Training Report on Python

Submitted in partial fulfillment of the requirements for

Bachelor of Engineering Degree in

Electrical and Electronics

Engineering By

Snowsha J(39140054)

DEPARTMENT OF ELECTRICAL AND

ELECTRONICS SCHOOL OF ELECTRICAL AND

ELECTRONICS SATHYABAMA INSTITUTE OF

SCIENCE AND TECHNOLOGY JEPPIAAR

NAGAR, RAJIV GANDHI SALAI,

CHENNAI – 600119. TAMILNADU.

JANUARY - 2021

SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY

(Established under Section 3 of UGC Act, 1956) Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai –600119

SCHOOL OF ELECTRICAL AND ELECTRONICS

<u>BONAFIDE CERTIFICATE</u>

This is to certify that this Professional Training Report is the Bonafide work o**f Snowsha J (39140054)** who underwent the professional training in **"Python"** from 20TH AUGUST to 20TH OCTOBER

Internal Guide

Head of the Department

Dr. SIVA CHIDAMBARANATHAN M.E., Ph.D.,

MENTOR

[illegible]

[illegible]

1stMentor.com/certificate

Acknowledgement

It is our proud privilege and duty to acknowledge the kind of help and guidance received from several people in preparation of this report. It would not have been possible to prepare this report in this form without their valuable help, cooperation and guidance.

First and foremost, we wish to record our sincere gratitude to Prof., **Mr** for his constant support and encouragement in preparation of this report and for making available library and laboratory facilities needed to prepare this report.

The seminar on **“Python”** was very helpful to us in giving the necessary background information and inspiration in choosing this topic for the seminar. Their contributions and technical support in preparing this report are greatly acknowledged.

Last but not the least, we wish to thank our parents for financing our studies in this college as well as for constantly encouraging us to learn engineering. Their personal sacrifice in providing this opportunity to learn engineering is gratefully acknowledgement.

Table Of Contents

Python

Python is a widely used high-level, general-purpose, interpreted, dynamic programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java. The language provides constructs intended to enable clear programs on both a small and large scale.

Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library. Python interpreters are available for installation on many operating systems, allowing Python code execution on a wide variety of systems.

Scripting Language

A scripting or script language is a programming language that supports scripts, programs written for a special run-time environment that automate the execution of tasks that could alternatively be executed one-by-one by a human operator.

Scripting languages are often interpreted (rather than compiled). Primitives are usually the elementary tasks or API calls, and the language allows them to be combined into more complex programs. Environments that can be automated through scripting include software applications, web pages within a web browser, the shells of operating systems (OS), embedded systems, as well as numerous games.

A scripting language can be viewed as a domain-specific language for a particular environment; in the case of scripting an application, this is also known as an **extension language**. Scripting languages are also sometimes referred to as very high-level programming languages, as they operate at a high level of abstraction, or as control languages.

Object Oriented Programming Language

Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which may contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods. A distinguishing feature of objects is that an object's procedures can access and often modify the data fields of the object with which they are associated (objects have a notion of "this" or "self").

In OO programming, computer programs are designed by making them out of objects that interact with one another. There is significant diversity in objectoriented programming, but most popular languages are class-based, meaning that objects are instances of classes, which typically also determines their type.

History

Python was conceived in the late 1980s, and its implementation was started in December 1989 by Guido van Rossum at CWI in the Netherlands as a successor to the ABC language (itself inspired by SETL) capable of exception handling and interfacing with the Amoeba operating system. Van Rossum is Python's principal author, and his continuing central role in deciding the direction of Python is reflected in the title given to him by the Python community, benevolent dictator for life (BDFL).

“Python is an experiment in how much freedom programmers need. Too much freedom and nobody can read another's code; too little and expressiveness is endangered.”

- Guido van Rossum

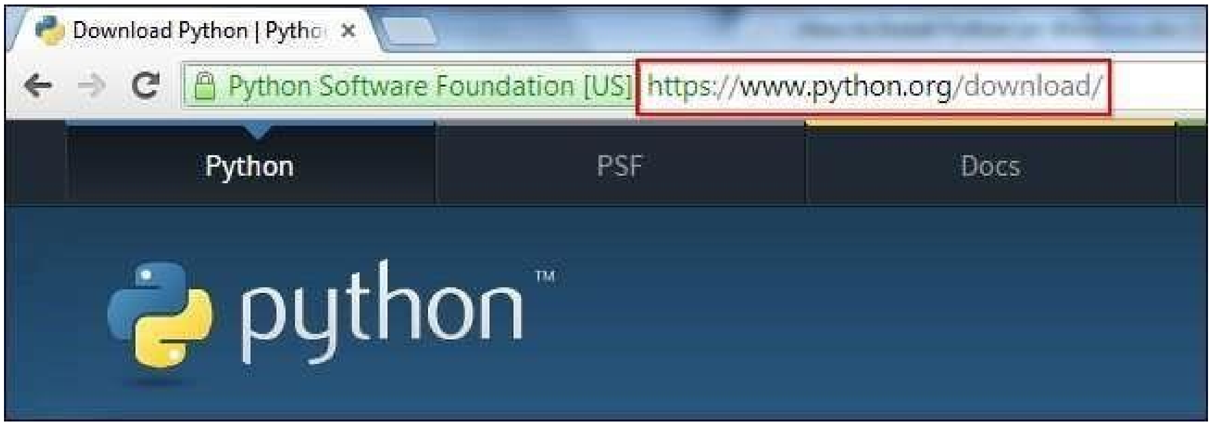
Behind The Scene of Python

About the origin of Python, Van Rossum wrote in 1996:

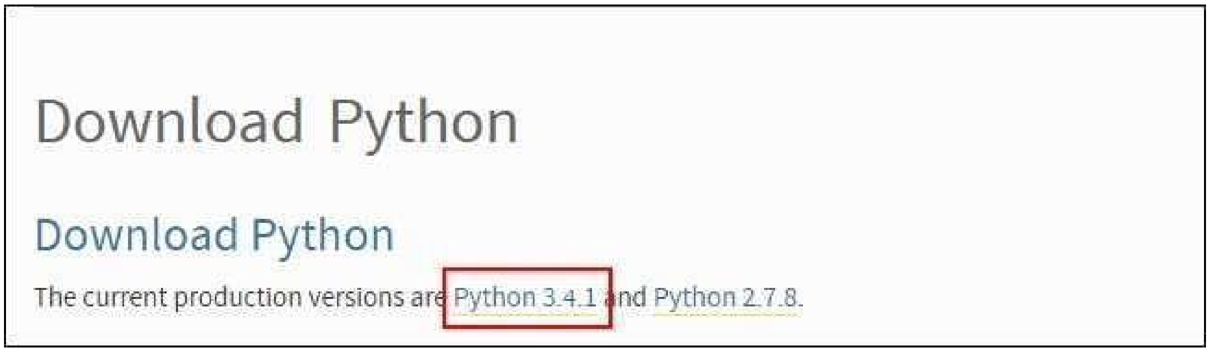
Over six years ago, in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office ... would be closed, but I had a home Computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a **descendant of ABC** that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of **Monty Python's Flying Circus**).

Downloading python

If you don't already have a copy of Python installed on your computer, you will need to open up your Internet browser and go to the Python download page ***(http://www.python.org/download/).***


Download Python | Pytho
Python Software Foundation [US]
https://www.python.org/download/
Python
PSF
Docs
python™

Now that you are on the download page, select which of the software builds you would like to download. For the purposes of this article we will use the most up to date version available (Python 3.4.1).

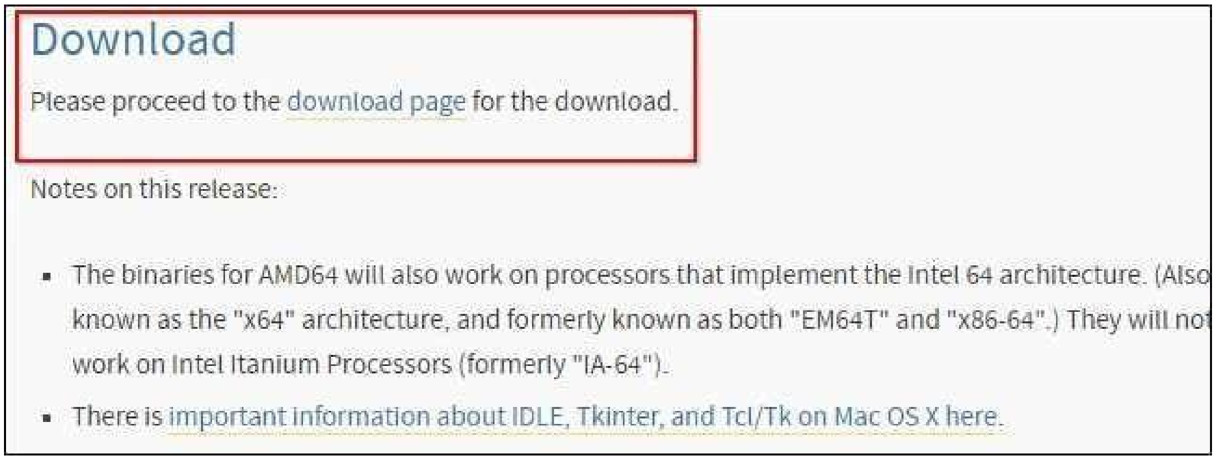


Download Python

Download Python

The current production versions are Python 3.4.1 and Python 2.7.8.

Once you have clicked on that, you will be taken to a page with a description of all the new updates and features of 3.4.1, however, you can always read that while the download is in process. Scroll to the bottom of the page till you find the "Download" section and click on the link that says "download page."



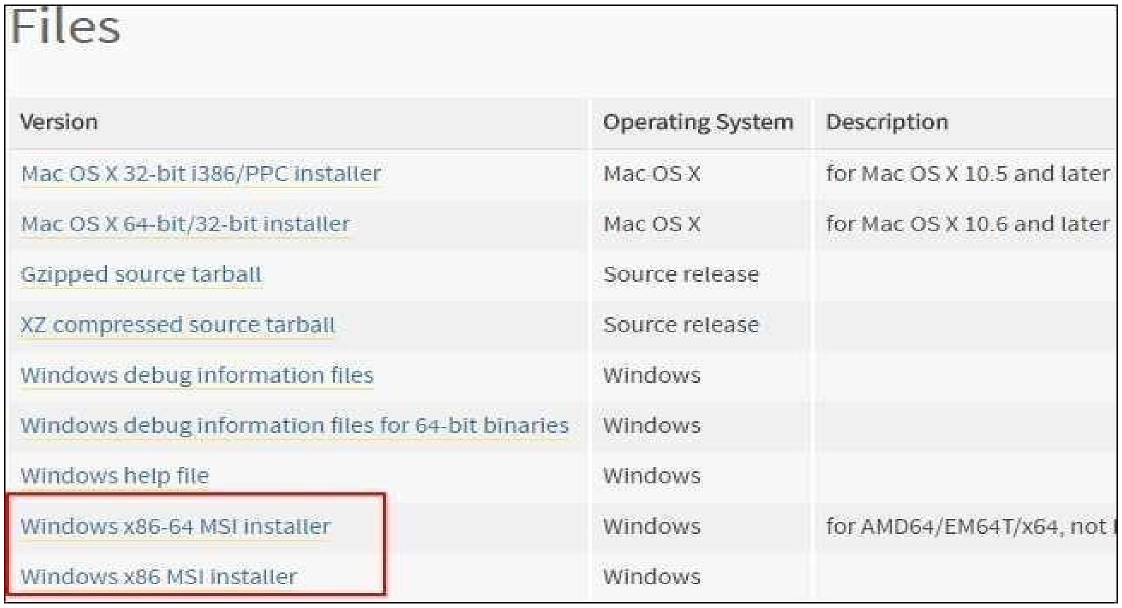
Download

Please proceed to the download page for the download.

Notes on this release:

- The binaries for AMD64 will also work on processors that implement the Intel 64 architecture. (Also known as the "x64" architecture, and formerly known as both "EM64T" and "x86-64".) They will not work on Intel Itanium Processors (formerly "IA-64").
- There is important information about IDLE, Tkinter, and Tcl/Tk on Mac OS X here.

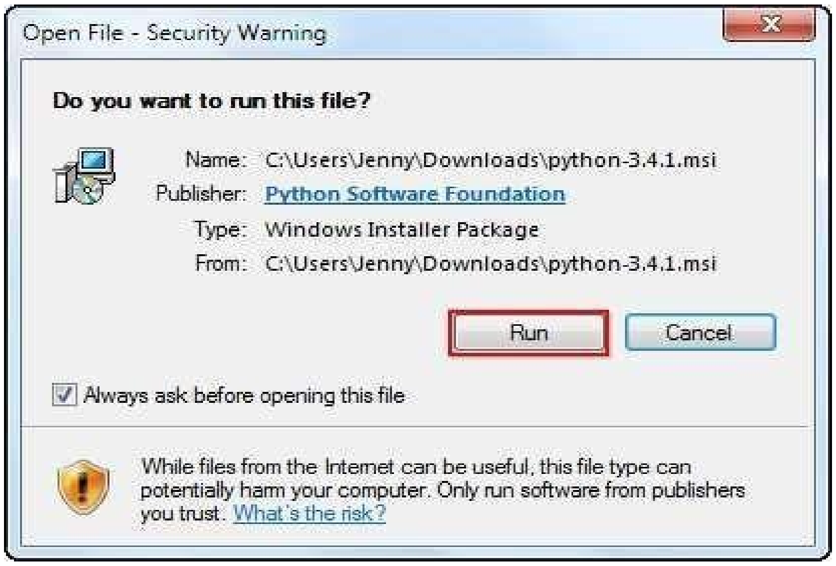
Now you will scroll all the way to the bottom of the page and find the "Windows x86 MSI installer." If you want to download the 86-64 bit MSI, feel free to do so. We believe that even if you have a 64-bit operating system installed on your computer, the 86-bit MSI is preferable. We say this because it will still run well and sometimes, with the 64- bit architectures, some of the compiled binaries and Python libraries don't work well.


Files

Version	Operating System	Description
Mac OS X 32-bit i386/PPC installer	Mac OS X	for Mac OS X 10.5 and later
Mac OS X 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later
Gzipped source tarball	Source release	
XZ compressed source tarball	Source release	
Windows debug information files	Windows	
Windows debug information files for 64-bit binaries	Windows	
Windows help file	Windows	
Windows x86-64 MSI installer	Windows	for AMD64/EM64T/x64, not
Windows x86 MSI installer	Windows	

Installing Python

Once you have downloaded the Python MSI, simply navigate to the download location on your computer, double clicking the file and pressing Run when the dialog box pops up.

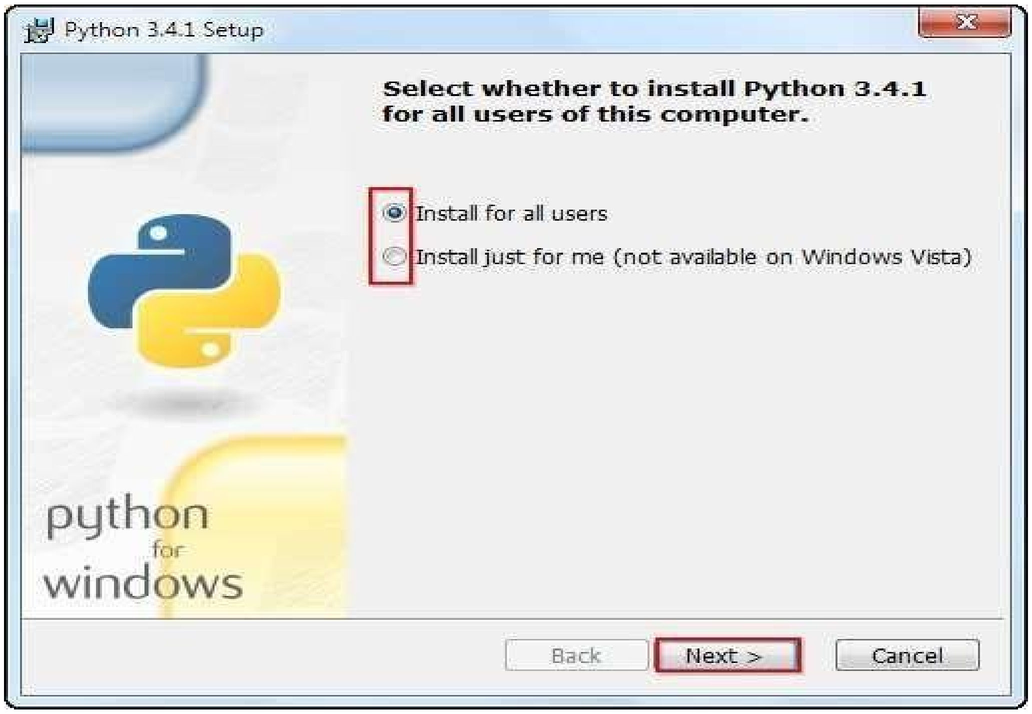

Open File - Security Warning

Do you want to run this file?

Name: C:\Users\Jenny\Downloads\python-3.4.1.msi
Publisher: Python Software Foundation
Type: Windows Installer Package
From: C:\Users\Jenny\Downloads\python-3.4.1.msi

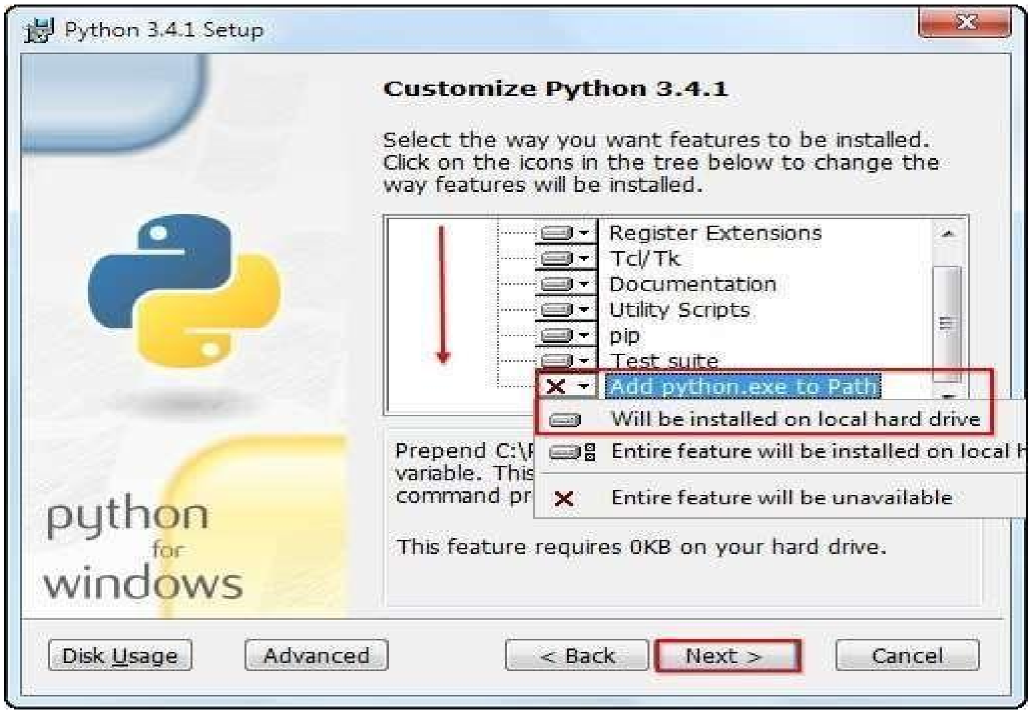
While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust. What's the risk?

If you are the only person who uses your computer, simply leave the “Install for all users” option selected. If you have multiple accounts on your PC and don’t want to install it across all accounts, select the “Install just for me” option then press “Next.”

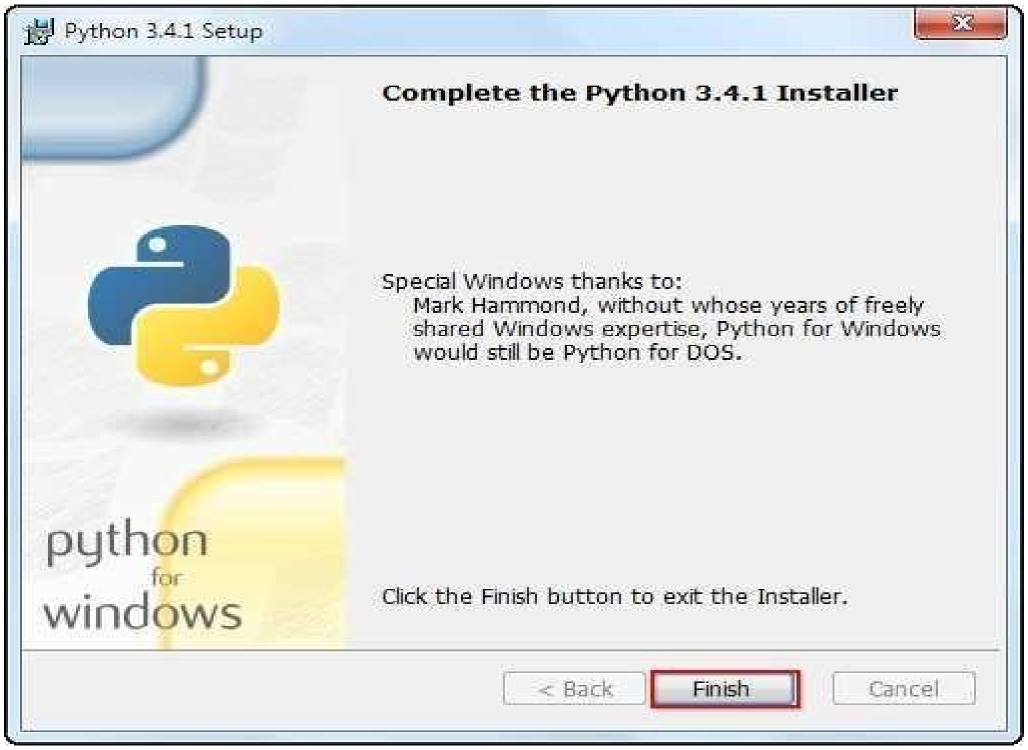

Python 3.4.1 Setup
Select whether to install Python 3.4.1 for all users of this computer.
Install for all users
Install just for me (not available on Windows Vista)
python for windows
Back
Next >
Cancel

f you want to change the install location, feel free to do so; however, it is best to leave it as is and simply select next, Otherwise...

Scroll down in the window and find the “Add Python.exe to Path” and click on the small red “x.” Choose the “Will be installed on local hard drive” option then press “Next.”


Python 3.4.1 Setup
Customize Python 3.4.1
Select the way you want features to be installed. Click on the icons in the tree below to change the way features will be installed.
Register Extensions
Tcl/Tk
Documentation
Utility Scripts
pip
Test suite
Add python.exe to Path
Will be installed on local hard drive
Entire feature will be installed on local h
Entire feature will be unavailable
Prepend C:\P
variable. This
command pr
This feature requires 0KB on your hard drive.
python for windows
Disk Usage
Advanced
< Back
Next >
Cancel

Now that you have completed the installation process, click on “Finish.



Complete the Python 3.4.1 Installer

Special Windows thanks to:
Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

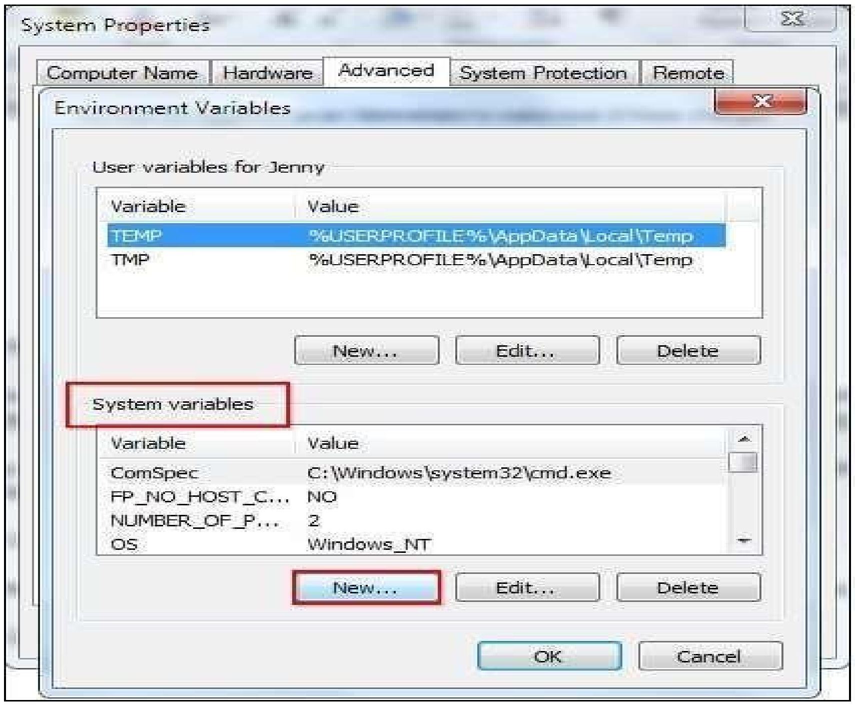
Click the Finish button to exit the Installer.

Setup the Path Variable

Begin by opening the start menu and typing in “environment” and select the option called “Edit the system environment variables.”

When the “System Properties” window appears, click on “Environment Variables…”

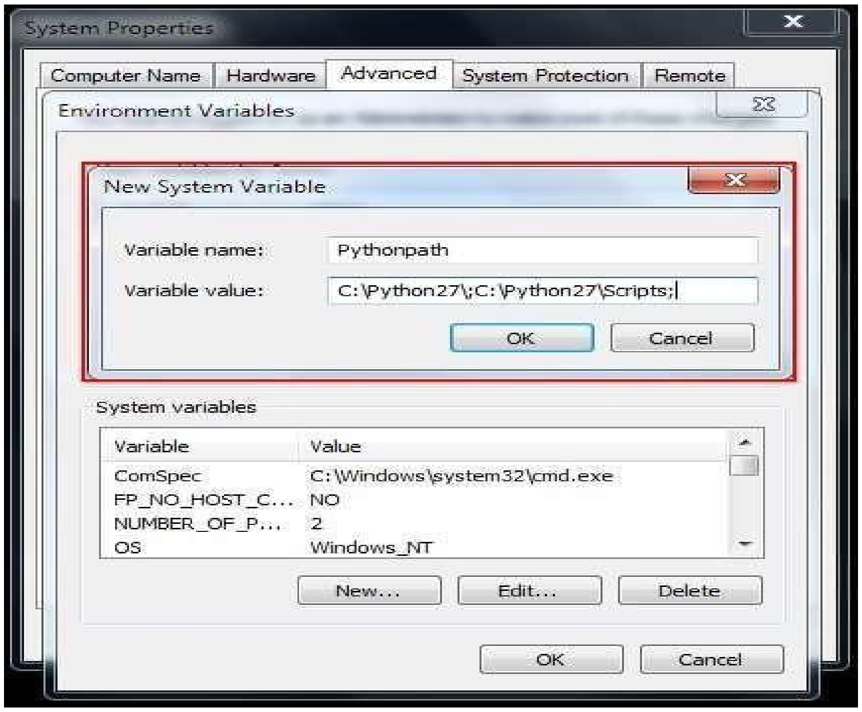
Once you have the “Environment Variables” window open, direct your focus to the bottom half. You will notice that it controls all the “System Variables” rather than just this associated with your user. Click on “New…” to create a new variable for Python.



Variable	Value
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

Variable	Value
ComSpec	C:\Windows\system32\cmd.exe
FP_NO_HOST_C...	NO
NUMBER_OF_P...	2
OS	Windows_NT

Simply enter a name for your Path and the code shown below. For the purposes of this example we have installed Python 2.7.3, so we will call the path: "Pythonpath." The string that you will need to enter is: "C:\Python27\;C:\Python27\Scripts;"



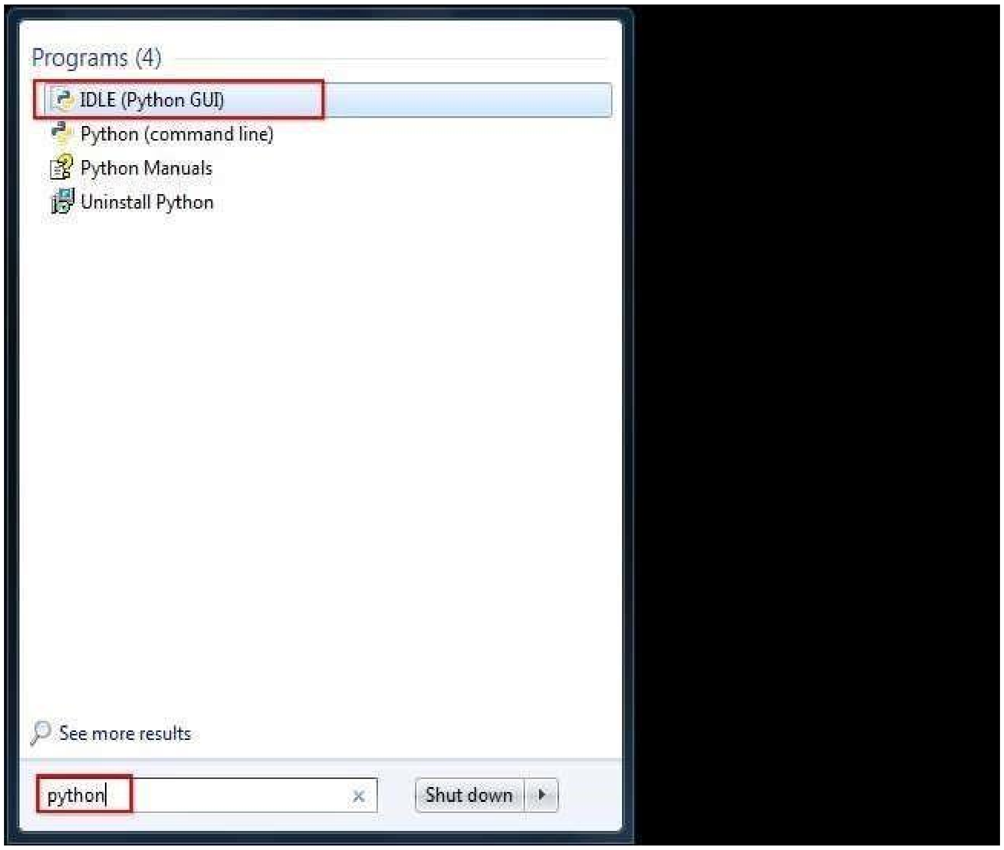
Variable name: Pythonpath

Variable value: C:\Python27\;C:\Python27\Scripts;

Variable	Value
ComSpec	C:\Windows\system32\cmd.exe
FP_NO_HOST_C...	NO
NUMBER_OF_P...	2
OS	Windows_NT

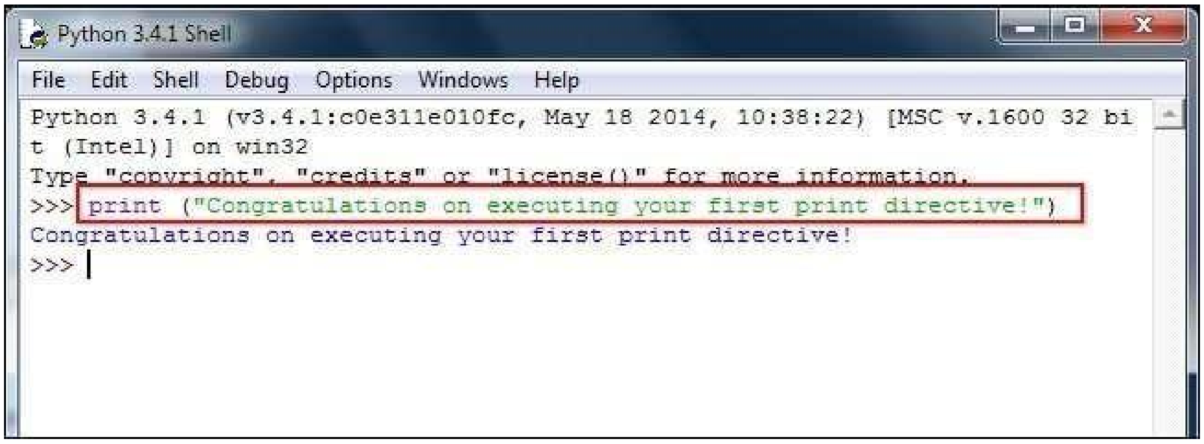
Running The Python IDE

Now that we have successfully completed the installation process and added our "Environment Variable," you are ready to create your first basic Python script. Let's begin by opening Python's GUI by pressing "Start" and typing "Python" and selecting the "IDLE (Python GUI)."



Once the GUI is open, we will begin by using the simplest directive possible. This is the "print" directive which simply prints whatever you tell it to, into a new line. Start by typing a print directive like the one shown in the image below or copy and paste this text then press

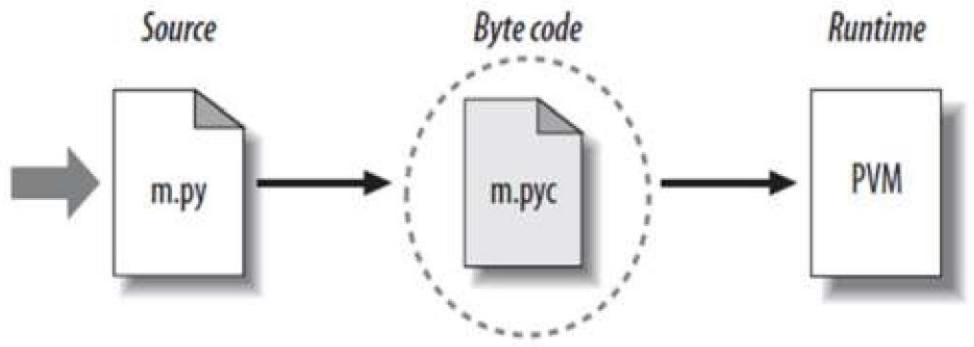
"Enter": print ("Congratulations on executing your first print directive!")



```
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:38:22) [MSC v.1600 32 bi
t (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print ("Congratulations on executing your first print directive!")
Congratulations on executing your first print directive!
>>> 
```

Python Code Execution

Python's traditional runtime execution model: source code you type is translated to byte code, which is then run by the Python Virtual Machine. Your code is automatically compiled, but then it is interpreted.



Source code extension is .py

Byte code extension is .pyc (compiled python code)

Data Type

(this is called dynamic typing). Data types determine whether an object can do something, or whether it just would not make sense. Other programming languages often determine whether an operation makes sense for an object by making sure the object can never be stored somewhere where the operation will be performed on the object (this type system is called static typing). Python does not do that. Instead it stores the type of an object with the object, and checks when the operation is performed whether that operation makes sense for that object

Python has many native data types. Here are the important ones:

Booleans are either True or False.

Numbers can be integers (1 and 2), floats (1.1 and 1.2), fractions (1/2 and 2/3), or even complex numbers.

Strings are sequences of Unicode characters, e.g. an HTML document.

Bytes and byte arrays, e.g. a JPEG image file.

Lists are ordered sequences of values.

Tuples are ordered, immutable sequences of values.

Sets are unordered bags of values.

Variable

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

```
Ex: counter = 100                    # An integer

assignment miles  = 1000.0           # A floating

point name  = "John"                 # A string
```

String

In programming terms, we usually call text a string. When you think of a string as a collection of letters, the term makes sense.

All the letters, numbers, and symbols in this book could be a

string. For that matter, your name could be a string, and so could

your address.

Creating Strings

In Python, we create a string by putting quotes around text. For example, we could take our otherwise useless

- "hello"+"world"   "helloworld"   # concatenation

• "hello"*3	"hellohellohello"	# repetition
• "hello"[0]	"h"	# indexing
• "hello"[-1]	"o"	# (from end)
• "hello"[1:4]	"ell"	# slicing
• len("hello")	5	# size
• "hello" < "jello"	1	# comparison
• "e" in "hello"	1	# search

Python Operator

Arithmetic Operator

Operator	Meaning	Example
+	Add two operands or unary plus	x + y +2
-	Subtract right operand from the left or unary minus	x - y -2
*	Multiply two operands	x * y
/	Divide left operand by the right one (always results into float)	x / y

%	Modulus - remainder of the division of left operand by the right	x % y (remainder of x/y)
//	Floor division - division that results into whole number adjusted to the left in the number line	x // y
**	Exponent - left operand raised to the power of right	x**y (x to the power y)

Comparison Operator

Cha

Tuples

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses.

Accessing Values in Tuples:

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example − tup1 = ('physics', 'chemistry', 1997, 2000); tup2 = (1, 2, 3, 4, 5, 6, 7); print "tup1[0]: ", tup1[0] print "tup2[1:5]: ", tup2[1:5]

When the above code is executed, it produces the following result − tup1[0]: physics tup2[1:5]: [2, 3, 4, 5]

Basic Tuples Operations

Tuples respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new tuple, not a string. In fact, tuples respond to all of the general sequence operations we used on strings in the prior chapter −

Python Expression	**Results**	**Description**
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1, 2, 3): print x,	1 2 3	Iteration

Built-in Tuple Functions

Python includes the following tuple functions −

SN	Function with Description
1	cmp(tuple1, tuple2) Compares elements of both tuples.
2	len(tuple) Gives the total length of the tuple.
3	max(tuple) Returns item from the tuple with max value.
4	min(tuple) Returns item from the tuple with min value.
5	tuple(seq) Converts a list into tuple.

List

The list is a most versatile datatype available in Python which can be written as a list of comma- separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.

Creating a list is as simple as putting different comma-separated values between square brackets. For example − list1 = ['physics', 'chemistry', 1997, 2000]; list2 = [1, 2, 3, 4, 5]; list3 = ["a", "b", "c", "d"];

Similar to string indices, list indices start at 0, and lists can be sliced, concatenated and so on.

Accessing Values in Lists:

To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example − list1 = ['physics', 'chemistry', 1997, 2000]; list2 = [1, 2, 3, 4, 5, 6, 7]; print "list1[0]: ", list1[0] print "list2[1:5]: ", list2[1:5]

Output: list1[0]: physics

list2[1:5]: [2, 3, 4, 5]

Update: list = ['physics', 'chemistry', 1997, 2000]; print

"Value available at index 2 : " print list[2] list[2] = 2001; print

"New value available at index 2 : " print list[2]

Output: Value available at index 2 :

1997 New value available at index 2 :
2001

Delete: list1 = ['physics', 'chemistry', 1997, 2000]; print

list1 del list1[2]; print "After deleting value at index 2 : " print

list1

['physics', 'chemistry', 1997, 2000]

Output: After deleting value at index 2 :

['physics', 'chemistry', 2000]

Basic List Operation

Python Expression	**Results**	**Description**
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership
for x in [1, 2, 3]: print x,	1 2 3	Iteration

Built-in List Functions & Methods:

SN	Function with Description
1	**cmp(list1, list2)** Compares elements of both lists.
2	**len(list)** Gives the total length of the list.
3	**max(list)** Returns item from the list with max value.
4	**min(list)** Returns item from the list with min value.
5	**list(seq)** Converts a tuple into list.

Python includes following list methods

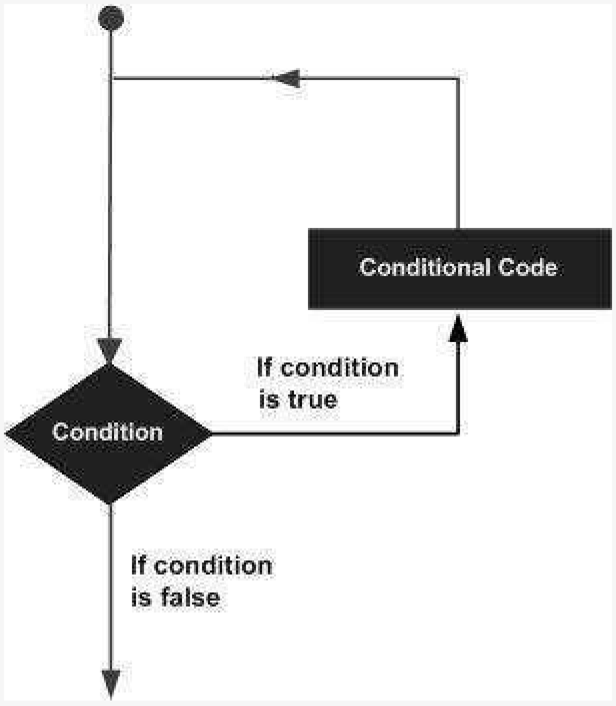
SN	Methods with Description
1	**list.append(obj)** Appends object obj to list
2	**list.count(obj)** Returns count of how many times obj occurs in list
3	**list.extend(seq)** Appends the contents of seq to list
4	**list.index(obj)** Returns the lowest index in list that obj appears
5	**list.insert(index, obj)** Inserts object obj into list at offset index
6	**list.pop(obj=list[-1])** Removes and returns last object or obj from list

7	**list.remove(obj)** Removes object obj from list
8	**list.reverse()** Reverses objects of list in place
9	**list.sort([func])** Sorts objects of list, use compare func if given

Loop definition

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement −

	 Conditional Code If condition is true Condition If condition is false	
>	Greater that - True if left operand is greater than the right	x > y
<	Less that - True if left operand is less than the right	x < y
==	Equal to - True if both operands are equal	x == y

!=	Not equal to - True if operands are not equal	x != y
>=	Greater than or equal to - True if left operand is greater than or equal to the right	x >= y
<=	Less than or equal to - True if left operand is less than or equal to the right	+x <= y

Python programming language provides following types of loops to handle looping requirements.

Loop Type	Description

while loop	Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
nested loops	You can use one or more loop inside any another while, for or do..while loop.

Loop Example:

For Loop:

```
>>> for mynum in [1, 2, 3, 4, 5]:

print ("Hello", mynum )

Hello 1

Hello 2

Hello 3

Hello 4

Hello 5
```

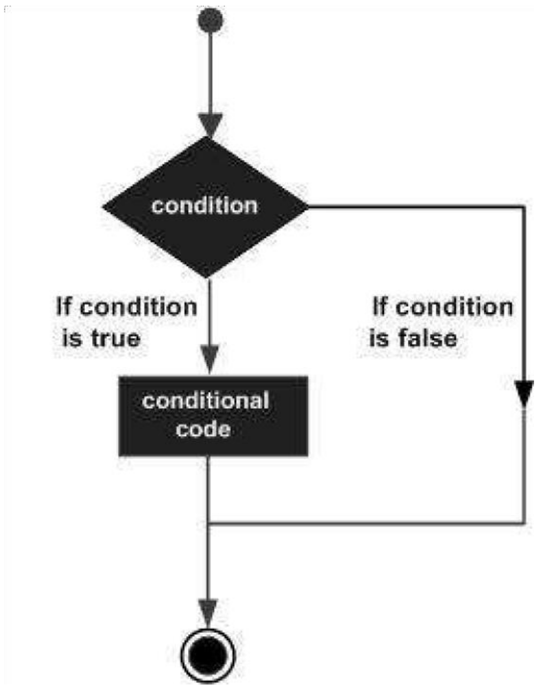
While Loop:

```
>>> count = 0 >>while(count< 4):

print 'The count is:', count count =
count + 1

The count is: 0

The  count  is:  1

The  count  is:  2

The count is: 3
```

Conditional Statements:

Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions.

Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome. You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise.



Python programming language provides following types of decision making statements. Click the following links to check their detail.

Statement	**Description**
[if statements]()	An **if statement** consists of a boolean expression followed by one or more statements.
[if...else statements]()	An **if statement** can be followed by an optional **else statement**, which executes when the boolean expression is FALSE.
[nested if statements]()	You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).

Example:

If Statement:

```
a=33

b=200

If b>a:

print(“b”)
```

If...Else Statement:

```
a=200

b=33

if b>a:

     print(“b is greater than a”)

else:

   print(“a is greater than b”)
```

Function

Function blocks begin with the keyword **def** followed by the function name and parentheses (( )).

Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
The first statement of a function can be an optional statement - the documentation string of the function.

The code block within every function starts with a colon (:) and is indented.

The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Syntex:

```
Def functionname(parameters):

“function_docstring”

Function_suite

Return[expression]
```

Example:

```
Def printme(str):
```

“this print a passed string into this

function” print str

return

1. *# Function definition is here* def
   printme(str):

   "This prints a passed string into this function" print
   str return;

   # Now you can call printme function printme("I'm first call to user defined function!") printme("Again second call to the same function")

SCOPE OF PYTHON

1 - Science

- Bioinformatics

2 - System Administration

- Unix
- Web logic
- Web sphere

3 - Web Application Development

What Can We do With Python?

1 - System programming

2 - Graphical User Interface

Programming 3 - Internet Scripting

4 - Component Integration

5 - Database Programming

6 - Gaming, Images, XML , Robot and more

WHO USES PYTHON TODAY?

- Python is being applied in real revenue-generating products by real companies.
- Google makes extensive use of Python in its web search system, and employs Python's creator.
- Intel, Cisco, Hewlett-Packard, Seagate, Qualcomm, and IBM use Python for hardware testing.
- ESRI uses Python as an end-user customization tool for its popular GIS mapping products.

WHY DO PEOPLE USE PYTHON?

- The YouTube video sharing service is largely written in Python.
- Python is object-oriented ○ Structure supports such concepts as polymorphism, operation overloading, and multiple inheritance.
- Indentation ○ Indentation is one of the greatest future in Python.
- It's free (open source) ○ Downloading and installing Python is free and easy ○ Source code is easily accessible
- It's powerful ○ Dynamic typing ○ Built-in types and tools ○ Library utilities

  ○ Third party utilities (e.g. Numeric, NumPy, SciPy) ○ Automatic memory management
- It's portable ○ Python runs virtually every major platform used today ○ As long as you have a compatible Python interpreter installed, Python programs will run in exactly the same manner, irrespective of platform.

Conclusion

I believe the trial has shown conclusively that it is both possible and desirable to use Python as the principal teaching language:

- It is Free (as in both cost and source code).
- It is trivial to install on a Windows PC allowing students to take their interest further. For many the hurdle of installing a Pascal or C compiler on a Windows machine is either too expensive or too complicated;
- It is a flexible tool that allows both the teaching of traditional procedural programming and modern OOP; It can be used to teach a large number of transferable skills;
- It is a real-world programming language that can be and is used in academia and the commercial world;
- It appears to be quicker to learn and, in combination with its many libraries, this offers the possibility of more rapid student development allowing the course to be made more challenging and varied;

and most importantly, its clean syntax offers increased understanding and enjoyment for students