UNIT I

Syllabus :-

Introduction: Introduction to Python, Program Development Cycle, Input, Processing, and Output, Displaying Output with the Print Function, Comments, Variables, Reading Input from the Keyboard, Performing Calculations, Operators. Type conversions, Expressions, More about Data Output. Data Types, and Expression: Strings Assignment, and Comment, Numeric Data Types and Character Sets, Using functions and Modules. Decision Structures and Boolean Logic: if, if-else, if-elif-else Statements, Nested Decision Structures, Comparing Strings, Logical Operators, Boolean Variables. Repetition Structures: Introduction, while loop, for loop, Calculating a Running Total, Input Validation Loops, Nested Loops.

Introduction to Python

History of Python :-

BCPL (Basic combined programming language), it was developed by martin Richard. BCPL lead to B. B was developed by ken Thompson. BCPL and B are type less programming language. B lead to C. C was developed by Dennis Ritchie. C is a typed programming language. C leads to C++. C++ was developed by Bjarne Stroustrup. C++ is a component dependent programming language. C++ leads to OAK. OAK was developed by James Gosling. OAK is a component independent programming language. OAK language leads to python. Python was developed by Guido Van Rossum. Python is a cross platform programming language.

What is python :-

- ❖ Python is an object oriented programming language.
- ❖ Python is a scripting language.
- ❖ It is a general purpose programming language.
- ❖ It is a cross platform programming language.
- ❖ It is an interactive programming language.
- ❖ Python is a Beginner's Language.
- ❖ It was developed by Guido Van Rossum.
- ❖ It was developed in 1991.
- ❖ It was developed at Netherlands.

Features of Python:

Python's Features Include:

- ❖ **Easy-to-Learn**: Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- ❖ **Easy-to-read**: Python code is more clearly defined and visible to eyes.
- ❖ **Easy-to-maintain**: Python's source code is fairly easy-to-maintain.
- ❖ **A broad standard library**: Python's bulk of the library is very portable and cross-platform compatible on UNIX, and Windows.
- ❖ **Interactive Mode**: Python has support for an interactive mode which allows interactive testing and debugging of code.
- ❖ **Portable**: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- ❖ **Extendable**: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- ❖ **Databases**: Python provides interfaces to all major commercial databases.

- **GUI Programming**: Python supports GUI applications that can be created and ported to many system calls, libraries, and windows system, such as windows MFC, Macintosh, and the X Window system of UNIX.
- **Scalable**: Python provides a better structure and support for large programs than shell scripting.
- **Dynamically Typed**: We need to specify the data type of variable while creation. Python can understand the data type of variable after storing the value. It is called dynamically typed.

Limitation of python :-

1. **Slow**:- When compare to C and C++, python runs a little bit of slow because of dynamic dynamically typed.
2. **Mobile Application Development**:- python is very suitable for server side programming. Python is not suitable for mobile application development.
3. **Database Applications**:- Python does not provides interfaces to communicate with all the databases. The popular technologies like ODBC, JDBC python is still under development. So python does not support to all types of databases.
4. **Memory consumption**:- Memory consumption in python is high because of dynamically typed.
5. **Run Time Errors**:- Python program need not to be compiled. It can be directly run. So all the errors will be occurred at run time. So developers need more concentration on program.
6. **Simplicity**: - python is very simple to learn and use. So python developers adopted to the python code.

Flavors of python :-

Python is available in different flavors.

- Jpython :- it was developed using java.
- Cpython :- it was developed using C
- Iron python :- it was developed in C#.
- PyPy :- it was developed using python.
- Ruby python :- it was developed for communicate with ruby interpreter.
- Brython :- it was developed for browsers.
- Micro Python :- it was developed to control micro controllers.
- Anakonda Python :- it is a high end python software.

Versions of python :-

Python is available in following versions.

- Python 1.0
- Python 2.0
- Python 3.0
- Python 3.4
- Python 3.5
- Python 3.6
- Python 3.7
- Python 3.8
- Python 3.9

Who use python :-

- **Google** use python in web searching services.
- **Youtube** uses python in video sharing services.
- **Bittorrentz** uses python in file sharing services.

- **NASA, Los Alamos, Fermilab, JPL** uses python in scientific application development.
- **Intel, Cisco, Hewlett-Packard, Seagate, Qualcomm, and IBM** uses python in hardware testing.
- **Maya, 3D animation** uses python in scripting coding.
- The **Dropbox storage service** codes both its server and desktop client software primarily in Python.
- The **Raspberry Pi** single-board computer promotes Python as its educational language.

Applications of pythons :-

We can build following type of applications using python.

- Web or Internet applications.
- Desktop application / stand alone applications.
- Scientific application.
- Systems Programming.
- Educational Applications.
- Business Application.
- Database applications.

Need of Python Programming

- **Software quality**

  Python code is designed to be readable, and hence reusable and maintainable—much more so than traditional scripting languages. The uniformity of Python code makes it easy to understand, even if you did not write it. In addition, Python has deep support for more advanced software reuse mechanisms, such as object-oriented (OO) and function programming.

- **Developer productivity**

  Python boosts developer productivity many times beyond compiled or statically typed languages such as C, C++, and Java. Python code is typically one-third to less to debug, and less to maintain after the fact. Python programs also run immediately, without the lengthy compile and link steps required by some other tools, further boosting programmer speed. Program portability Most Python programs run unchanged on all major computer platforms. Porting Python code between Linux and Windows, for example, is usually just a matter of copying a script‘s code between machines.

- **Support libraries**

  Python comes with a large collection of prebuilt and portable functionality, known as the standard library. This library supports an array of application-level programming tasks, from text pattern matching to network scripting. In addition, Python can be extended with both home grown libraries and a vast collection of third-party application support software. Python‘s third-party domain offers tools for website construction, numeric programming, serial port access, game development, and much more (see ahead for a sampling).

- **Component integration**

  Python scripts can easily communicate with other parts of an application, using a variety of integration mechanisms. Such integrations allow Python to be used as a product customization and extension tool. Today, Python code can invoke C and C++ libraries, can be called from C and C++ programs, can integrate with Java and .NET components, can communicate over frameworks such as COM and Silverlight, can interface with devices over serial ports, and can interact over networks with interfaces like SOAP, XML-RPC, and CORBA. It is not a standalone tool.

- **Enjoyment**

  Because of Python‘s ease of use and built-in toolset, it can make the act of programming more pleasure than chore. Although this may be an intangible benefit, its effect on productivity is an important asset. Of these factors, the first two (quality and productivity) are probably the most compelling benefits to most Python users, and merit a fuller description.

- **It's Object-Oriented**

  Python is an object-oriented language, from the ground up. Its class model supports advanced notions such as polymorphism, operator overloading, and multiple inheritance; yet in the context of Python's dynamic typing, object-oriented programming (OOP) is remarkably easy to apply. Python's OOP nature makes it ideal as a scripting tool for object-oriented systems languages such as C++ and Java. For example, Python programs can subclass (specialized) classes implemented in C++ or Java.

- **It's Free**

  Python is freeware—something which has lately been come to be called open source software. As with Tcl and Perl, you can get the entire system for free over the Internet. There are no restrictions on copying it, embedding it in your systems, or shipping it with your products. In fact, you can even sell Python, if you're so inclined. But don't get the wrong idea: "free" doesn't mean "unsupported". On the contrary, the Python online community responds to user queries with a speed that most commercial software vendors would do well to notice.

- **It's Portable**

  Python is written in portable ANSI C, and compiles and runs on virtually every major platform in use today. For example, it runs on UNIX systems, Linux, MS-DOS, MS-Windows (95, 98, NT), Macintosh, Amiga, Be-OS, OS/2, VMS, QNX, and more. Further, Python programs are automatically compiled to portable bytecode, which runs the same on any platform with a compatible version of Python installed (more on this in the section "It's easy to use"). What that means is that Python programs that use the core language run the same on UNIX, MS-Windows, and any other system with a Python interpreter.

- **It's Powerful**

  From a features perspective, Python is something of a hybrid. Its tool set places it between traditional scripting languages (such as Tcl, Scheme, and Perl), and systems languages (such as C, C++, and Java). Python provides all the simplicity and ease of use of a scripting language, along with more advanced programming tools typically found in systems development languages.

- **Automatic memory management**

  Python automatically allocates and reclaims ("garbage collects") objects when no longer used, and most grow and shrink on demand; Python, not you, keeps track of low-level memory details.

- **Programming-in-the-large support**

  Finally, for building larger systems, Python includes tools such as modules, classes, and exceptions; they allow you to organize systems into components, do OOP, and handle events gracefully.

- **It's Mixable**

  Python programs can be easily "glued" to components written in other languages. In technical terms, by employing the Python/C integration APIs, Python programs can be both extended by (called to) components written in C or C++, and embedded in (called by) C or C++ programs. That means you can add functionality to the Python system as needed and use Python programs within other environments or systems.

- **It's Easy to Use**

  For many, Python's combination of rapid turnaround and language simplicity make programming more fun than work. To run a Python program, you simply type it and run it. There are no intermediate compile and link steps (as when using languages such as C or C++). As with other interpreted languages, Python executes programs immediately, which makes for both an interactive programming experience and rapid turnaround after program changes. Strictly speaking, Python programs are compiled (translated) to an intermediate form called bytecode, which is then run by the interpreter.

- **It's Easy to Learn**

  This brings us to the topic of this book: compared to other programming languages, the core Python language is amazingly easy to learn. In fact In fact, you can expect to be coding significant Python programs in a matter of days (and perhaps in just hours, if you're already an experienced programmer).

- **Internet Scripting**

  Python comes with standard Internet utility modules that allow Python programs to communicate over sockets, extract form information sent to a server-side CGI script, parse HTML, transfer files by FTP, process XML files, and much more. There are also a number of peripheral tools for doing Internet programming in Python. For instance, the HTMLGen and pythondoc systems generate HTML files from Python class-based descriptions, and the JPython system mentioned above provides for seamless Python/Java integration.

- **Database Programming**

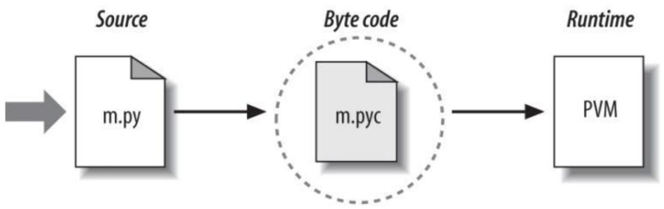
  Python's standard pickle module provides a simple object-persistence system: it allows programs to easily save and restore entire Python objects to files. For more traditional database demands, there are Python interfaces to Sybase, Oracle, Informix, ODBC, and more. There is even a portable SQL database API for Python that runs the same on a variety of underlying database systems, and a system named gadfly that implements an SQL database for Python programs.

❖ **Image Processing, AI, Distributed Objects, Etc.**

Python is commonly applied in more domains than can be mentioned here. But in general, many are just instances of Python's component integration role in action. By adding Python as a frontend to libraries of components written in a compiled language such as C, Python becomes useful for scripting in a variety of domains. For instance, image processing for Python is implemented as a set of library components implemented in a compiled language such as C, along with a Python frontend layer on top used to configure and launch the compiled components.

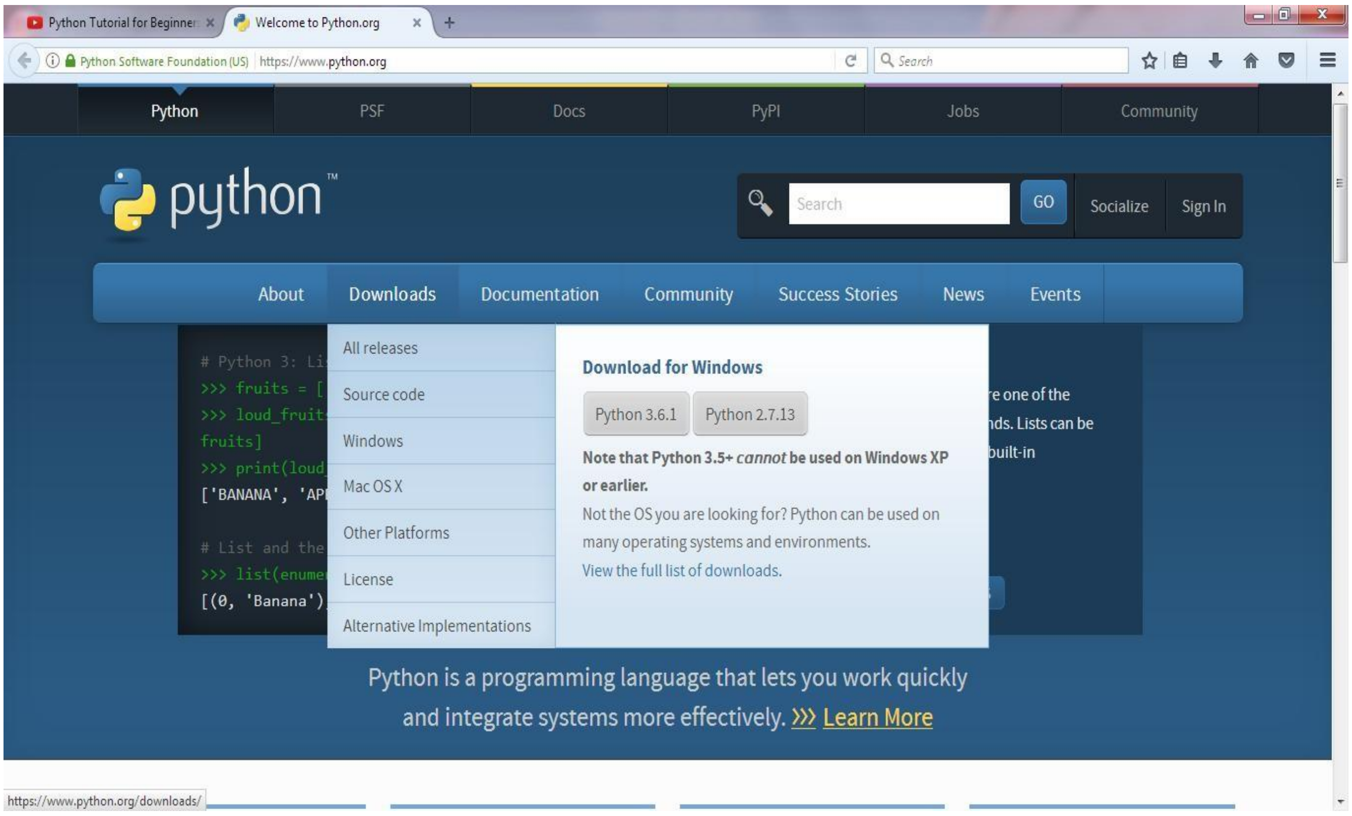
PVM :-

After interpretation the python source code called ".py" file gets converted into Python Byte Code called ".PyC" file(python class file). Byte code is machine independent. It can not be understand by any machine. It can be understood by a special software called python virtual machine(PVM). It always resides in computer memory. pVM can do what a machine can do so PVM is called virtual machine.



Download and installation of python software

Step 1: Go to website www.python.org and click downloads select version which you want.

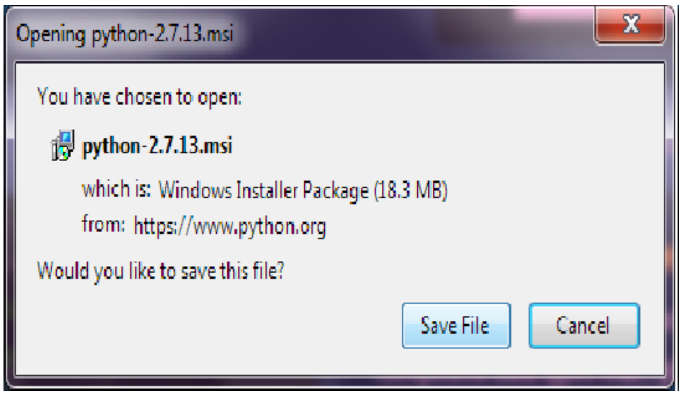


Note that Python 3.5+ *cannot* be used on Windows XP or earlier.

Not the OS you are looking for? Python can be used on many operating systems and environments.

Python is a programming language that lets you work quickly and integrate systems more effectively.

Step 2: Click on **Python 2.7.13** and download. After download open the file.



Opening python-2.7.13.msi

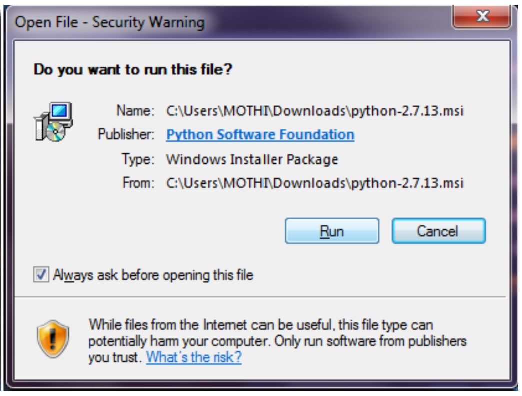
You have chosen to open:

python-2.7.13.msi

which is: Windows Installer Package (18.3 MB)

from: https://www.python.org

Would you like to save this file?



Open File - Security Warning

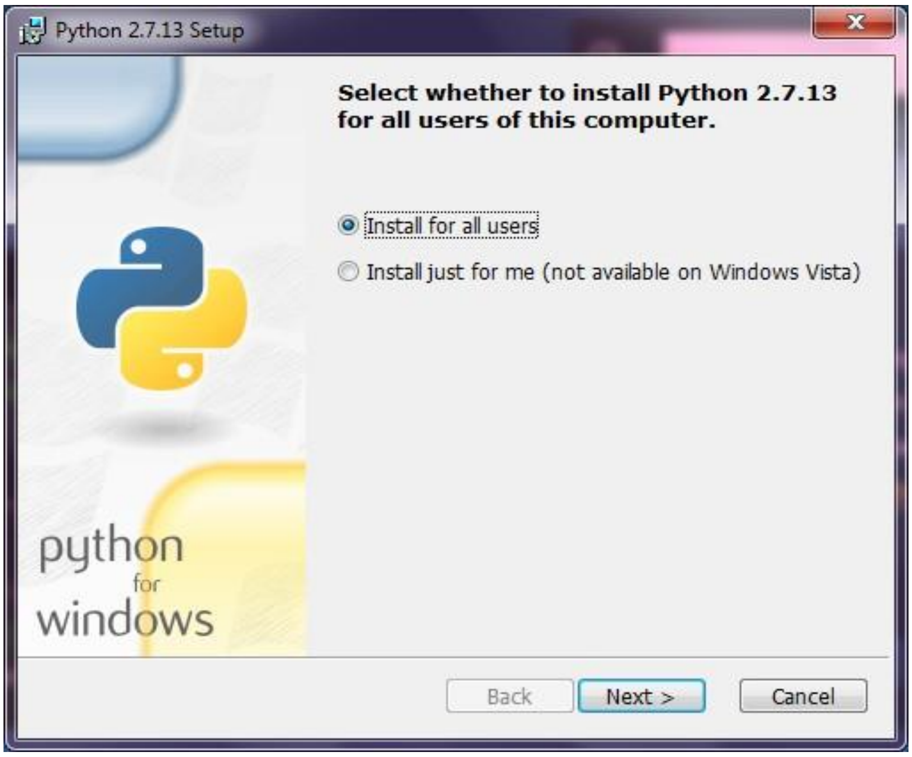
Do you want to run this file?

Name: C:\Users\MOTHI\Downloads\python-2.7.13.msi
Publisher: Python Software Foundation
Type: Windows Installer Package
From: C:\Users\MOTHI\Downloads\python-2.7.13.msi

[✓] Always ask before opening this file

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?

Step 3: Click on **Next** to continue.



Python 2.7.13 Setup

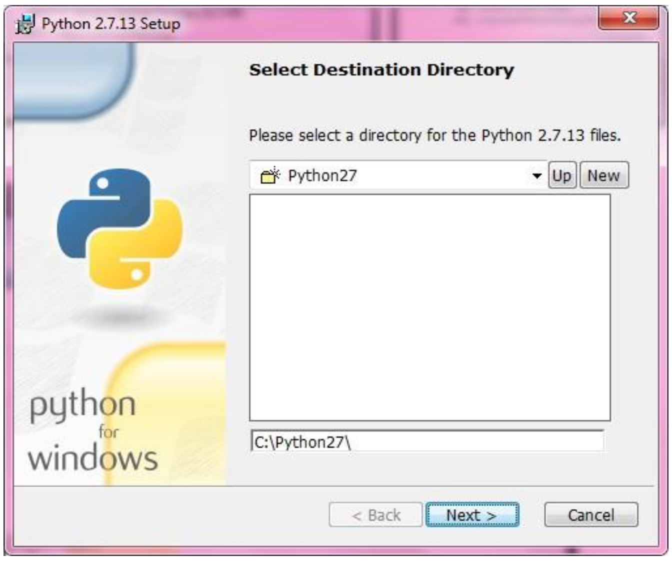
Select whether to install Python 2.7.13 for all users of this computer.

[✓ Install for all users]

Install just for me (not available on Windows Vista)

Step 4: After installation location will be displayed. The Default location is **C:\Python27.**

Click on next to continue.



Python 2.7.13 Setup

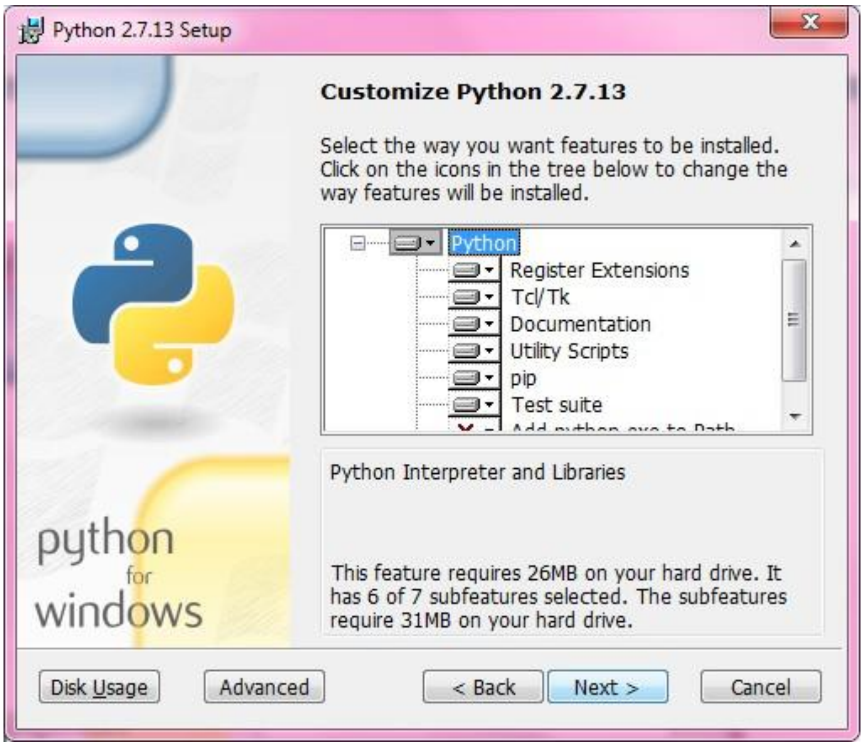
Select Destination Directory

Please select a directory for the Python 2.7.13 files.

Python27

C:\Python27\

Step 5: After the python interpreter and libraries are displayed for installation. Click on Next to continue.



Python 2.7.13 Setup

Customize Python 2.7.13

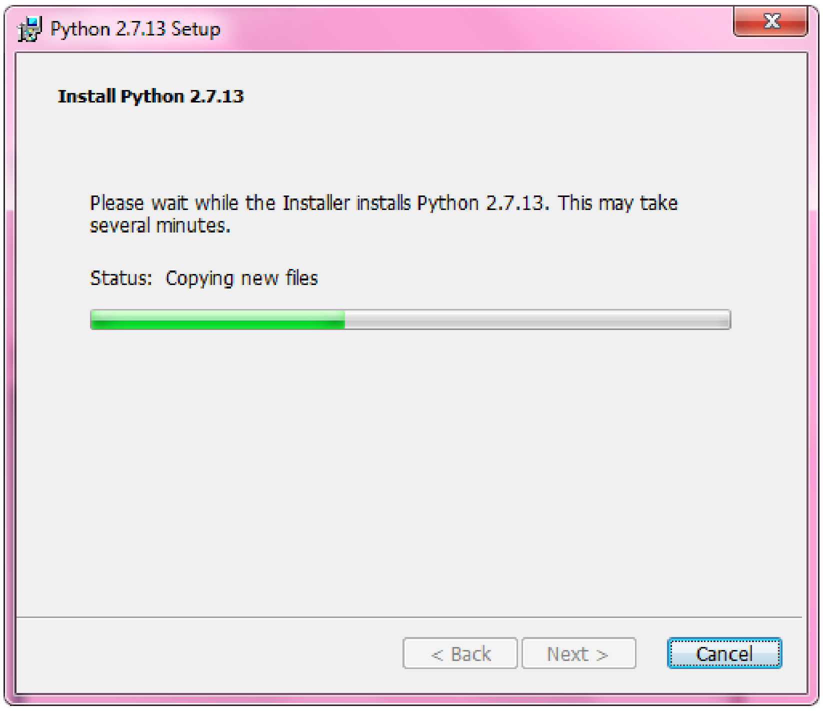
Select the way you want features to be installed. Click on the icons in the tree below to change the way features will be installed.

Python
- Register Extensions
- Tcl/Tk
- Documentation
- Utility Scripts
- pip
- Test suite

Python Interpreter and Libraries

This feature requires 26MB on your hard drive. It has 6 of 7 subfeatures selected. The subfeatures require 31MB on your hard drive.

Step 6: The installation has been processed.



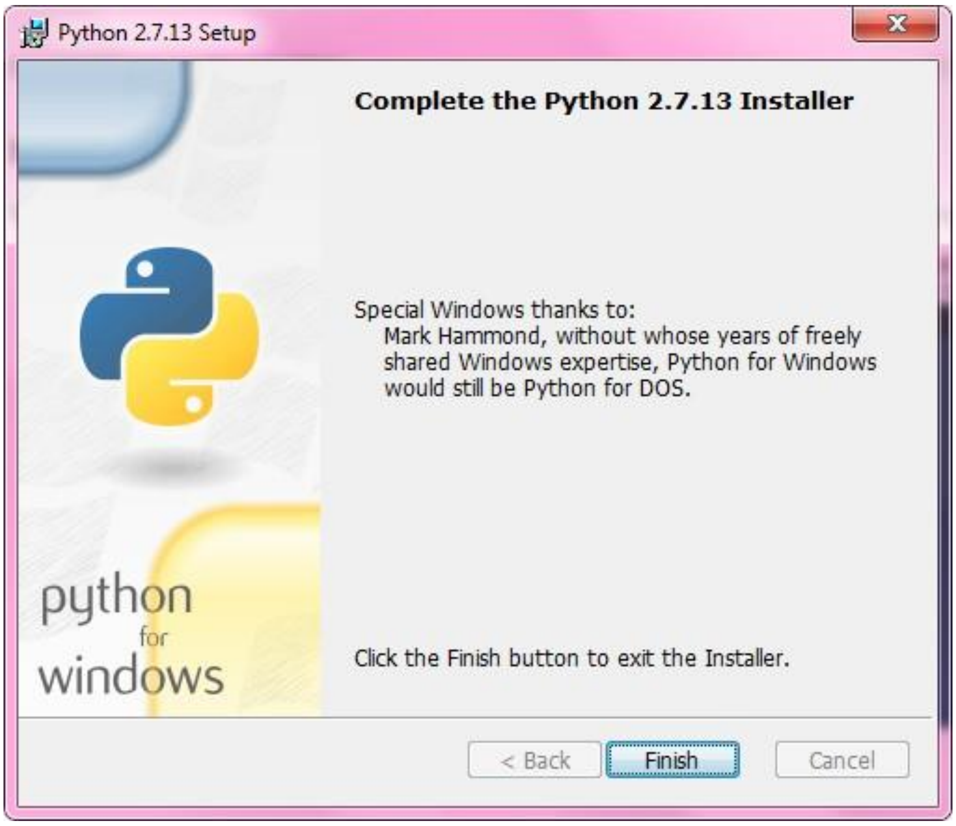
Python 2.7.13 Setup

Install Python 2.7.13

Please wait while the Installer installs Python 2.7.13. This may take several minutes.

Status: Copying new files

Step 7: Click the **Finish** to complete the installation.


Python 2.7.13 Setup

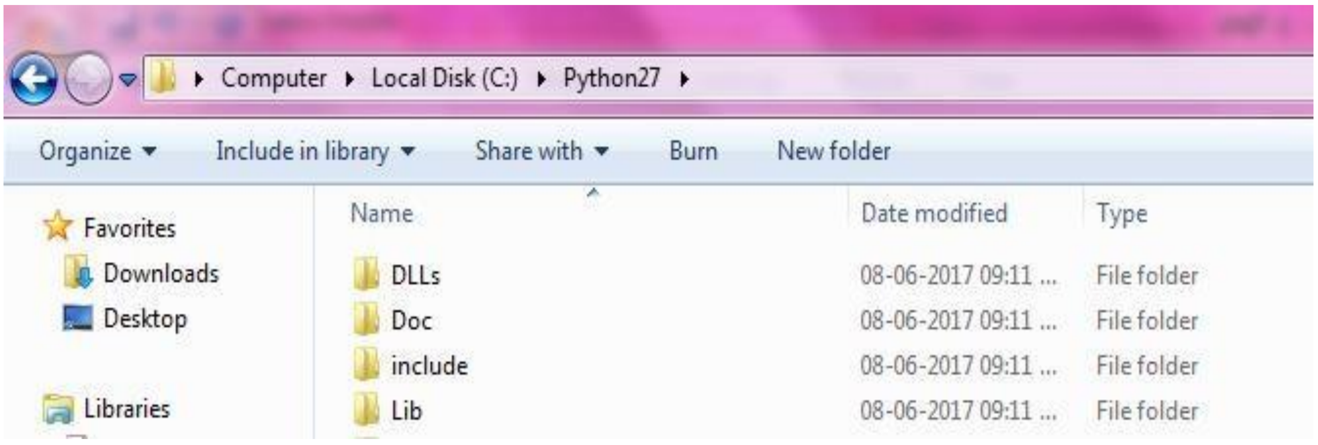
Complete the Python 2.7.13 Installer

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.

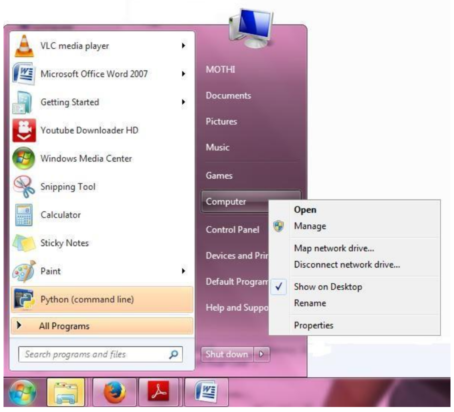
Setting up PATH to python

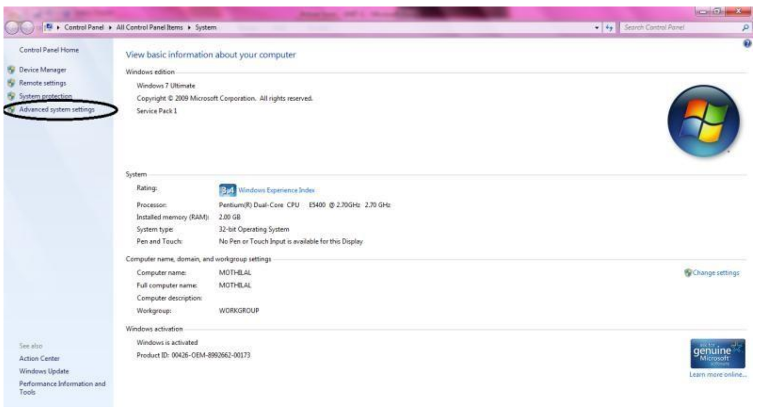
- Programs and other executable files can be in many directories, so operating systems provide a search path that lists the directories that the OS searches for executables.
- The path is stored in an environment variable, which is a named string maintained by the operating system. This variable contains information available to the command shell and other programs.
- Copy the Python installation location C:\Python27


Computer ▸ Local Disk (C:) ▸ Python27 ▸

Name	Date modified	Type
DLLs	08-06-2017 09:11 ...	File folder
Doc	08-06-2017 09:11 ...	File folder
include	08-06-2017 09:11 ...	File folder
Lib	08-06-2017 09:11 ...	File folder

- Right-click the My Computer icon on your desktop and choose **Properties**. And then select **Advanced System properties.**





View basic information about your computer

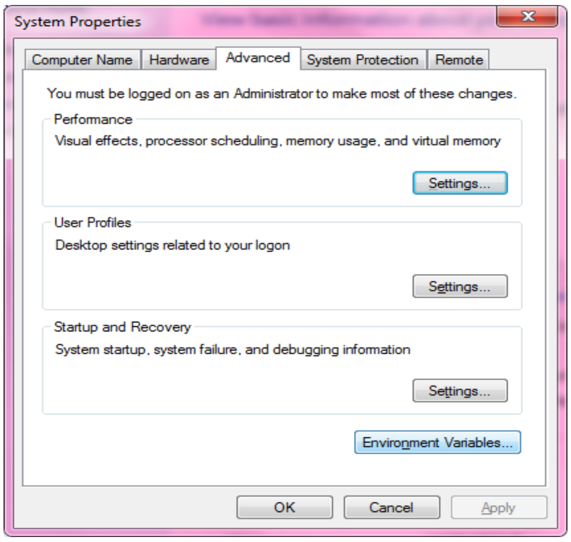
Windows edition
Windows 7 Ultimate
Copyright © 2009 Microsoft Corporation. All rights reserved.
Service Pack 1

System
Rating: 3.4 Windows Experience Index
Processor: Pentium(R) Dual-Core CPU E5400 @ 2.70GHz 2.70 GHz
Installed memory (RAM): 2.00 GB
System type: 32-bit Operating System
Pen and Touch: No Pen or Touch Input is available for this Display

Computer name, domain, and workgroup settings
Computer name: MOTHILAL
Full computer name: MOTHILAL
Computer description:
Workgroup: WORKGROUP

Windows activation
Windows is activated
Product ID: 00426-OEM-8992662-00173

➢ Goto **Environment Variables** and go to **System Variables** select **Path** and click on **Edit.**

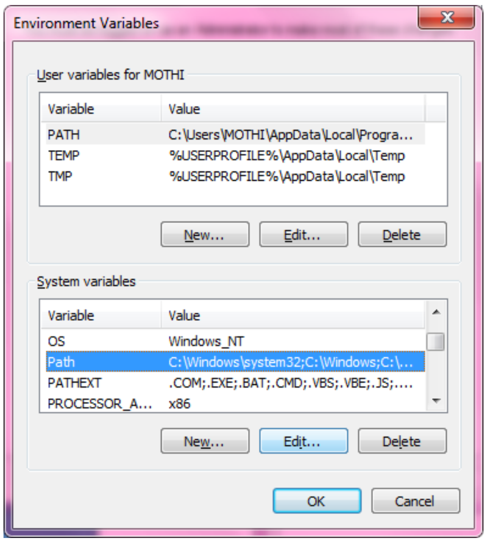


You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles
Desktop settings related to your logon

Startup and Recovery
System startup, system failure, and debugging information



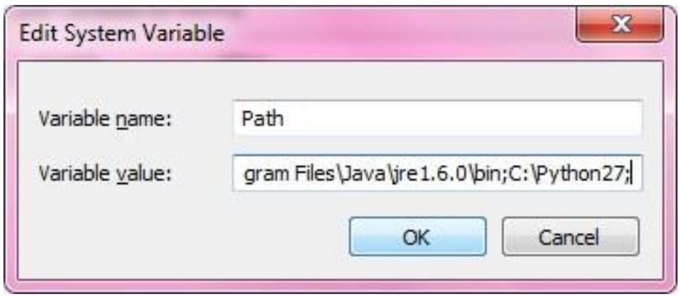
User variables for MOTHI

Variable	Value
PATH	C:\Users\MOTHI\AppData\Local\Progra...
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

System variables

Variable	Value
OS	Windows_NT
Path	C:\Windows\system32;C:\Windows;C:\...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....
PROCESSOR_A...	x86

➢ Add semicolon (;) at end and copy the location **C:\Python27** and give semicolon (;) and click OK.



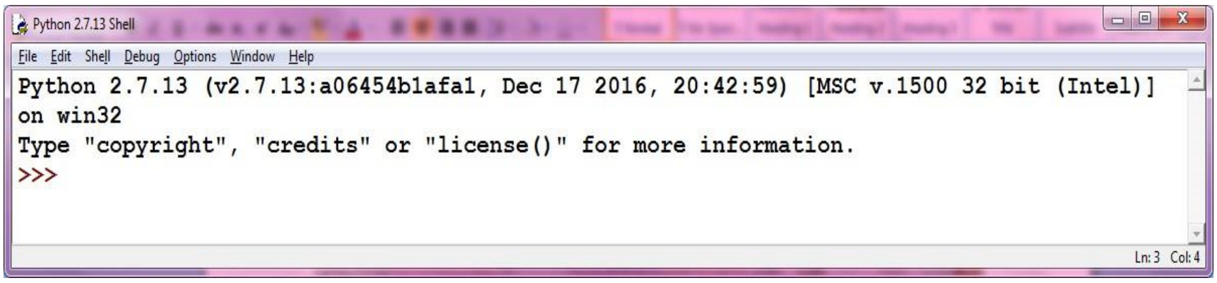
Variable name: Path

Variable value: gram Files\Java\jre1.6.0\bin;C:\Python27;

Running Python

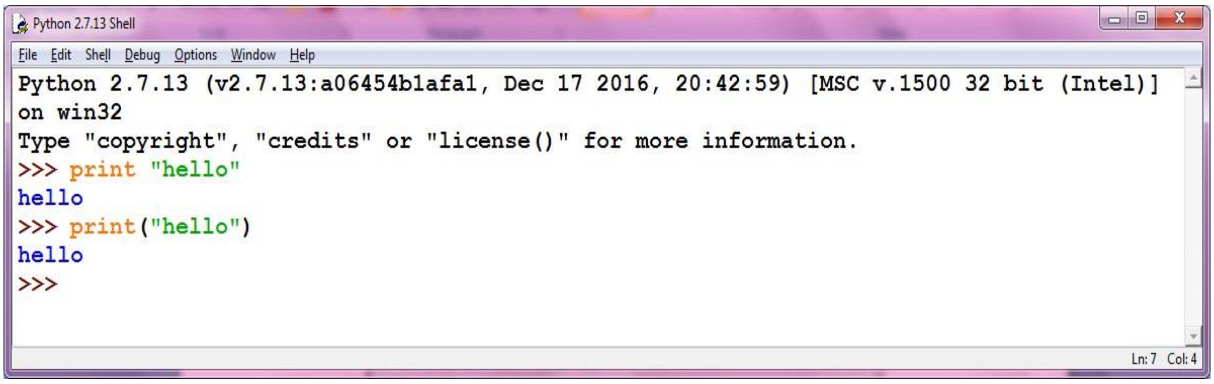
a. Running Python Interpreter:

Python comes with an interactive interpreter. When you type python in your shell or command prompt, the python interpreter becomes active with a >>> prompt and waits for your commands.



```
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:42:59) [MSC v.1500 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

Now you can type any valid python expression at the prompt. Python reads the typed expression, evaluates it and prints the result.



```
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:42:59) [MSC v.1500 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> print "hello"
hello
>>> print("hello")
hello
>>>
```

Running Python Scripts in IDLE:

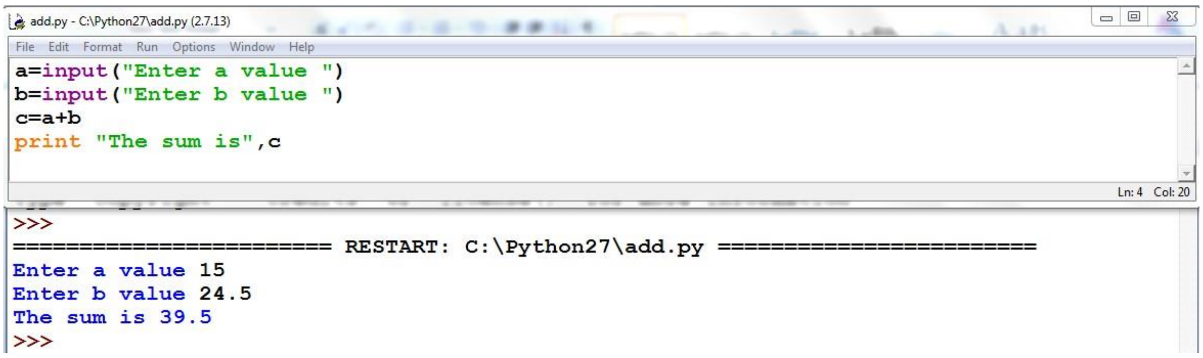
- Goto **File** menu click on New File (CTRL+N) and write the code and save add.py

  a=input("Enter a value ")

  b=input("Enter b value ")c=a+b

  print "The sum is",c

- And run the program by pressing F5 or Run→Run Module.


add.py - C:\Python27\add.py (2.7.13)
File Edit Format Run Options Window Help
Ln: 4 Col: 20

```
a=input("Enter a value ")
b=input("Enter b value ")
c=a+b
print "The sum is",c
```

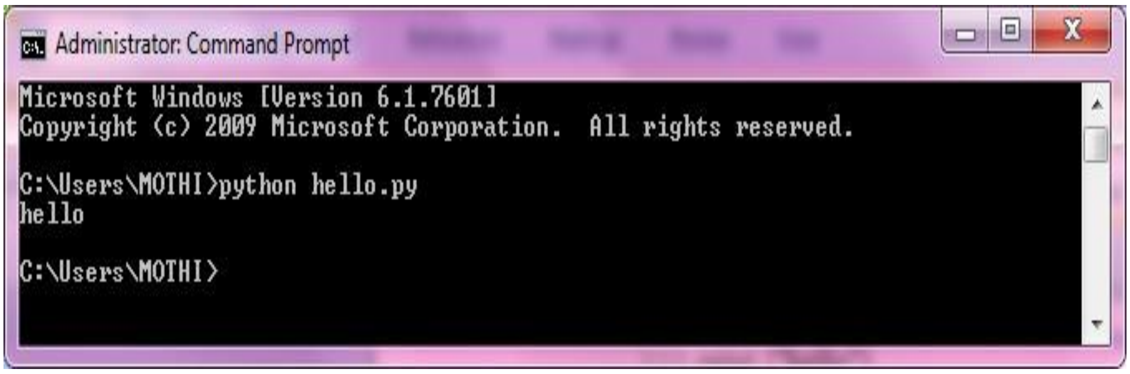
```
>>>
======================== RESTART: C:\Python27\add.py ========================
Enter a value 15
Enter b value 24.5
The sum is 39.5
>>>
```

Running python scripts in Command Prompt:

- Before going to run we have to check the PATH in environment variables.
- Open your text editor, type the following text and save it as hello.py.

  print "hello"

- And run this program by calling python hello.py. Make sure you change to the directorywhere you saved the file before doing it.



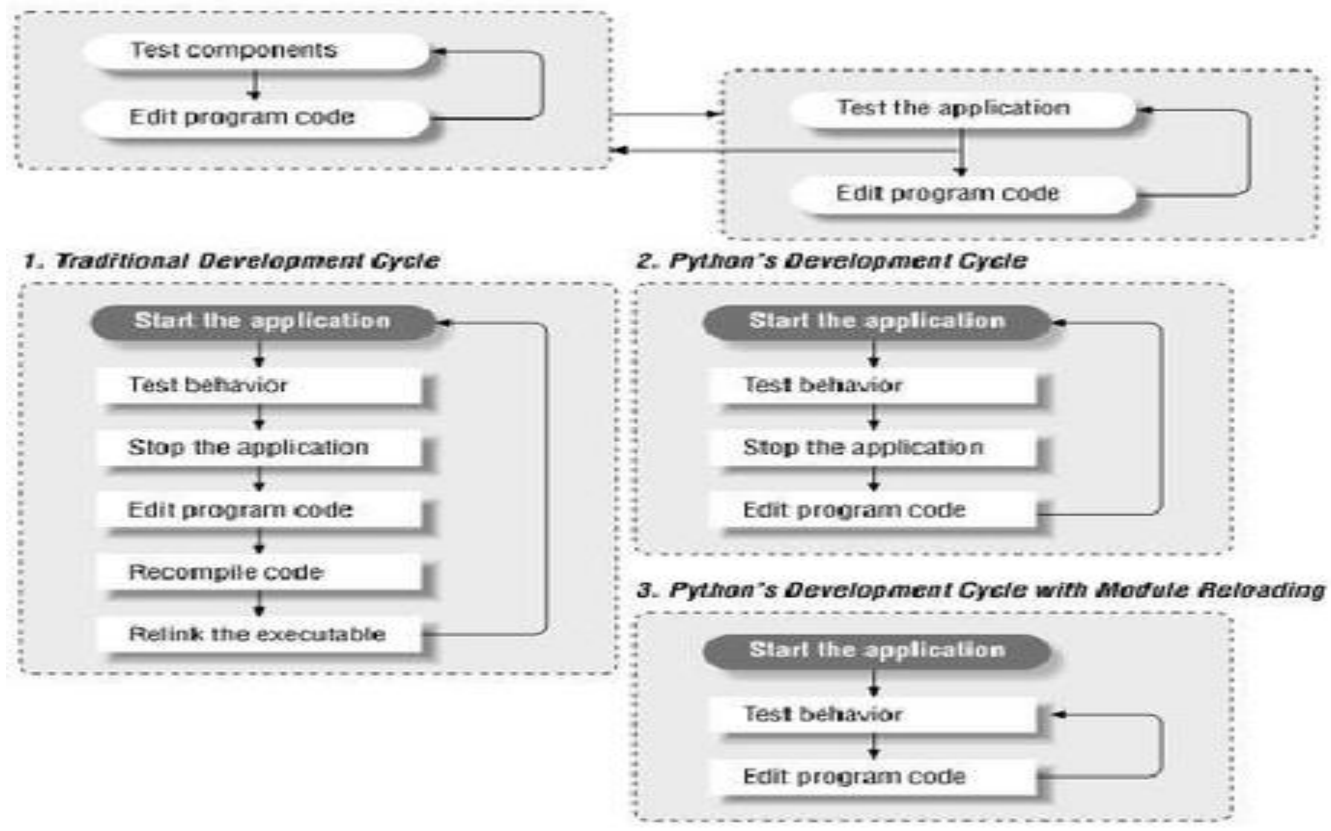
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\MOTHI>python hello.py
hello

C:\Users\MOTHI>
```

Program Development Cycle

Python's development cycle is dramatically shorter than that of traditional tools. In Python, there are no compile or link steps -- Python programs simply import modules at runtime and use the objects they contain. Because of this, Python programs run immediately after changes are made. And in cases where dynamic module reloading can be used, it's even possible to change and reload parts of a running program without stopping it at all. Figure shows Python's impact on the development cycle.



Because Python is interpreted, there's a rapid turnaround after program changes. And because Python's sparser is embedded in Python-based systems, it's easy to modify programs at runtime. For example, we saw how GUI programs developed with Python allow developers to change the code that handles a button press while the GUI remains active; the effect of the code change may be observed immediately when the button is pressed again. There's no need to stop and rebuild.

More generally, the entire development process in Python is an exercise in rapid prototyping. Python lends itself to experimental, interactive program development, and encourages developing systems incrementally by testing components in isolation and putting them together later. In fact, we've seen that we can switch from testing components (unit tests) to testing whole systems (integration tests) arbitrarily.

Input

- ❖ Input is something enters into the computer for processing.
- ❖ Input is a process of entering the data into computer.
- ❖ Input is always a raw data.
- ❖ Input data must be processed.
- ❖ Input is sent to a computer using input devices.

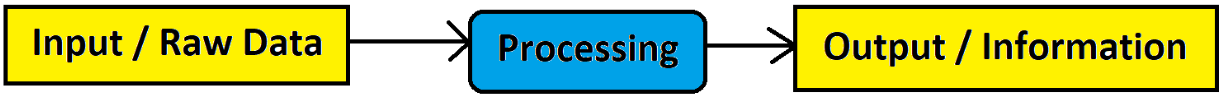
- ❖ Devices commonly used to provide input to a computer include:
  Keyboard, Mouse, Microphone, Webcam, Touchpad, Scanner

Processing

- ❖ Data processing is manipulation of data by a computer.
- ❖ Data processing is a process of translation of raw data into usable information.
- ❖ Processing includes the conversion of raw data to machine-readable form, flow of data through the CPU and memory to output devices, and formatting or transformation of output.
- ❖ Raw data must be processed to produce meaningful information.

Output

- ❖ Output is something generated after processing raw data.
- ❖ Output is data generated after processing.
- ❖ Output is meaningful information.
- ❖ Output may be text or video or audio
- ❖ Devices commonly used to present output are monitor, speakers, LCD projector.



Comments

- ❖ Comment is a message used to make the code more readable.
- ❖ Comment does not interpret.
- ❖ Comment does not convert to machine code.
- ❖ Comment does not execute.
- ❖ It is just for user purpose.
- ❖ It can also be used to prevent the execution of code:
- ❖ These are 2 types.

a. **Single line comment**: - a comment which fit in one line is called as single line comment. It is also called as line comment. It always begins with # symbol.

Example

```
#WAP to print 1-5 in words.
print("one")
#print("two")
print("three")
#print("four")
print("five")
```

output :

```
one
three
five
```

b. **Multi line comment**: - it is a comment which does not fit in one line. It wraps to multiple lines. It is called multi line comment. It is also called as block comment. It always enclosed with in ‘ ’ ’ or within “ “ “ .

Example :

```
print("one")
'''print("two")
print("three")
print("four")'''
print("five")
```

Output :

```
one
five
```

Variables

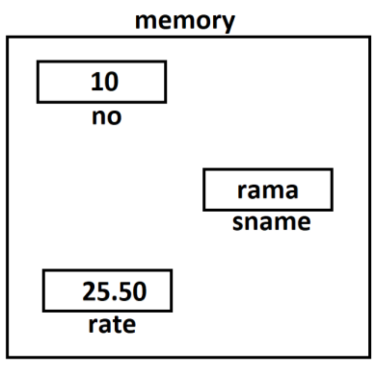
- ❖ Variable is a container.
- ❖ It is used to store the data.
- ❖ It can store only one value.
- ❖ It can vary its value so it is called variable.
- ❖ These create inside the memory.
- ❖ It must be identified by a specific name.
- ❖ It is also called as identifier.
- ❖ Variable names are case-sensitive.
- ❖ It is also called as named location in memory.

Creation / declaration of variables:-

- ❖ Variables need not need to be declared in python.
- ❖ Data type specification is not required to create a variable.
- ❖ Data type of value is the data type of variable.
- ❖ Data type of variable can be changed after they have been created.

Example :

```
no=10
sname='rama'
rate=25.50
```



Example 1 : Creating variables.

```
a=10
b=20
c=30
```

Example 2 : creating multiple variable in single line.

```
a,b,c=5,10,20
print( a,b,c )
```

output:

```
5 10 20
```

Example 3 : Python allows you to assign a single value to several variables simultaneously.

```
a=b=c=10
```

Example 4:

a=10,b=20,c=30

Example 5 create multiple variables and assign with different values

a=10;b=20;c=30

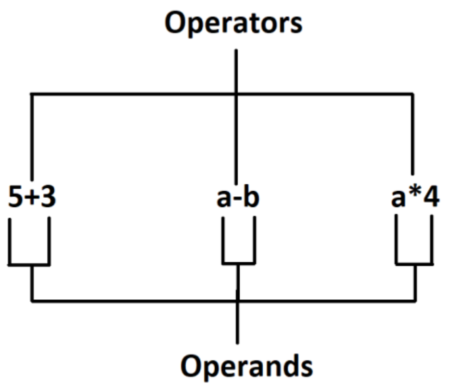
Keywords

- It is a reserved word.
- It is a pre defined identifier.
- We cannot alter the meaning of keywords.
- We cannot use a keyword as a variable name, function name or any other identifier.
- Python provides 33 keywords.

True	False	None	and	as
asset	def	class	continue	break
else	finally	elif	del	except
global	for	if	from	import
raise	try	or	return	pass
nonlocal	in	not	is	lambda

Operators

- Operators are used to perform operations on variables and values.
- Operands may be a constant or variable or both.



Operators are mainly classified into 7 types.

1) Arithmetic operators
2) Relational operators
3) Logical operators
4) Bitwise operators
5) Assignment operators
6) Conditional operators
7) Special operators.

1. **Arithmetic operators :-**

- ❖ The operators those are used to evaluate the arithmetic expressions are called arithmetic operators.
- ❖ These are used with numeric values.
- ❖ These are used to perform common mathematical operation

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
// Floor Division	The division of operands where the result isthe quotient in which the digits after the decimal point are removed.	9//2 = 4 and 9.0//2.0 = 4.0

1. **WAP to demonstrate / and // operators:**

```
a=100
r1=a/3
r2=a//3
print( r1 )
print( r2 )
output :
        33.333333
        33
```

2. **WAP to demonstrate * and ****

```
a=10
b=3
print( a*b )
print( a**b )
output :
        30
        1000
```

2. **Relational operators :-**

- ❖ The operators those are used to test the relationship between 2 values.
- ❖ The output of relational operators is always Boolean value.
- ❖ Relational operators in python work on strings also.

Operator	Description	Example
= =	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a != b) is true.
< >	If values of two operands are not equal, then condition becomes true.	(a <> b) is true. This is similar to != operator.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
> =	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
< =	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Example 1:

```
a=10
b=5
print( a<b )
print( a>b  )
print( a<=b )
print( a>=b )
print( a==b )
print( a!=b )
```

output :

```
False
True
False
True
False
True
```

Example 2:

```
str1="rama"
str2="rama"
print( str1==str2 )
print( str1<str2 )
print( str1>str2 )
print( str1<=str2 )
print( str1>=str2 )
print( str1!=str2 )
```

Output :

```
True
False
False
True
True
False
```

3. **Logical operators :-**

These works on Boolean values(T or F). Its output also a Boolean value.

Operator	Description	Example
And Logical AND	If both the operands are true then condition becomes true.	(a and b) is true.
Or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand.	Not (a and b) is false.

example :

```
print( True and True )
print( True or False )
print( not True )
```

output :

```
True
True
False
```

4. **Bitwise operators :**

The operators those works only on numbers.
These works at binary level or bit level.

Operator	Description	Example
& Binary AND	Operator copies a bit to the result if it exists in both operands.	(a & b) = 12 (means 0000 1100)
| Binary OR	It copies a bit if it exists in either operand.	(a | b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	a << 2 = 240 (means 1111 0000)
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 = 15 (means 0000 1111)

Example :

```
a=10
b=7
print( a>>2 )
print( a<<2 )
print( a&b )
print( a|b )
print( a^b )
print( ~a)
```

output :

```
2
40
2
15
13
-11
```

5. **Assignment operators :-**

The operator which is used to assign a value to a variable.

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / a
%= Modulus AND	It takes modulus using two operands andassign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculationon operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Example :

```
a=10
a+=5
print( a )
a-=3
print( a )
a*=2
print( a )
```

output :

```
15
12
24
```

6. **Conditional operators :-**

It work on given condition.
So it is called conditional operators
Example :

```
a=10
b=5
```

```
big=(b,a)[a>b]
print( big )
```
output:
```
10
```

7. **Special operators :-**

Python provides 2 types of special operators

a. Membership operators

These are again classified into 2 types.

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 ifx is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

b. Indentify operators

These are again classified into 2 types

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here is results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here is not results in 1 if id(x) is not equal to id(y).

Example 1 :
```
a=[10,20,30,40]
print( 20 in a )
output : True
```
Example 2 :
```
a=[10,20,30,40]
print( 200 in a )
output : False
```
Example 3 :
```
a=[10,20,30,40]
print( 200 not in a )
output: True
```
Example 4 :
```
a=[10,20,30,40]
print( 20 not in a )
output : False
```
Example 5 :
```
a=10
b=5
print( a is b )
output : False
```
Example 6 :
```
a=10
```

```
        b=10
        print( a is b )
        output : True
```
Example 7:
```
        a=10
        b=10
        print( a is not b )
        output : False
```
Example 8:
```
        a=10
        b=5
        print( a is not b )
        output : True
```

Type Conversion

- It is a process of converting data of one data type to another data type.
- It is also called as type casting.
- It is 2 types.
  - A. Implicit type conversion
  - B. Explicit type conversion

Implicit type conversion: -

- In Implicit type casting, Python automatically converts data from one type to another.
- This process doesn't need any user involvement.
- Python avoids the loss of data in implicit type conversion.

  Example :
```
        a=10
        b=5.5
        sum=a+b
        print("sum=",sum)
```

Explicit type conversion: -

- In Explicit Type Conversion, users convert the data from one type to required type.
- We use predefined functions like int(), float(), str()… to perform explicit type conversion.
- In explicit type conversion loss of data may be occurred.
- Syntax :
```
        <required data type>(Expression)
```
  Example 1 : convert a char to ascii value.
```
        ch='a'
        ascii_value=ord(ch)
        print( “ASCII value=”,ascii_value )
        output :
                ASCII value= 97
```
  Example 2 : convert a ascii value to character.
```
        ascii_value=65
        ch=chr(ascii_value)
        print("Character=",ch)
        output :
                Character= A
```
  Example 3: convert string to int.
```
        no=int(input("Enter a number : "))
        print( "no=",no)
        output:
                Enter a number : 125
                no= 125
```

Example 4: convert string to float

```
        no=float(input("Enter a number : "))
        print( "no=",no)
        output :
                Enter a number : 12.50
                no= 12.5
```

Type caste Functions	**Description**
int(x [,base])	Converts x to an integer.
long(x [,base])	Converts x to a long integer.
float(x)	Converts x to a floating-point number.
complex(real [,imag])	Creates a complex number.
str(x)	Converts object x to a string representation.
repr(x)	Converts object x to an expression string.
eval(str)	Evaluates a string and returns an object.
tuple(s)	Converts s to a tuple.
list(s)	Converts s to a list.
set(s)	Converts s to a set.
dict(d)	Creates a dictionary, d must be a groupe of (key, value) tuples.
frozenset(s)	Converts s to a frozen set.
chr(x)	Converts an integer to a character.
unichr(x)	Converts an integer to a Unicode character.
ord(x)	Converts a single character to its integer value.
hex(x)	Converts an integer to a hexadecimal string.
oct(x)	Converts an integer to an octal string.

Expressions

- Expression is a combination of values, variables, operators and call of functions.
- After evaluate the expression, it produce a result, which is always a value.

Algebraic Expression	**Python Expression**
a x b – c	a * b – c
(m + n) (x + y)	(m + n) * (x + y)
(ab / c)	a * b / c
$3x^2$ +2x + 1	3*x*x+2*x+1
(x / y) + c	x / y + c

- Expressions are classified into 3 types.
  a. Infix expression
  b. Prefix expression
  c. Postfix expression

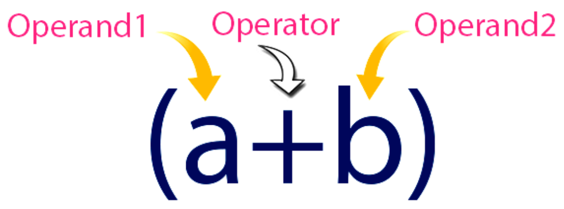
Infix Expression

In infix expression, operator is used in between operands.

General Structure

Operand1   Operator   Operand2

Example



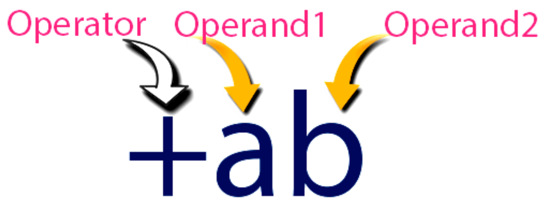
Prefix Expression

In prefix expression, operator is used before operands. We can say that "**Operands follows the Operator**".

General Structure

Operator Operand1 Operand2

Example



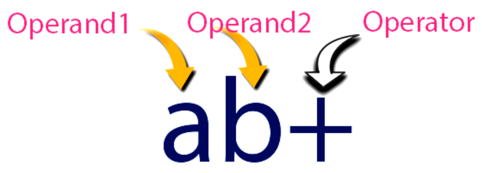
Postfix Expression

In postfix expression, operator is used after operands. We can say that "**Operator follows the Operands**".

General Structure

Operand1 Operand2 Operator

Example



Any expression can be represented using the above three different types of expressions.

Data Types

- ❖ These are the keyword
- ❖ These tell about the type of data.
- ❖ These are attributes of data.
- ❖ Python supports 8 types of data types.
  1. Number
  2. Boolean
  3. Byte
  4. List
  5. Tuple
  6. String
  7. Set
  8. Dictionary

1. Number :-

Numbers are classified into 3 types.

1. Integer numbers.
2. Floating pointer numbers.
3. Complex numbers.

Integer numbers :-

- ❖ Integer is a single or group of digits.
- ❖ Integer is a round number.

  Example :

  1, 2, 3, 12,125,

Sample Program:

```
no=int(input("enter an integer number : "))
print("No=",no)
```

output:

```
enter an integer number : 15
No= 15
```

Floating pointer numbers :-

It must be a group of digits those are separate decimal point (.)

Example :

12.5, 1.0

Sample Program:

```
no=float(input("enter an floating point number : "))
print("No=",no)
```

Output :

```
enter an floating point number : 12.55
No= 12.55
```

Complex numbers :-

Complex number is a mixing of real number and imaginary. It always satisfies the formula $j^2=-1$. Python provides built in function to handle complex numbers.

Sample program:

```
a=10+2j
b=5+3j
print("sum of complex number=",a+b)
print("product of complex numbers=",a*b)
```

Output :

```
sum of complex number= (15+5j)
product of complex numbers= (44+40j)
```

2. Boolean Datatype :-

- ❖ Whenever we evaluate an expression, its result is always True or False.
- ❖ The values of True or False are called Boolean values.

Example 1 :

```
a=True
b=False
print( a )
print( b )
```

output :

```
True
False
```

Example 2 :

```
a=10
b=20
c=30
```

```
        print( a==b )
        print( b==c )
        print( a==a )
    Output :
        False
        False
        True
```

3. **Byte :-**

   Whatever the data entered by you is in human readable. It cannot be understood by machine. We need to convert the input into machine readable. This process is called encoding. After manipulating the data, computer generate output. It is in machine readable format. We need to convert it into human readable. It is called decoding. Byte is a data in machine language. It is a machine word. It is a group of 8 bits.

   Example :

```
        str="Andhra Pradesh"
        encode_str=str.encode("ascii")
        decode_str=encode_str.decode("ascii")
        print("Original String=",str)
        print("Encoded String=",encode_str)
        print("Decoded Stringh=",decode_str)
```

   Output :

```
        Original String= Andhra Pradesh
        Encoded String= b'Andhra Pradesh'
        Decoded Stringh= Andhra Pradesh
```

4. **List :**
   - ❖ It is as same as arrays in C language.
   - ❖ It is a group of elements.
   - ❖ It contains the elements with different data types.
   - ❖ It always represent with in [ ].
   - ❖ List elements are separated with camma(,)
   - ❖ List elements are used using index starting with 0.
   - ❖ List elements are used with reverse index starting with -1.
   - ❖ List allows duplicate elements.
   - ❖ List allows to change.

   Example 1 : Create an empty list using [ ].

```
        list1=[ ]
        list2=list()
        print( list1 )
        print( list2 )
    output :
        [ ]
        [ ]
```

   Example 2: create a list with multiple elements.

```
        list1=[10,20,30,40]
        print( list1 )
    output :
        [10, 20, 30, 40]
```

Example 3 : create a list with different data type elements.

```
list1=['AP','UP','MP',10,20,30,40]
print( list1 )
```

OUTPUT :

```
['AP', 'UP', 'MP', 10, 20, 30, 40]
```

Example 4: use list elements with index.

```
list1=['AP','UP','MP',10,20,30,40]
print( list1[0] )
print( list1[3] )
print( list1[5] )
```

Output :

```
AP
10
30
```

Example 5 : use list elements with –ve index.

```
list1=['AP','UP','MP',10,20,30,40]
print( list1[-6] )
print( list1[-4] )
print( list1[-2] )
```

Output :

```
UP
10
30
```

Example 6 : use items of list using slicing operator [m:n]

```
list1=['AP','UP','MP',10,20,30,40]
print( list1[2:6] )
```

output :

```
['MP', 10, 20, 30]
```

5. **Tuple :-**
   - It is as same as list.
   - It is write protected data.
   - It can not be changed.
   - It has group of elements.
   - Elements are separated with camma.
   - Elements are enclosed with in ()
   - It allows duplicate elements.
   - Elements can be accessed using index.
   - Elements can be accessed using –ve index.

Create an empty tuple :-

```
t1=()
t2=tuple()
print( t1 )
print( t2 )
```

output :

```
( )
( )
```

Create a tuple with elements :-

```
t1=(501,'rama','cse')
print( t1 )
```

output :

```
(501, 'rama', 'cse')
```

Use data of tuple using index :-

```
t1=(501,'rama','cse')
print("Rno=",t1[0] )
print("Student=",t1[1] )
print("Branch=",t1[2])
```

output :

```
Rno= 501
Student= rama
Branch= cse
```

Use data using –ve index :-

```
t1=(501,'rama','cse')
print("Rno=",t1[-3] )
print("Student=",t1[-2] )
print("Branch=",t1[-1])
```

output :

```
Rno= 501
Student= rama
Branch= cse
```

6. **String :-**

- ❖ String is a group of characters.
- ❖ String not allow to change.
- ❖ String can be represented using ' ' or " " or ' ' ' ' ' ' or " " " " " "
- ❖ String allow to use "+" to concatenate.
- ❖ String allows to use "*" to repeat.

Example to create string in different model :

```
str1='rama'
str2="sita"
str3='''Aravind'''
str4="""Nani"""
print( str1 )
print( str2 )
print( str3 )
print( str4 )
```

output :

```
rama
sita
Aravind
Nani
```

Create multi line Strings :-

Multi line string can be created using ''' or """

```
str1='''I am
      a student
         of JNTUN'''
str2="""I am a faculty of
         RVRJC COLLEGE"""
print( str1 )
print( str2 )
```

output :
```
        I am
            a student
                of JNTUN
        I am a faculty of
                RVRJC COLLEGE
```
Use characters of a string :-

We can use characters of string using index.
```
str1="rama"
str2="sita"
print( str1[0] )
print( str2[2] )
```
output :
```
r
t
```
Built-in String methods for Strings:

SNO	Method Name	Description
1	capitalize()	Capitalizes first letter of string.
2	center(width, fillchar)	Returns a space-padded string with the original string centered to a total of width columns.
3	count(str, beg= 0,end=len(string))	Counts how many times str occurs in string or in a substring of string if starting index beg and ending index end are given.
4	decode(encoding='UTF-8',errors='strict')	Decodes the string using the codec registered for encoding. Encoding defaults to the default string encoding.
5	encode(encoding='UTF-8',errors='strict')	Returns encoded string version of string; on error,default is to raise a Value Error unless errors is given with 'ignore' or 'replace'.
6	endswith(suffix, beg=0, end=len(string))	Determines if string or a substring of string (if starting index beg and ending index end are given)ends with suffix; returns true if so and false otherwise.
7	expandtabs(tabsize=8)	Expands tabs in string to multiple spaces; defaults to 8 spaces per tab if tabsize not provided.
8	find(str, beg=0 end=len(string))	Determine if str occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise.
9	index(str, beg=0, end=len(string))	Same as find(), but raises an exception if str not found.
10	isalnum()	Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.
11	isalpha()	Returns true if string has at least 1 character and all characters are alphabetic and false otherwise.
12	isdigit()	Returns true if string contains only digits and false otherwise.
13	islower()	Returns true if string has at least 1 cased characterand all cased characters are in lowercase and false otherwise.
14	isnumeric()	Returns true if a unicode string contains only numeric characters and false otherwise.

15	isspace()	Returns true if string contains only whitespace characters and false otherwise.
16	istitle()	Returns true if string is properly "titlecased" and false otherwise.
17	isupper()	Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise.
18	join(seq)	Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string.
19	len(string)	Returns the length of the string.
20	ljust(width[, fillchar])	Returns a space-padded string with the original string left-justified to a total of width columns.
21	lower()	Converts all uppercase letters in string to lowercase.
22	lstrip()	Removes all leading whitespace in string.
23	maketrans()	Returns a translation table to be used in translates function.
24	max(str)	Returns the max alphabetical character from the string str.
25	min(str)	Returns min alphabetical character from the string str.
26	replace(old, new [, max])	Replaces all occurrences of old in string with new or at most max occurrences if max given.
27	rfind(str, beg=0,end=len(string))	Same as find(), but search backwards in string.
28	rindex(str, beg=0, end=len(string))	Same as index(), but search backwards in string.
29	rjust(width,[, fillchar])	Returns a space-padded string with the original string right-justified to a total of width columns.
30	rstrip()	Removes all trailing whitespace of string.
31	split(str="", num=string.count(str))	Splits string according to delimiter str (space if not provided) and returns list of substrings; split into atmost num substrings if given.
32	splitlines (num=string.count('\n'))	Splits string at all (or num) NEWLINEs and returns a list of each line with NEWLINEs removed.
33	startswith(str, beg=0,end=len(string))	Determines if string or a substring of string (if starting index beg and ending index end are given)starts with substring str; returns true if so and false otherwise.
34	strip([chars])	Performs both lstrip() and rstrip() on string.
35	swapcase()	Inverts case for all letters in string.
36	title()	Returns "titlecased" version of string, that is, allwords begin with uppercase and the rest are lowercase.
37	translate(table, deletechars="")	Translates string according to translation table str(256 chars), removing those in the del string.
38	upper()	Converts lowercase letters in string to uppercase.
39	zfill (width)	Returns original string leftpadded with zeros to atotal of width characters; intended for numbers, zfill() retains any sign given (less one zero).
40	isdecimal()	Returns true if a unicode string contains only decimal characters and false otherwise.

7. **<u>Set :-</u>**
   - ❖ It is a group of elements.
   - ❖ Elements separated with camma.
   - ❖ Elements are represented with in {}
   - ❖ Set does not have any duplicates.
   - ❖ It can be modified.

   <u>Create an empty set :-</u>
   ```
   set1={}
   set2=set()
   print( set1 )
   print( set2 )
   output :
   {}
   set()
   ```
   <u>Create a set with data :-</u>
   ```
   set1={1,2,3,4,5}
   print( set1 )
   output :
   {1, 2, 3, 4, 5}
   Example 2 :
   set1={1,2,3,4,5,1,2,3,4}
   print( set1 )
   output :
   {1, 2, 3, 4, 5}
   ```
   <u>Adding element to a set :-</u>

   We can add element to a set using add()
   ```
   set1={1,2,3,4,5}
   set1.add( 6 )
   set1.add( 3 )
   print( set1 )
   output :
   {1, 2, 3, 4, 5, 6}
   ```
8. **<u>Dictionary :-</u>**
   - ❖ It is a group of key and value pairs.
   - ❖ Key and value are separated with ":" symbol.
   - ❖ Dictionary is enclosed with in { }
   - ❖ It is used when we have huge amount of data.
   - ❖ Key is necessary to get values.

   <u>Create an empty dictionary.</u>
   ```
   dict1={}
   dict2=dict()
   print( dict1 )
   print( dict2 )
   output :
   { }
   { }
   ```
   <u>Create a dictionary with elements :</u>

   Example 1 :
   ```
           dict1={97:'a',98:'b',99:'c'}
   ```

```
        print( dict1 )
    output :
        {97: 'a', 98: 'b', 99: 'c'}
    Example 2 :
        dict1={'rama':'CSE','sita':'ECE'}
        print( dict1 )
    output :
        {'rama': 'CSE', 'sita': 'ECE'}
```

Adding elements to dictionary :-

```
    Example :
        dict1={97:'a',98:'b',99:'c'}
        dict1[100]='d'
        print( dict1 )
    output :
        {97: 'a', 98: 'b', 99: 'c', 100: 'd'}
```

Access elements of a dictionary :-

```
    Example :
        dict1={97:'a',65:'A',98:'b',66:'B'}
        print( dict1[97] )
        print( dict1[66] )
        print( dict1[98] )
        print( dict1[65] )
    output :
        a
        B
        b
        A
```

Delete an element from dictionary :-

```
    Example :
        dict1={97:'a',65:'A',98:'b',66:'B'}
        del dict1[98]
        print( dict1 )
    output :
        {97: 'a', 65: 'A', 66: 'B'}
```

Remove all elements of dictionary :-

```
    Example:
        dict1={97:'a',65:'A',98:'b',66:'B'}
        dict1.clear()
        print( dict1 )
    output :
        { }
```

Input and output functions

Display using print function :-

- it is a pre define function.
- It is used to print data on screen.
- The data can be a string, or any other object.
- These objects will be converted into a string before print on screen.

Syntax :

```
print(object(s), sep=separator, end=end)
```

Note:

- ❖ Objects are the value to be printed on screen.
- ❖ Optional. Specify how to separate the objects, if there is more than one. Default is ' '
- ❖ Optional. Specify what to print at the end. Default is '\n' (line feed)

Example 1 :

```
cname="VVIT"
print("College=",cname)
output : College= VVIT
```

example 2 :

```
rno=205
print("rno=",rno)
output: rno= 205
```

Example 3:

```
sname="Aravind"
print("Student="+sname)
output : Student=Aravind
```

Example 4:

```
rate=15
print("Rate="+str(rate))
output : Rate=15
```

Example 5: to print the result at specified location we have to user place holders {}

```
a=10
b=20
sum=a+b
print("sum of {} and {} is {}".format(a,b,sum))
output : sum of 10 and 20 is 30
```

Example 6:

```
a=10
b=20
print("big number between {0} and {1} is {1}".format(a,b))
output : big number between 10 and 20 is 20
```

Example 7 :

```
print("big number between {a} and {b} is {b}".format(a=10,b=20))
output : big number between 10 and 20 is 20
```

Example 8: if you want to specify how to separate multiple values we can specify using sep command.

```
a=10
b=20
c=30
d=40
print( a,b,c,d,sep=", ")
output : 10, 20, 30, 40
```

Example 9 : end command has to use to specify how print functions ends.

```
print("*",end=" ")
print("*",end=" ")
print("*",end=" ")
print("*",end=" ")
```

print("*",end=" ")
output : * * * * *

Read input using input () :-

- It is a system defined function.
- It can read data.
- It can read data as string.
- Syntax :

  input(prompt)

  Note :

  - Prompt is a String, representing a default message before the input.
  - Prompt is optional.

  Example 1: Read string type of data.

  ```
  vname=input("enter your village name : ")
  print("Village=",vname)
  ```

  Output :

  ```
  enter your village name : narasaraopeta
  Village= narasaraopeta
  ```

  Example 2 :- WAP to read int type of data.

  ```
  a=int(input("Enter first number : "))
  b=int(input("Enter second number : "))
  print("sum=",a+b)
  ```

  output : sum=30

  Example 3: WAP to read float format data.

  ```
  rate=float(input("enter rate of an item : "))
  print("Rate=",rate)
  ```

  Output :

  ```
  enter rate of an item : 10.20
  Rate= 10.2
  ```

Formatted Output

- There are several ways to present the output of a program.
- Data can be printed in a human readable form.
- Data can be written to a file for future use.
- Sometimes user wants more control on formatting of output.
- There are several ways to format output.
- To use formatted string literals, begin a string with f or F before the opening quotation mark or triple quotation mark.
- The format() help user to print formatted Output

1. Example to print data using place holder :
   ```
   print("sum of {} and {} is {}".format(10,20,30))
   ```
   output : sum of 10 and 20 is 30
2. Example to print data using place holder:
   ```
   print("Big number between {0} and {1} is {1}".format(10,20))
   ```
   Big number between 10 and 20 is 20
3. Example to print data using place holder:
   ```
   print("rno={no}\nsname={name}".format(no=10,name="ravi"))
   ```
   output :
   ```
   rno=10
   sname=ravi
   ```

- Formatting output using % operator.
  1. Example to print interger:
     print("No=%5d"%10)
     output : No=   10
  2. Example to print octal :
     print("No=%o"%10)
     output : No=12
  3. Example to print hexadecimal
     print("No=%X"%27)
     output : N0=1B
- Formatting output using center() method:
  1. Example
     str="rama"
     print(str.center(80))
     output :

```
                                        rama
```

- Formatting output using center() method:
  1. Example
     str="rama"
     print(str.rjust(80))
     output :

```
                                                                            rama
```

Indentation

- In c, c++, java we can build the code block using {}.
- Python does not support {} to indicate blocks of code.
- But in python we have to build a block using indentation.
- Indentation is a process of moving the text towards right and left.
- Indentation is a four spaces or a tab distance.
- Indentation must follow to throw out the block.
- Programs are easier to read.
- Indentation clearly identifies which block of code a statement belongs to.
- Blocks of code are denoted by line indentation.

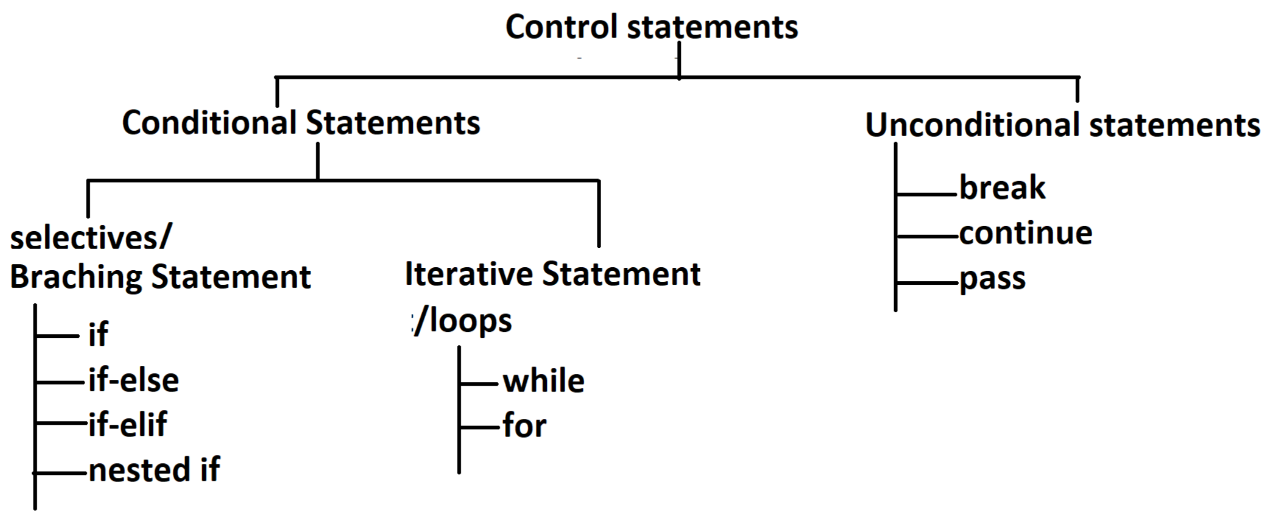
Of course, code blocks can consist of single statements, too. When one is new to Python, indentation may come as a surprise. Humans generally prefer to avoid change, so perhaps after many years of coding with brace delimitation, the first impression of using pure indentation may not be completely positive. However, recall that two of Python's features are that it is simplistic in nature and easy to read.

All the continuous lines indented with same number of spaces would form a block. Python strictly follow indentation rules to indicate the blocks.

Example :-

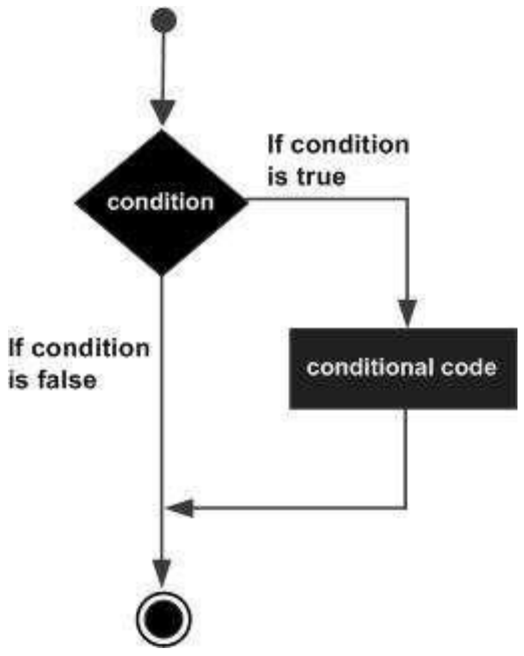
```
No=int(input(“enter a number : “))
if no%2==0 :
        print(“even number”)
else:
        print(“odd number”)
```

Decision Structures and Boolean Logic



<u>If :-</u>

- ❖ It is a control statement.
- ❖ It is a decision making statement.
- ❖ It is a conditional statement
- ❖ It is a selective statement.
- ❖ It is a branching statement.
- ❖ It works on result of given condition.



- ❖ Syntax :

```
if condition:
        Statements those executes if the condition is true
```

First, the condition is tested. If the condition is True, then the statements given after colon (:) are executed. We can write one or more statements after colon (:).

1. WAP to convert a lower case character to upper case character.

```
ch=input("Enter a lower case character : ")[0]
if ch>='a' and ch<='z':
   ch=chr(ord(ch)-32)
print("Upper case character =",ch)
```

Output 1:

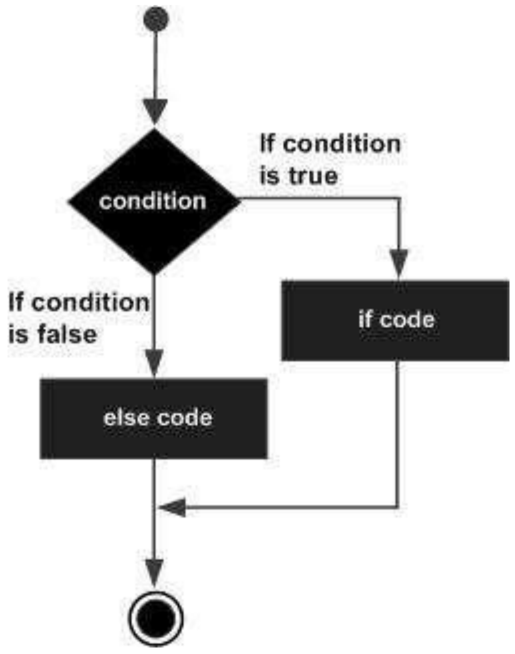
```
Enter a lower case character : a
Upper case character = A
```

Output 2:

```
Enter a lower case character : D
Upper case character = D
```

If-else :-

- It is a control statement.
- It is a decision making statement.
- It is a conditional statement
- It is a 2-way selection statement.
- It is a branching statement.
- It works on result of given condition.



- Syntax :

```
if condition:
        Statements those executes if the condition is true
else:
        Statements those executes if the condition is false.
```

1. **WAP to read a char. Print whether it is vowel or consonant.**

```
ch=input("Enter a character : ")[0]
str="aeiouAEIOU"
if ch in str:
   print("Vowel")
else:
   print("Consonant")
```

Output 1:

```
Enter a character : a
Vowel
```

Output2 :

```
Enter a character : T
Consonant
```

If-elif-else

- It is a control statement.
- It is a decision making statement.
- It is a conditional statement
- It is a multi-way selection statement.
- It is a branching statement.
- It works on result of conditions.
- Syntax :

```
if condition1:
        Statements those executes if the condition is true
elif condition2:
        Statements those execute when condition1 is false and condition2 is true.
else:
        Statements those executes if the condition is false.
```

1. **WAP to read 2 numbers, print whether a is big or b is big or both are equal.**

```
a=int(input("enter first number : "))
b=int(input("Enter second number : "))
if a>b:
   print("a is big")
elif b>a:
   print("b is big")
else:
   print("Both are equal")
```

output 1 :

```
enter first number : 10
Enter second number : 5
10 is big
```

output2:

```
enter first number : 15
Enter second number : 50
50 is big
```

output 3:.

```
enter first number : 50
Enter second number : 50
Both are equal
```

Nested if :-

A if statement with in another if statement is called nested if statement.

Syntax :

```
If condition1 :
        If condition2:
                Body
```

Note : body will execute when condition1 and condition 2 are true.

WAP to print the typed input is vowel or consonant or digit or symbol.

```
ch=input("Press a key : ")[0]
str1="aeiouAEIOU"
if (ch>='a' and ch<='z') or (ch>='A' and ch<='Z') :
   if ch in str1:
      print("Vowel")
```

```
  else:
     print("Consonant")
elif ch>='0' and ch<='9':
  print("digit")
else:
  print("Symbol")
```

output 1:

```
        Press a key : +
        Symbol
```

output2:

```
        Press a key : 7
        digit
```

output3:

```
        Press a key : k
        Consonant
```

output4:

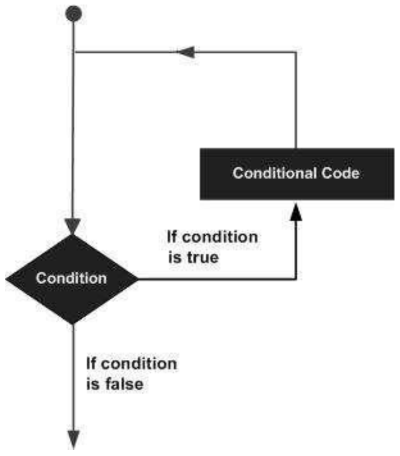
```
        Press a key : o
        Vowel
```

Decision Loops

In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times.

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:

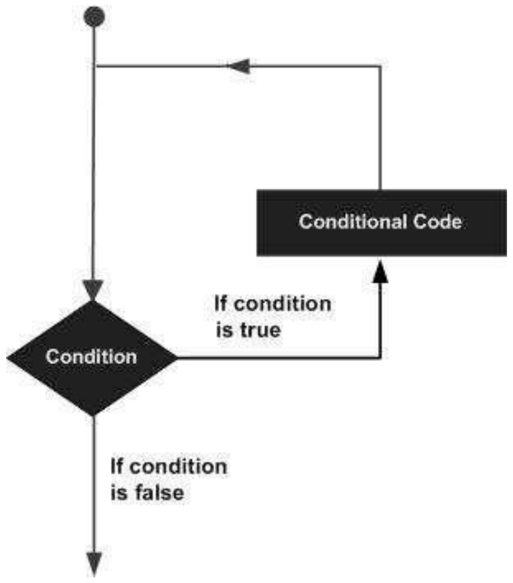


Python programming language provides following types of loops to handle loopingrequirements.

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 loop.

While Loop:-

- ❖ It is a control statement.
- ❖ It is a decision statement.
- ❖ It is a conditional statement.
- ❖ It is a pre test loop.
- ❖ It is used to execute the statements repeatedly.



❖ Syntax :

```
while condition:
        body
```

example :

1. **WAP to print your name 5 times.**

```
name="anand"
i=1
while i<=5:
    print(name)
    i+=1
```

Output :

```
    Anand
    Anand
    Anand
    Anand
    Anand
```

2. **WAP to print natural numbers from 1 to 100.**

```
no=1
while no<=100:
  print(no,end=", ")
  no+=1
```

output :

```
    1, 2, 3, 4, 5, … 100
```

For :-

- It is a control statement.
- It is a decision making statement.
- It is a conditional statement.
- It is a pre test loop.
- For in python is different from for loop in C language.
- It is used to iterate over the elements of a sequence.
- It can execute a group of statements repeatedly depending upon the number of elements in the sequence.
- The for loop can work with sequence like string, list, tuple, range etc.

  Syntax :

  ```
  for var in sequence:
          statement (s)
  ```

The first element of the sequence is assigned to the variable written after „for‟ and then the statements are executed. Next, the second element of the sequence is assigned to the variable and then the statements are executed second time. In this way, for each element of the sequence, the statements are executed once. So, the for loop is executed as many times as there are number of elements in the sequence.

1. **WAP to print 0 to 9.**
   ```
   for no in range( 10 ):
      print( no,end="  " )
   output : 0  1  2  3  4 …  9
   ```
2. **WAP to print natural numbers from 1 to 100.**
   ```
   for no in range( 1,101 ):
      print( no,end="  " )
   output : 1  2  3  4  5  …   100
   ```
3. **WAP to print odd numbers from 1 to 100.**
   ```
   for no in range( 1,101,2 ):
      print( no,end="  " )
   Output : 3    5   7    9    11    13  … 99
   ```
4. **WAP to print elements of a list.**
   ```
   list1=[10,20,30,40,50]
   for ele in list1:
      print(ele,end="  ")
   output: 10   20   30   40   50
   ```

Nested Loop:

It is possible to write one loop inside another loop. For example, we can write a for loop inside a while loop or a for loop inside another for loop. Such loops are called “nested loops”.

Example 1:

```
for i in range( 1,6 ):
   for j in range( i ):
      print("*",end=" ")
   print()
```

```
*
*   *
*   *   *
*   *   *   *
*   *   *   *  *
```

Example 2:

```
for i in range( 5,0,-1 ):
    for j in range( i ):
        print("*",end=" ")
    print()
```

```
*   *   *   *   *
*   *   *   *
*   *   *
*   *
*
```

Example 3:

```
        s=4
        for i in range(1,6):
            for j in range(s):
                print(" ",end=" ")
            for j in range(2*i-1):
                print("*",end=" ")
            print()
            s-=1
```

```
                  *
              *   *   *
          *   *   *   *   *
      *   *   *   *   *   *   *
```