object through its physical appearance. While designing user interfaces, you can achieve affordance either by simulating ‘physical world’ affordances (e.g. buttons or switches) or keeping consistency with web standards and interface design elements (e.g. underlined links or default button styles). The thing is, in an intuitive interface, users are able to navigate and use the functionalities of the application without any formal training.

Interaction design is not always about creating a better interface for the users; it is also about using technology in the way people want. It is necessary to know the target users to design a desirable product for them. Interactive design is the basis for the success of any product. These **10 interaction design principles** are based on the study and experiences of our team in designing mobile and web apps for a broad product portfolio and on multiple mobile and web platforms.

## **10. PROTOTYPING TOOLS**

By definition, a prototype is an early sample, model, or release of a product built for the purpose of testing a concept or process. Generally, the prototype is used to evaluate a new product or concept design for its usefulness in the real world.

Additionally, the main motive behind a prototype is to validate the design with your target market by collecting feedback that will guide you during product development.

### **Importance of prototyping**

Would you ever walk into a stakeholder meeting to present your concept without first getting customer feedback? Hopefully not. Doing so could undermine your credibility and capacity to defend your design.

Communicating and justifying the value, look, and feel of your product to stakeholders can be a challenge. But through prototyping you're able to:

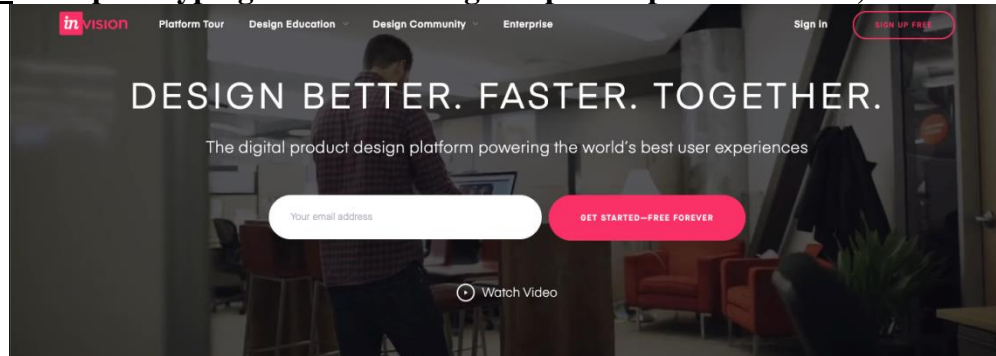
- Better depict the intent of your final design
- Defend your design decisions with customer feedback
- Save time and money by making changes early rather than in final development
- Feel confident that what you're presenting has a strong product-market fit

Once you've bought into the value of prototyping, how do you design your mobile app, website, online forms, and various other prototypes in a way you can get in front of your customers? The best prototyping tools can make all the difference.

### **Best prototyping tools for UI/UX designers**

To help you select the best prototyping tool for you and your business, here's a list of five of the most popular, affordable, and accessible solutions you can use today. By no means is this list comprehensive, but we've compiled some of our favorites for your consideration.

#### **1. InVision: best prototyping tool for building exceptional products in one, connected workflow**



**Fig 5.36 InVision**

InVision makes it easy to create interactive prototypes. Simply upload static screenshots and create clickable prototypes your users can interact with and understand. The app runs on the web and works well simplifying the workflow between designers and other stakeholding teams.

- Create rich interactive prototypes that allow for rapid iteration
- Seamlessly communicate, gather feedback, and advance projects
- Compatibility with most popular graphics file formats like PNG, GIF, PSD, JPG, and others
- Pricing: Free to \$100/month

#### **2. Balsamiq: best prototyping tool for building wireframes**

Balsamiq helps you quickly design mockups that are great for sketching and wireframing. With excellent ease of use, a great widget library, and its cloud-based software, it makes team collaboration easy.

- Allows for quick wireframing so you can learn fast and fail faster
- Minimal learning curve with powerful technology
- Drag and drop simplicity
- Pricing: \$9-\$199/month, discounts for annual subscriptions

#### **3. Justinmind: best all-in-one prototyping tool for web and mobile apps**

Justinmind is an all-in-one prototyping tool for web and mobile apps that helps you build wireframes to highly interactive prototypes without any coding. Justinmind lets you design from scratch and leverage a full range of web interactions and mobile gestures, so you can focus on building exceptional user experiences.

- Responsive prototyping that ensures your designs adapt to fit multiple screen resolutions
- Prototype smart forms and data lists without writing code
- Free, ready-made, regularly-updated UI kits for web and mobile
- Pricing: \$19-\$49/month, discounts for annual subscriptions\

#### **4. Figma: best prototyping tool for designing collaborative**

Figma is truly a one-tool solution for all of your design needs. Thanks to real-time collaboration, web-based functionality, and exceptional price-to-value, Figma is rising through the ranks and gaining traction with design teams.

- Easy switching between design and prototype modes
- Quick sharing and real-time feedback

- Powerful editing features
- Pricing: Free to \$45/month. Some plans are available with annual pricing only.

#### **5. Adobe XD: best prototyping tool for enterprise business**

Wireframing, designing, prototyping, presenting, and sharing amazing experiences for web, mobile, voice, and more— Adobe XD is your all in one app. Adobe XD provides you access to all your assets in one place, eliminates tedious manual tasks, create experiences that are adaptable to any size screen, and integrates seamlessly with the UserTesting platform.

- High-fidelity interactions
- Integration with the other Adobe products you already use
- Real-time viewing, commenting, and sharing capabilities
- Pricing: Free to \$23/month

### **11. USABILITY TEST**

Usability testing refers to evaluating a product or service by testing it with representative users. Typically, during a test, participants will try to complete typical tasks while observers watch, listen and takes notes. The goal is to identify any usability problems, collect qualitative and quantitative data and determine the participant's satisfaction with the product.

To run an effective usability test, you need to develop a solid test plan, recruit participants , and then analyze and report your findings.

#### **Benefits of Usability Testing**

Usability testing lets the design and development teams identify problems before they are coded. The earlier issues are identified and fixed, the less expensive the fixes will be in terms of both staff time and possible impact to the schedule. During a usability test, you will:

- Learn if participants are able to complete specified tasks successfully and
- Identify how long it takes to complete specified tasks
- Find out how satisfied participants are with your Web site or other product
- Identify changes required to improve user performance and satisfaction
- And analyze the performance to see if it meets your usability objectives

#### **You Do Not Need a Formal Lab**

Effective Usability Testing does not require a formal usability lab for testing. You can do effective usability testing in any of these settings:

- Fixed laboratory having two or three connected rooms outfitted with audio-visual equipment
- Room with portable recording equipment
- Room with no recording equipment, as long as someone is observing the user and taking notes
- Remotely, with the user in a different location (either moderated or unmoderated)

#### **Factors Affecting Cost**

Your testing costs depend on

- Type of testing performed
- Size of the team assembled for testing
- Number of participants for testing
- Number of days you will be testing

Remember to budget for more than one usability test. Building usability into a Web site (or any product) is an iterative process. Consider these elements when budgeting for usability testing:

- **Time:** You will need time to plan the usability test. It will take the usability specialist and the team time to become familiar with the site and pilot test the test scenarios. Be sure to budget in

time for this test prep as well as running tests, analyzing the data, writing the report, and presenting the findings.

- **Recruiting Costs:** Consider how or where you will recruit your participants. You will either need to allow for staff time to recruit or engage a recruiting firm to schedule participants for you based on the requirements.
- **Participant Compensation:** If you will be compensating participants for their time or travel, factor that into your testing budget.
- **Rental Costs:** If you do not have monitoring or recording equipment, you will need to budget for rental costs for the lab or other equipment. You may also need to secure a location for testing, a conference room for example, so consider this as well.

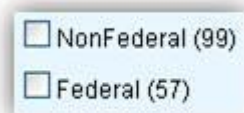
## 12. USER INTERFACE COMPONENTS

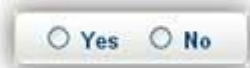

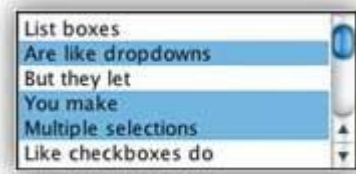

When designing your interface, try to be consistent and predictable in your choice of interface elements. Whether they are aware of it or not, users have become familiar with elements acting in a certain way, so choosing to adopt those elements when appropriate will help with task completion, efficiency, and satisfaction.





Interface elements include but are not limited to:

- **Input Controls:** checkboxes, radio buttons, dropdown lists, list boxes, buttons, toggles, text fields, date field
- **Navigational Components:** breadcrumb, slider, search field, pagination, slider, tags, icons
- **Informational Components:** tooltips, icons, progress bar, notifications, message boxes, modal windows
- **Containers:** accordion




### Input Controls

Element	Description	Examples
<b>Checkboxes</b>	Checkboxes allow the user to select one or more options from a set. It is usually best to present checkboxes in a vertical list. More than one column is acceptable as well if the list is	



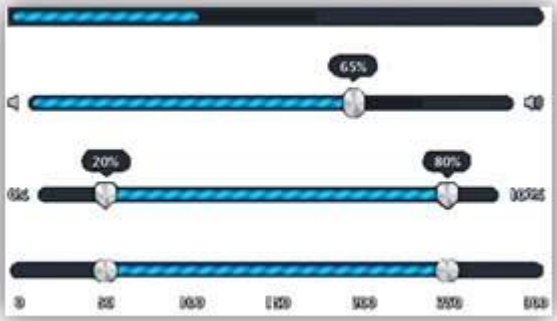
Element	Description	Examples
	long enough that it might require scrolling or if comparison of terms might be necessary.	
<b>Radio buttons</b>	Radio buttons are used to allow users to select one item at a time.	
<b>Dropdown lists</b>	Dropdown lists allow users to select one item at a time, similarly to radio buttons, but are more compact allowing you to save space. Consider adding text to the field, such as 'Select one' to help the user recognize the necessary action.	
<b>List boxes</b>	List boxes, like checkboxes, allow users to select a multiple items at a time, but are more compact and can support a longer list of options if needed.	
<b>Buttons</b>	A button indicates an action upon touch and is typically labeled using text, an icon, or both.	



Element	Description	Examples
<b>Dropdown Button</b>	The dropdown button consists of a button that when clicked displays a drop-down list of mutually exclusive items.	
<b>Toggles</b>	A toggle button allows the user to change a setting between two states. They are most effective when the on/off states are visually distinct.	
<b>Text fields</b>	Text fields allow users to enter text. It can allow either a single line or multiple lines of text.	
<b>Date and time pickers</b>	A date picker allows users to select a date and/or time. By using the picker, the information is consistently formatted and input into the system.	

## Navigational Components


Element	Description	Examples
<b>Search Field</b>	A search box allows users to enter a keyword or phrase (query) and submit it to search the index with the intention of getting back the most relevant results. Typically search fields are single-line text boxes and are often accompanied by a search button.	
<b>Breadcrumb</b>	Breadcrumbs allow users to identify their current location within the system by providing a clickable trail of preceding pages to navigate by.	
<b>Pagination</b>	Pagination divides content up between pages, and allows users to skip between pages or go in order through the content.	

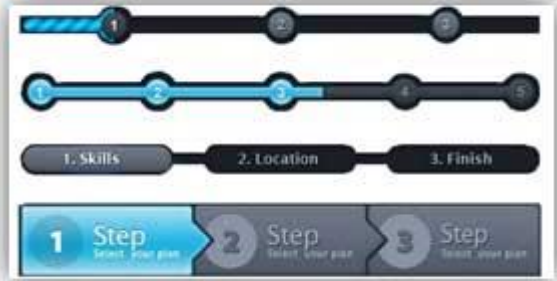





Element	Description	Examples
Tags	Tags allow users to find content in the same category. Some tagging systems also allow users to apply their own tags to content by entering them into the system.	 
Sliders	A slider, also known as a track bar, allows users to set or adjust a value. When the user changes the value, it does not change the format of the interface or other info on the screen.	

Element	Description	Examples
<b>Icons</b>	An icon is a simplified image serving as an intuitive symbol that is used to help users to navigate the system. Typically, icons are hyperlinked.	
<b>Image Carousel</b>	Image carousels allow users to browse through a set of items and make a selection of one if they so choose. Typically, the images are hyperlinked.	

## Information Components

Element	Description	Examples
<b>Notifications</b>	A notification is an update message that announces something new for the user to see. Notifications are typically used to	

Element	Description	Examples
	<p>indicate items such as, the successful completion of a task, or an error or warning message.</p>	
<b>Progress Bars</b>	<p>A progress bar indicates where a user is as they advance through a series of steps in a process. Typically, progress bars are not clickable.</p>	
<b>Tool Tips</b>	<p>A tooltip allows a user to see hints when they hover over an item indicating the name or purpose of the item.</p>	

Element	Description	Examples
<b>Message Boxes</b>	A message box is a small window that provides information to users and requires them to take an action before they can move forward.	
<b>Modal Window (pop-up)</b>	A modal window requires users to interact with it in some way before they can return to the system.	

## Containers

Element	Description	Examples
<b>Accordion</b>	An accordion is a vertically stacked list of items that utilizes show/ hide functionality. When a label is clicked, it expands the section showing the content within. There can have one or more items showing at a time and may have default states that reveal one or more sections without the user clicking	

## 13. TOOLS AND PROCESSES

Process improvement plays a very vital role in organizations. Lean Six Sigma is one means for creating a deployment that is to improve the business. Through this effort there should be an upbeat task of determining, analyzing and enhancing an organization's business processes to achieve optimization and new quality standards. Process improvement efforts should entail a systematic approach that adheres to a certain methodology, where the specific approaches to accomplish this task may differ.

When undertaking a process improvement endeavor, more efficient outcomes are expected. Process improvement may involve a sequence of actions to attain new objectives and goals, like improving performance, reducing costs and elevating profits. Such actions may follow a particular technique or methodology to increase the odds of achieving successful results.

The following five tools should be included in these process improvement execution roadmaps:

## 1. Process Baseline and Process Comparisons

In general, four processes can be involved when baselining and determining how a process is performing relative to other similar processes:

- **Building Baseline** – Create a clear business or organizational baseline. This effort would entail defining the baseline about all the business aspects.
- **Classifying Baseline** – Determine all the organizations that might be compared for performance. Different factors may enter into this decision; e.g., would it be beneficial to compare performance to a leading organization within the same industry or one that has a similar culture.
- **Do Comparison** – Observe how organizational baselines compare. Comparisons should be made statistically; e.g., a hypothesis of equality of means for a process-output response.
- **Determine Baseline Differences** – Identify the reasons for differences in performance. This understanding can help an organization make adjustments to their process so that performance improves.

## 2. *Flowcharting*

Flowcharting is one of the best tools for documenting and understanding various processes in an organization. This tool allows for a detailed breakdown of processes to activities and events, as well as describing logical relationships. By using flowcharts, an organization can better understand the work efforts involved in all their undertakings.

For instance, the process of getting orders and encoding them in the computer system can be represented more clearly by using a flowchart consisting of symbols that represent every event or step in the process. These can be circles, boxes or other forms of shapes that are connected with lines to direct the order or direction of the process. A good flowchart can help in communicating and clarifying what is happening or what needs to take place in an organization.

### 3. Value-Stream Mapping

Value stream mapping provides a picture of work flow and information flow in an end-to-end process. One can evaluate a current state and propose a future state that reduces organizational waste; i.e.,

transport, inventory, motion, waiting, over production, over processing, and defects. Much can be gained by examining a value-stream map; e.g., where improvement efforts or kaizen events should focus to improve a current 30,000-foot-level baseline value-stream map's performance metrics such as lead time.

**4. Cause and Effect Analysis**  
Problems can often be resolved by first exploring all possible causes. A cause-and-effect analysis approach provides a structure for this assessment, which involves the consideration of six areas or causes that can contribute to a characteristic response or effect: materials, machine, method, personnel, measurement and environment.

With this approach for evaluating a problem, a solution might become immediately apparent. In other cases, the potential cause may not be so obvious; however, statistical analyses of historical data can be used to test-out various theories. This information can be used to provide insight as to what might be done to improve a 30,000-foot-level process output baseline performance.

**5. Hypothesis Testing**  
Hypothesis testing consists of a null hypothesis and alternative hypothesis where, for example, a null hypothesis indicates equality between two process outputs and an alternative hypothesis indicates non-equality. Through a hypothesis test, a decision is made on whether to reject a null hypothesis or not reject a null hypothesis, with a risk of making an erroneous decision. Hypothesis tests can take many formats. It is important to select the most appropriate hypothesis test for each situation.

### QUESTION BANK

S.No	Questions (2 marks)	Competence	BTL Level
1.	Define UX design.	Remember	BTL1
2.	Evaluate the method to improve UX skills	Create	BTL6
3.	Discuss the UX investigate Strategies.	Analyze	BTL4
4.	Devise the best tool to use for the researcher to develop the good UX design process.	Analyze	BTL4
5.	Write one element Component of the Facebook interface that improves content quality	Evaluate	BTL5
6.	Demonstrate the UX information needed to have before start designing.	Understand	BTL2
7.	Compare and contrast UX and UI	Analyze	BTL4

8.	List out the example for usability test	Analyze	BTL4
9.	Name Four basic components of User interface	Remember	BTL1
10.	Construct a Process Improvement Tool	Evaluate	BTL5
11.	Apply UX design process for pre-planning and production.	Apply	BTL3
12.	Construct Drop box's responsive color system for UI	Apply	BTL3
13.	Decide whether Linear Layout using custom View and View Group object are in form of UI.	Evaluate	BTL5
14.	Find out 5 visual-design principles that can drive engagement and increase usability.	Remember	BTL1
15.	Choose a method that makes a good interaction design?	Evaluate	BTL5

S.No	Questions (16 marks)	Competence	BT Level
1.	Describe the elements of UX design.	Understand	BTL2
2.	Discuss the six step UX process by apply to any known web application design.	Analyze	BTL4
3.	Gathering user data and feedback is a hectic process but UX research methods serve as a way of streamlining the task a bit and making the data more readable justify.	Evaluate	BTL5
4.	Explain the levels of UX pyramid.	Remember	BTL1
5.	Creating a Digital App or showing multiple steps to use a physical product- Draw different sketches of the user interface on different piece of paper and simulate interactivity by linking and moving series of paper on table.	Create	BTL6
6.	Design the UX research methods for streamlining the task and reading the data.	Evaluate	BTL5
7.	Classify the Navigational Components for UI Elements	Understand	BTL2
8.	Organize a prototyping tools for i) Frame X ii) MOCPLUS iDOC with UX design.	Apply	BTL3

9.	Construct process analysis tool for FMEA (failure modes effects analysis)	Create	BTL6
10.	Utilize how the elements together build pages and app screens optimally and how to achieve unity, gestalt, hierarchy, balance, contrast, scale, similarity of visual design.	Apply	BTL3
11.	Examine the five stages of UX Design process in order.	Analyze	BTL4
12.	Justify if the labels “high” and “medium-high” are absent for each pie chart. Do you think new website visitors would be able to understand what each set of data implies for practice of information design aims to address challenging scenarios of data to make sense?	Evaluate	BTL5