Scripting Language

K.RaviKanth, Asst.Prof., Dept. of CSE, IIIT- RGUKT- Basara, T.S, IND.

COURSE DESCRIPTION:

- As an introductory course for the B.Tech, the student will be learning a very popular scripting language 'PYTHON', which is a pre-requisite to many Programming Languages.

- The purpose of the course is to provide the Basic programming methodology and writing programs in python This course will enable one to learn programming skills necessary to implement all the basic mathematical , scientific calculations and various operations. Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language.

- The study of computer science includes the study of how information is organized, manipulated and utilized in a computer. Thus it is extremely important for a student of computer science to understand the concepts of information organization and manipulation. It is important to recognize the logical connections among the data and represent them in a logical data structure.

- This will also give the foundation needed to learn any other programming language in an easy manner. By the end of the course the student will be able to know about different types of applications like Modules, Methods,searching, Data structures, List, Dictionaries, and IDE etc.

COURSE OBJECTIVES:

- To understand the various steps in Program development.
- To understand the basic concepts in Python Programming Language.
- To learn how to write programs in interactive mode and script mode
- To get knowledge how to use operators
- To learn to write programs using conditional, loop statements etc..
- To make the student understand simple sorting and searching methods.
- To get basic knowledge on Functions, Files and Exception Handling
- To understand the advanced concepts in Python Programming Language.
- To make the student understand class, embedding database using python programming
- To make the student to develop small project using Python programming language concepts.

COURSE OUTCOMES:

- The students will be able to write basic programs using python. It helps to improve logical and analytical skill set. It makes them easy and simple to learn high level programming skills.

- The students will be able to write advanced programs using python OOP, Networking and Web programming. It helps to develop industry oriented, web based and gaming applications.

COURSE PREREQUISITES:

- Logic & Reasoning [Mathematical Skills]
- Understanding Basic Computer Programming Terminologies
- Exposure to WINDOWS, UNIX / LINUX Environments

python
Programming Language

- **UNIT – I [12 Lectures]**
- **Introduction to Python Programming:** Features of Python, History of Python, Downloading and Installing Python, Writing and Executing First Python Program
- **Python Basics:** Literal Constants, Variables and Identifiers, Data Types, Input/Output Operations, Comments, Reserved Words, Indentation
- **Operators and Expressions:** Arithmetic, Comparison, Assignment, Relational, Unary, Bitwise, Shift, Logical, Membership, Identity, Operator Precedence and Associativity, Expressions
- **Decision Control Statements:** Selection/Conditional Branching Statements – if, if-else, Nested if, if- elif-else statements
- **Basic Loop Structures/Iterative Statements:** while, for, Nested loops, continue, break, pass statements
- **Assignments Programs and Case Study 1**

Introduction to Python Programming:

Features of Python

History of Python

Downloading and Installing Python

Writing and Executing First Python Program

Features of Python

–Beginners Language

-Simple & Easy to Learn

-- Free and Open Source

-- Multi Purpose [Web,GUI, Scripting etc..]

--High Level

-- Embedded

–Extensive Standard Library

–Cross Platform Compatibility

–Interactive Mode

-Interpreted

-- Object Oriented

–Portable, Extendable

–Databases and GUI Programming

–Scalable and Dynamic in nature

–Automatic Garbage Collection

python

Programming Language

Features of Python

1. Simple

- Reading a good Python program feels almost like reading English (Very strict English).
- This pseudo-code nature of Python is one of its greatest strengths. It allows you to concentrate on the solution to the problem rather than the language itself.

2. Easy to Learn

- Python is extremely easy to learn because Python has an extraordinarily simple syntax.
- Elegant syntax, making the programs you write easier to read.

3. Free and Open Source

- Python is an example of a FLOSS (Free/LibrÃ© and Open Source Software).
- In simple terms, you can freely distribute copies of this software, read it's source code, make changes to it, use pieces of it in new free programs, and that you know you can do these things. This is one of the reasons why Python is so good.

Features of Python

4. High-level Language

- When you write programs in Python, you never need to bother about the low-level details such as managing the memory used by your program, etc.

5. Portable

- Due to its open-source nature, Python has been ported (i.e. changed to make it work on) to many platforms. All your Python programs can work on any of these platforms without requiring any changes at all.
- You can use Python on Linux, Windows, FreeBSD, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, z/OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE and even PocketPC !

Features of Python

6. Interpreted

- A program written in a compiled language like C or C++ is converted from the source language i.e. C or C++ into a language that is spoken by your computer (binary code i.e. 0s and 1s) using a compiler with various flags and options. When you run the program, the linker/loader software copies the program from hard disk to memory and starts running it.
- Python, on the other hand, does not need compilation to binary. You just *run* the program directly from the source code. Internally, Python converts the source code into an intermediate form called bytecodes and then translates this into the native language of your computer and then runs it.
- All this, actually, makes using Python much easier since you don't have to worry about compiling the program, making sure that the proper libraries are linked and loaded, etc, etc. This also makes your Python programs much more portable, since you can just copy your Python program onto another computer and it just works.

Features of Python

7. Procedure Oriented and Object Oriented

- Python supports procedure-oriented programming as well as object-oriented programming. In *procedure-oriented* languages, the program is built around procedures or functions which are nothing but reusable pieces of programs. In *object-oriented* languages, the program is built around objects which combine data and functionality. Python has a very powerful but simplistic way of doing OOP, especially when compared to big languages like C++ or Java.

8. Extensible

- If you need a critical piece of code to run very fast or want to have some piece of algorithm not to be open, you can code that part of your program in C or C++ and then use them from your Python program.

9. Embeddable

- You can embed Python within your C/C++ programs to give 'scripting' capabilities for your program's users.

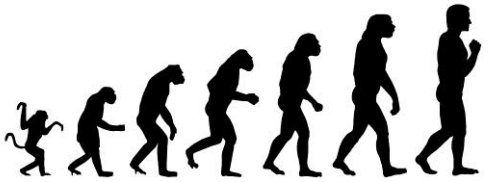
Features of Python

10. Extensive Libraries

- The Python Standard Library is huge indeed. It can help you do various things involving regular expressions, documentation generation, unit testing, threading, databases, web browsers, CGI, ftp, email, XML, XML-RPC, HTML, WAV files, cryptography, GUI (graphical user interfaces), Tk, and other system-dependent stuff. Remember, all this is always available wherever Python is installed. This is called the 'Batteries Included' philosophy of Python.

IIIT BASARA

History of Python



Python was created in the early 1980s by Guido van Rossum at CWI(Centrum Wiskunde & Informatica) in the Netherlands. CWI is the national research institute for mathematics and computer science.

Python language is named after the television show Monty Python's Flying Circus and many examples and tutorial include jokes from the show

Derives its features from many languages likeJava, C++, ABC, C, Modula3, Smalltalk, Algol-68, Unix shell and other scripting languages.

Available under the GNU General Public License (GPL) – Free and open-source software's

Major implementations of Python are Cpython, IronPython, Jpython,MicroPython,PyPy

python
Programming Language

History of Python

- Python is currently one of the most popular dynamic programming language.
- Python is often viewed as Scripting language but Python is really a general purpose language along the lines of Lips or smalltalk.
- Python is used for everything from throw-away scripts to large scalable web servers that provide uninterrupted service 24x7.
- Python is used for GUI and database programming, client and server side web programming, application testing, data analytics and scientific application development.

Introduction to Python Programming

•Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum.

•python first version is 1.0 released in 1991 and latest version is 3.6

•Python interpreters are available for many operating systems, allowing Python code to run on a wide variety of systems.

•Python uses whitespace indentation to delimit blocks rather than curly braces or keywords. An increase in indentation comes after certain statements.

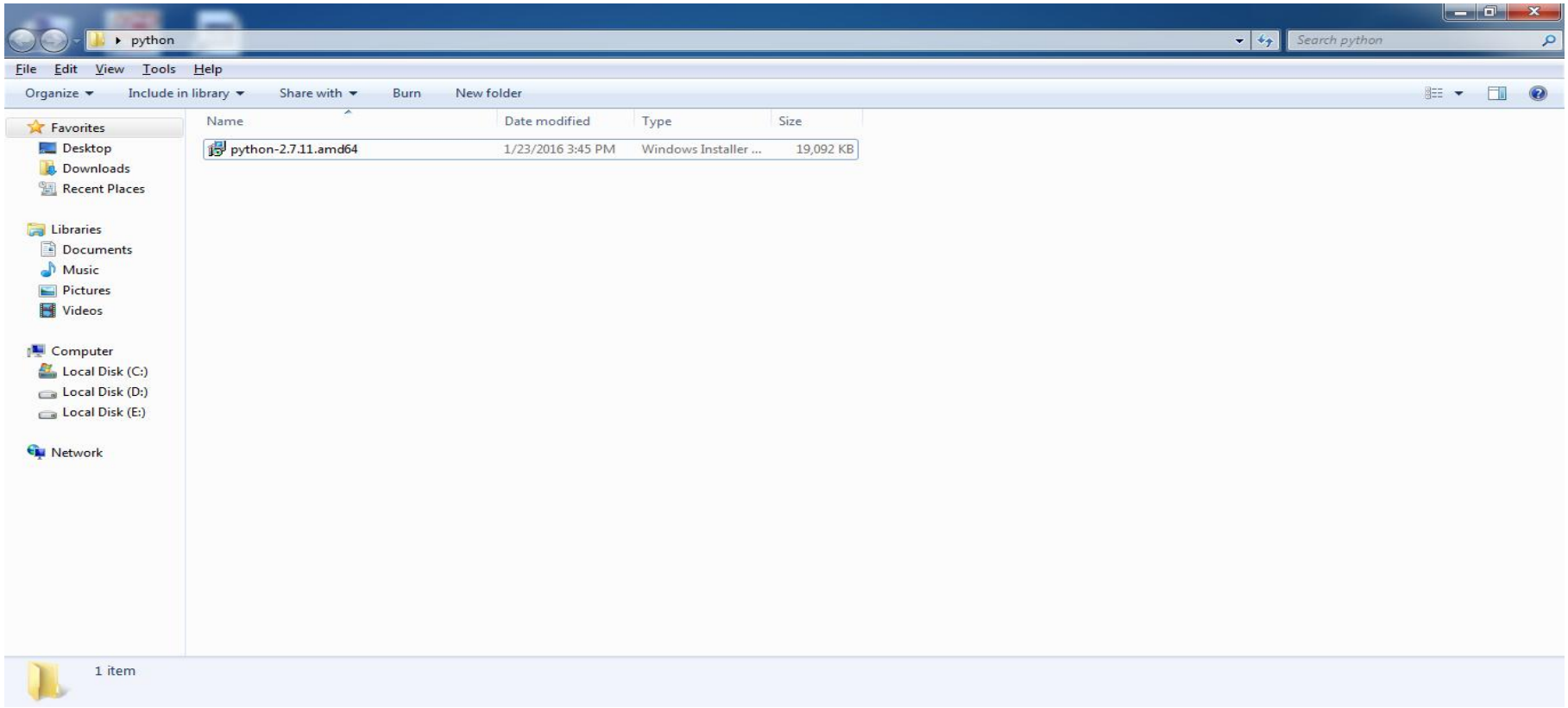
•Python file extension is .py

python

Programming Language

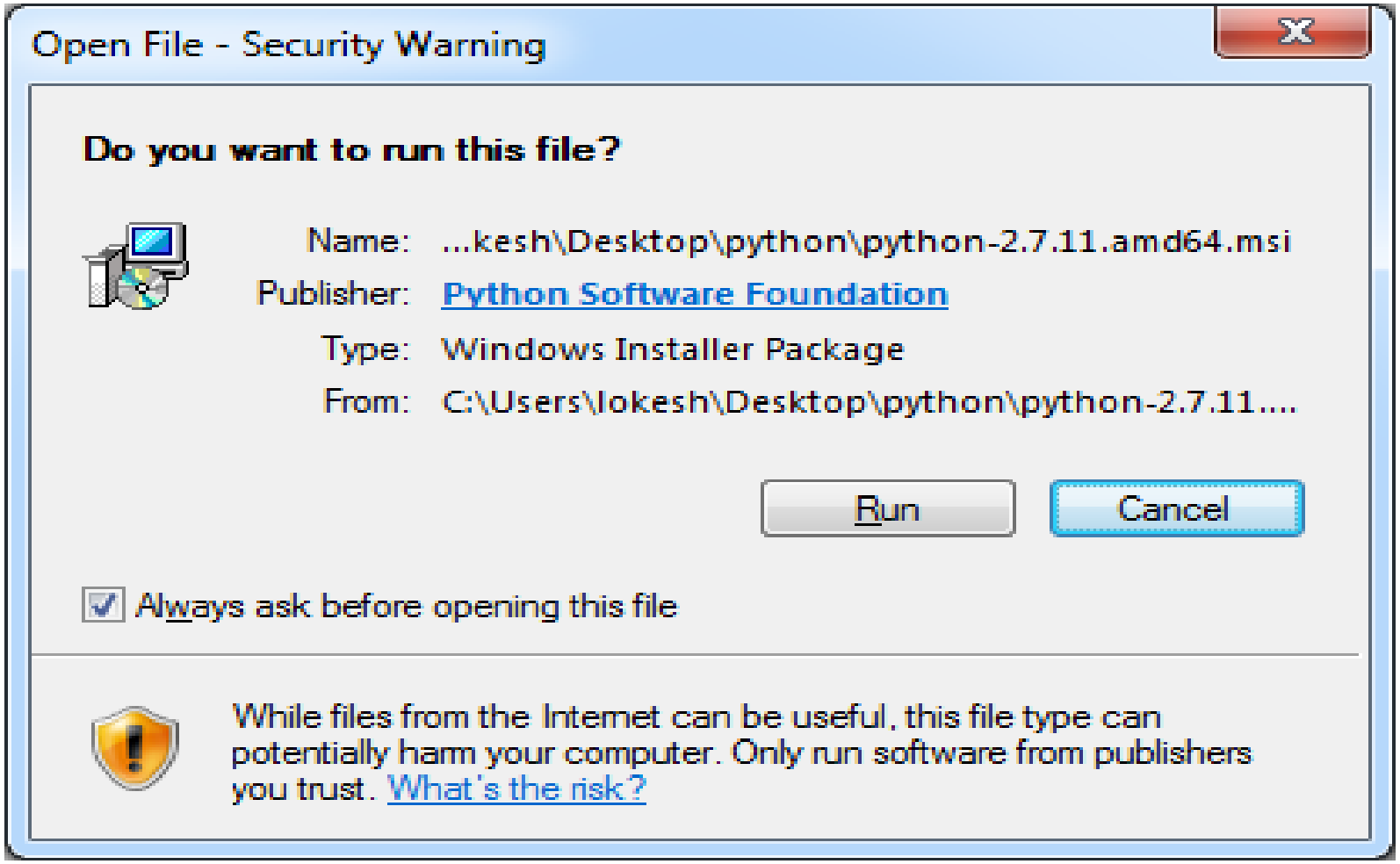
Downloading and Installing Python

- Download python from https://www.python.org/downloads website which is suitable for your machine.



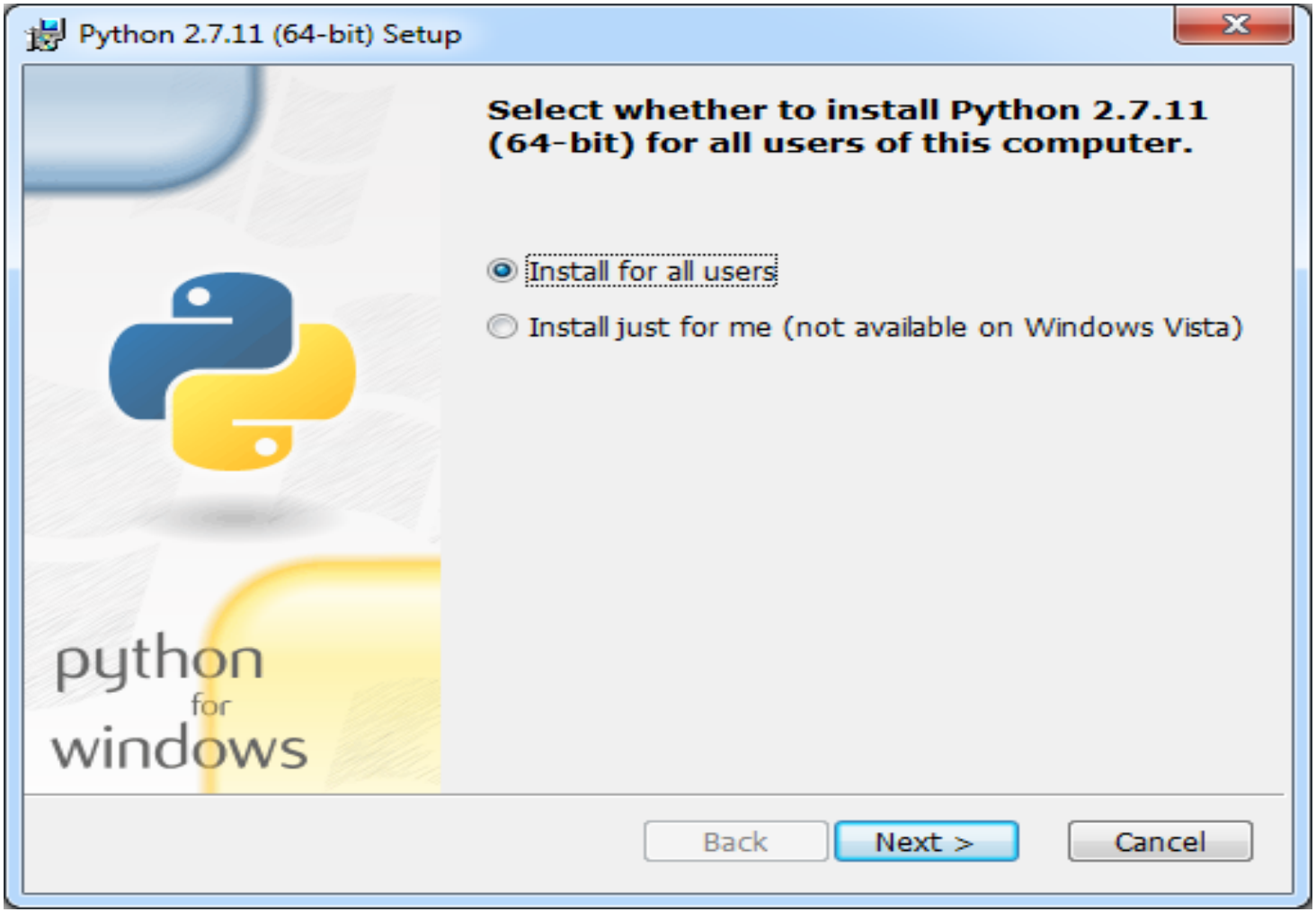
Click on installer

Downloading and Installing Python



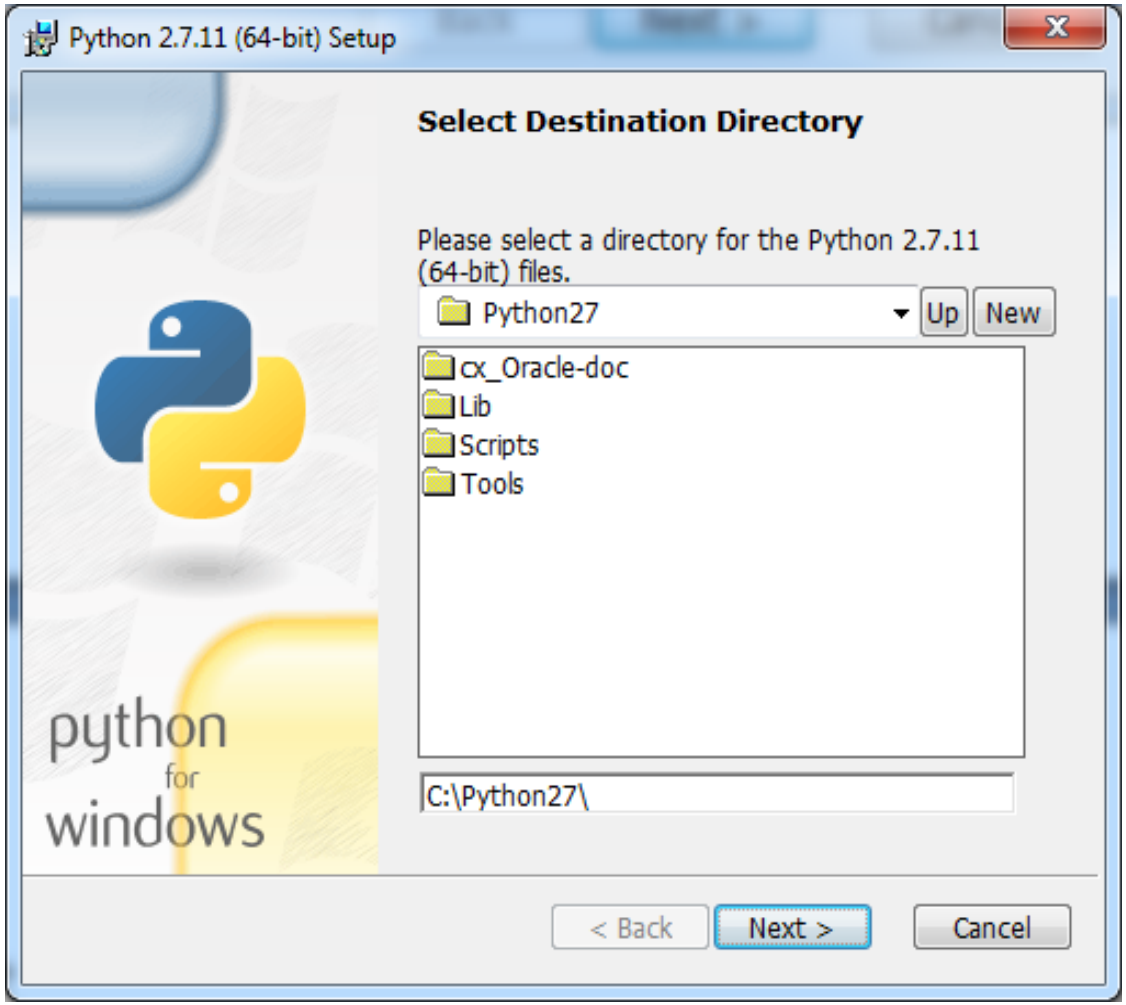
Click in run button

Downloading and Installing Python



Click on next

Downloading and Installing Python

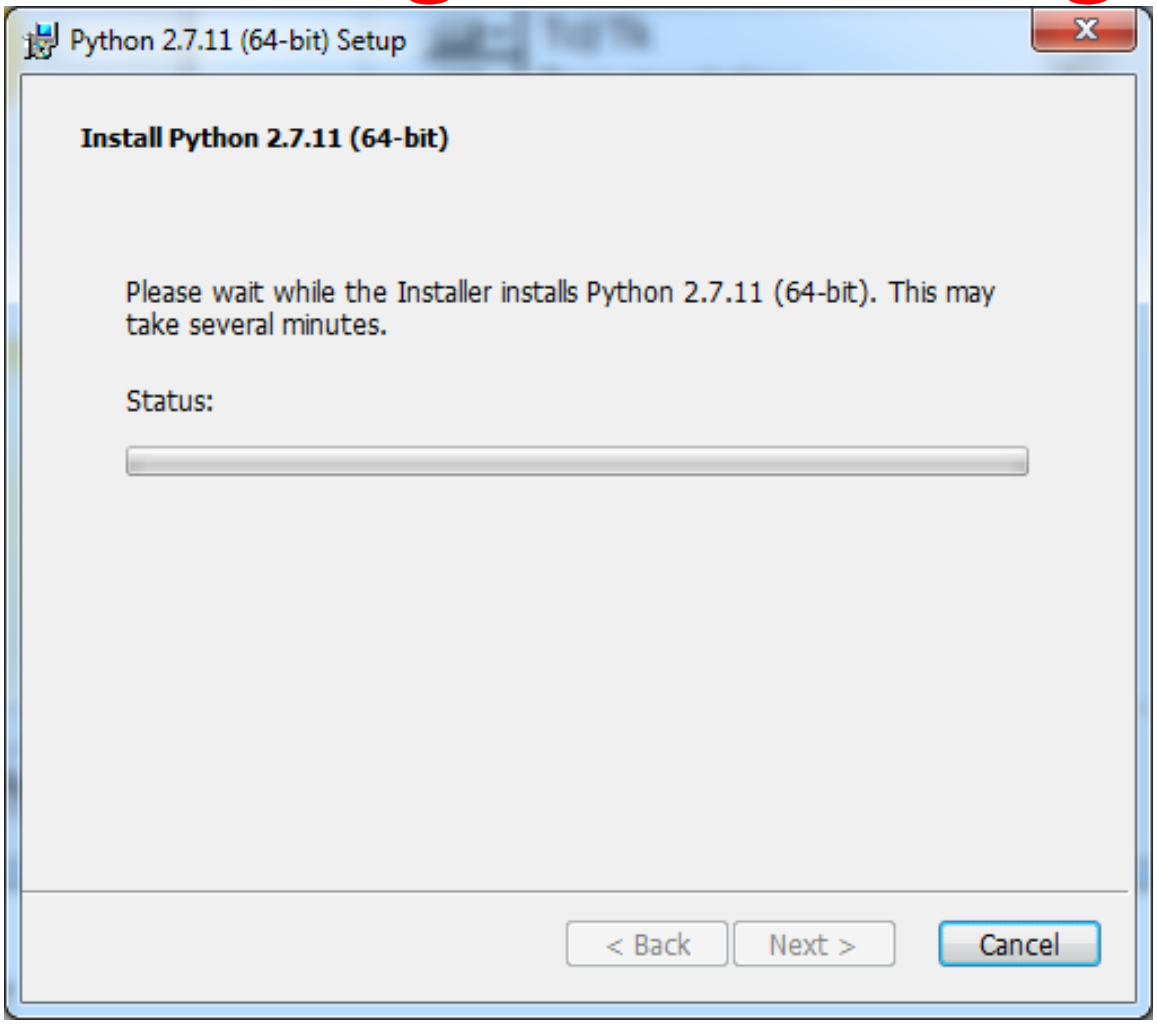


Click on next

Downloading and Installing Python

Click on next

Downloading and Installing Python



Click on yes

Downloading and Installing Python

Click on finish

IIIT BASARA

Python Programming Language

Why Python …?

Python is a popular programming language used for both standalone programs and scripting applications in a wide variety of domains. It is free, portable, powerful, and remarkably easy to use.

GOAL…!

Goal is to bring you quickly up to speed on the fundamentals of the Core Python Programming Language

Roughly 1 million Python users are out in the Industry market

The choice of *DEVELOPMENT TOOLS* is sometimes based on unique constraints or personal preference

python
Programming Language

Python Versions

- **Python v0.9.0 - February, 1991**

–Features: Exception Handling, Functions and core data types like List, Dictionary, String and others. It was object oriented and had module system

- **Python v1.0 - January 1994**

–Features: Functional Programming tools lambda, map, filter and reduce

- **Python v2.0 - October 2000**

–Features: List comprehensions, Garbage Collector and support for Unicode.

- **Python v3.0 - 2008**

–Known as "Python 3000" and "Py3k". It is not backward compatible with v2.0 and its other variants. Emphasizes more on removal of duplicate programming constructs and modules

- **Python v3.5 – 2016**

–Presently lets work with Python 3.5 …

python
Programming Language

Applications of Python

- 1. GUI-based desktop applications
- 2. Web frameworks and applications
- 3. Enterprise and business applications
- 4. Operating Systems
- 5. Language Development
- 6. Prototyping
- 7. Data Science & Analysis

python
Programming Language

Applications of Python

• **Embedded scripting language:** Python is used as an embedded scripting language for various testing/ building/ deployment/ monitoring frameworks, scientific apps, and quick scripts.

• **3D Software:** 3D software like Maya uses Python for automating small user tasks, or for doing more complex integration such as talking to databases and asset management systems.

• **Web development:** Python is an easily extensible language that provides good integration with database and other web standards.

•**GUI-based desktop applications:** Simple syntax, modular architecture, rich text processing tools and the ability to work on multiple operating systems makes Python a preferred choice for developing desktop-based applications.

Applications of Python

•**Image processing and graphic design applications**: Python is used to make 2D imaging software such as Inkscape, GIMP, Paint Shop Pro and Scribus. It is also used to make 3D animation packages, like Blender, 3ds Max, Cinema 4D, Houdini, Lightwave and Maya.

•***Scientific and computational applications:*** Features like high speed, productivity and availability of tools, such as Scientific Python and Numeric Python, have made Python a preferred language to perform computation and processing of scientific data.

•**Games:** It various modules, libraries, and platforms that support development of games. Games like Civilization-IV, Disney's Toontown Online, Vega Strike, etc. are coded using Python.

• **Enterprise and business applications:** Simple and reliable syntax, modules and libraries, extensibility, scalability together make Python a suitable coding language for customizing larger applications. For example, Reddit which was originally written in Common Lips, was rewritten in Python in 2005. A large part of Youtube code is also written in Python.

• **Operating Systems:** Python forms an integral part of Linux distributions

Worldwide Python Users

Web Development

–Yahoo Groups, Google, Shopzilla

Games

–Battelefield2, The Temple of Elemental Evil, Vampire

Graphics

–Walt Disney feature Animation, Blender 3D

Science

–National Weather Service

–NASA

–Environmental Systems Research Institutes

python
Programming Language

Why?

Google NASA Infosys Campus Connect MIT YAHOO!

Who uses Python ?:

- Google
- NASA
- Yahoo
- Youtube
- Linux (RedHat, Ubuntu, ...)
- Lots of researchers
- EVE online (Thousands of online players)
- MIT (Programming Intro. Course)
- etc...

INDUSTRIAL LIGHT & MAGIC

python

Watch this Google I/O video clip, Wesley Chun listing the uses of python and what companies use Python. (from 22:35 to 29:10)

YouTube

python
Programming Language

Python Users:

.Google makes extensive use of Python in its web search system, and employs Python's creator.

• The YouTube video sharing service is largely written in Python.

• The popular BitTorrent peer-to-peer file sharing system is a Python program.

• Intel, Cisco, Hewlett-Packard, Seagate, Qualcomm, and IBM use Python for hardware testing.

• Industrial Light & Magic, Pixar, and others use Python in the production of movie animation.

• JPMorgan Chase, UBS, Getco, and Citadel apply Python for financial market forecasting.

• NASA, Los Alamos, Fermilab, JPL, and others use Python for scientific programming tasks.

• iRobot uses Python to develop commercial robotic vacuum cleaners.

• ESRI uses Python as an end-user customization tool for its popular GIS mapping products.

• The NSA uses Python for cryptography and intelligence analysis.

• The IronPort email server product uses more than 1 million lines of Python code to do its ...

• The One Laptop Per Child (OLPC) project builds its user interface and activity model in Python.

Programming Language

Writing and Executing First Python Program

Step 1: Open an editor.

Step 2: Write the instructions

Step 3: Save it as a file with the filename having the extension .py.

Step 4: Run the interpreter with the command python program_name.py or use IDLE to run the programs.

To execute the program at the *command prompt*, simply change your working directory to C:\Python34 (or move to the directory where you have saved Python) then type python program_name.py.

If you want to execute the program in Python shell, then just press F5 key or click on Run Menu and then select Run Module.

Python is simple

```
print "Hello World!"
```
Python

```
#include <iostream.h>
int main()
{
cout << "Hello World!";
}
```
C++

```
public class helloWorld
{
    public static void main(String [] args)
    {
    System.out.println("Hello World!");
    }
}
```
Java

python
Programming Language

Coding Standards in Python

- Make sure of the readability and clarity of the program
- Make it easy to debug and maintain the program
- All the letters in a variable name should be lowercase
- When there are more than two words in variable name, underscore can be used between internal words
- Use meaningful names for variables
- Limit all lines to a maximum of 79 characters
- A function and a class should be separated by 2 blank lines
- Methods within the classes should be separated by single blank line
- Always surround binary operators with a space on either side Ex: a=a + 1

Write a Program to Display a Message on the Screen

```
>>> print("Hello All,Welcome to the World of Python Programming")
```

Press F5 for Execeution

Output: Hello All,Welcome to the World of Python Programming

Write a Program to justify thaat Programming with Python is quite simple and easy

```
a=10
b=20
c=a+b
print(c)
```

Result: 30

Write a Program to justify that Python is a Interpreted Programming Language

```
a=10
print(a)
b=0
c=a/b
print(c)
```

```
>>>
10
Traceback (most recent call last):
  File "C:\Users\Administrator\Desktop\interpreted.py"
    c=a/b
ZeroDivisionError: division by zero
>>>
```

TASK : Make use of IDE to Create and Execute a Python Program IDE Examples: Anaconda, Eclipse so on…

About Anaconda IDE:

Anaconda is a free and open source distribution of the Python and R programming languages for data science and machine learning related applications Anaconda consists of jupyterlab, jupyter notebook, qtconsole, spyder , rStudio and all. ... Python's PyCharm is an IDE, anaconda is a set of libraries or distribution of various libraries packed into one. Anaconda is professional data science platform and python IDE's

About Eclipse IDE:

Eclipse is a multi-language Integrated Development Environment and it has an extensible plug-in system. It can be used to develop applications in Python and also other programming languages including C,C++,Ruby, Perl,Python,COBOL,PHP,Java etc by means of various plug-ins

Every algorithm must have to satisfy the following properties:

1. **Finiteness** : Every alg. Must get terminated in known no. of steps
2. **Definiteness** : Each and every step of an alg. Must be easily and clearly understood
3. **Effectiveness** : Every step must be effective in the sense of programming the code & with in span of time
4. **Generality** : The alg. Must be complete in itself so that it can be used to solve all similar kind of problems for i/p data
5. **Input/Output** : Every alg. Must take Zero or More i/p data & produce one or more O/p

In general , the steps in an alg., can be divided into three Categories/Operations:

1. **Sequence -** A series of steps can be performed one after the other
2. **Selection** – Making a choice from multiple available options
3. **Iteration** – Performing repetitive tasks

Example:
Algorithm for adding two integer numbers :

Step1: Begin
Step2 : Initialize the variables sum with Zero
Step3 : Read two values
Step4 : Add the values & store in variable sum
Step5 : Print sum
Step6 : End

Flowchart:
It is a graphical representation of logical sequence of a given problem ie., it is a pictorial representation of an algorithm. It uses some symbols for specific tasks

Programming Development Lifecycle

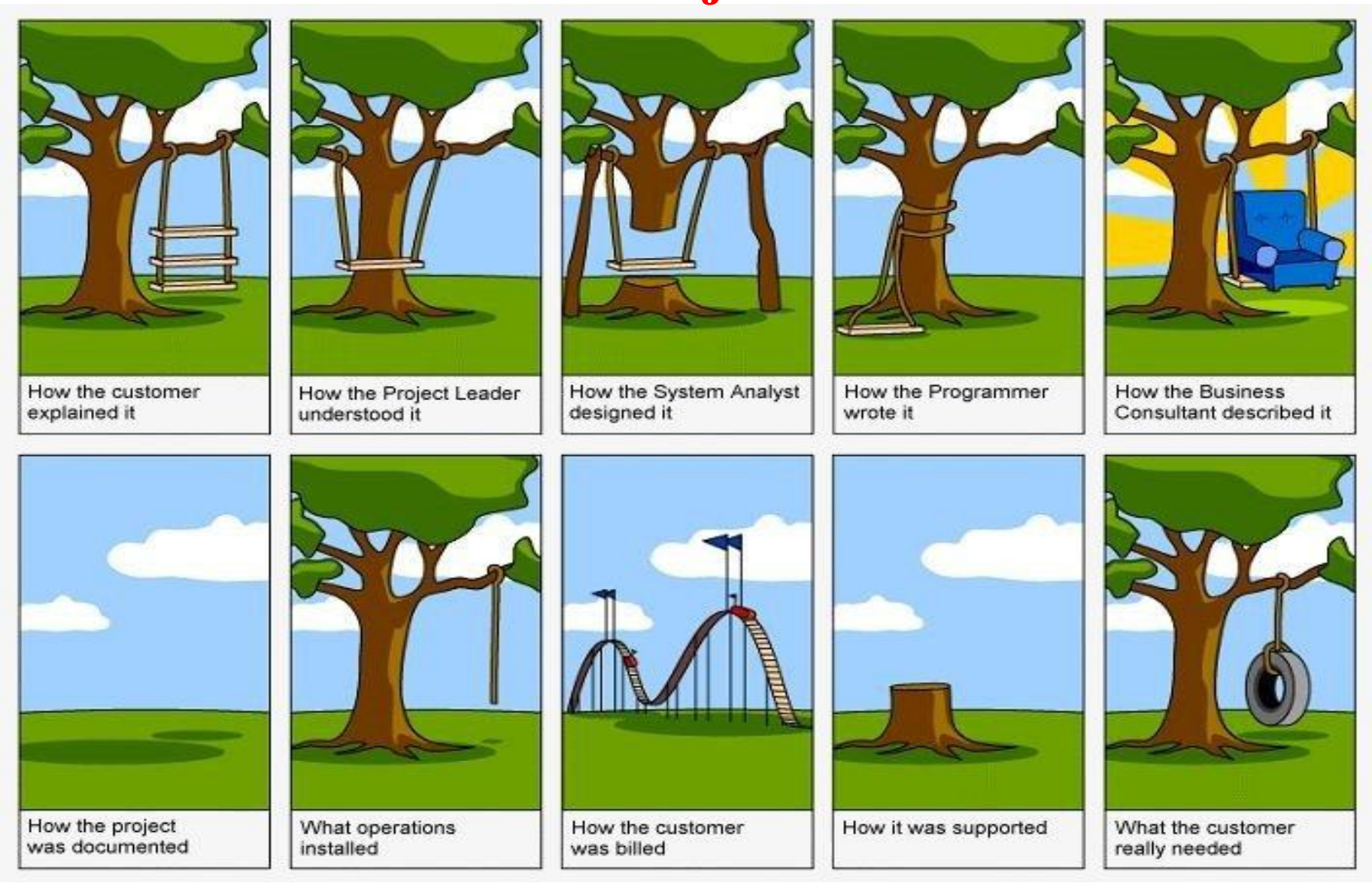
The program development life cycle is a set of steps or phases that are used to develop a program in any programming language.

Generally, program development life cycle contains 6 phases, they are as follows….

1. Problem Definition/ Problem Specification
2. Problem Analysis / Outlining the solution
3. Algorithm Development / Selecting & representing the Alg.
4. Designing the Algorithm using pictorial representation
5. Coding & Documentation/ Programming the algorithm
6. Debugging (Removing Errors)
7. Testing & Validation
8. Documentation & Maintenance

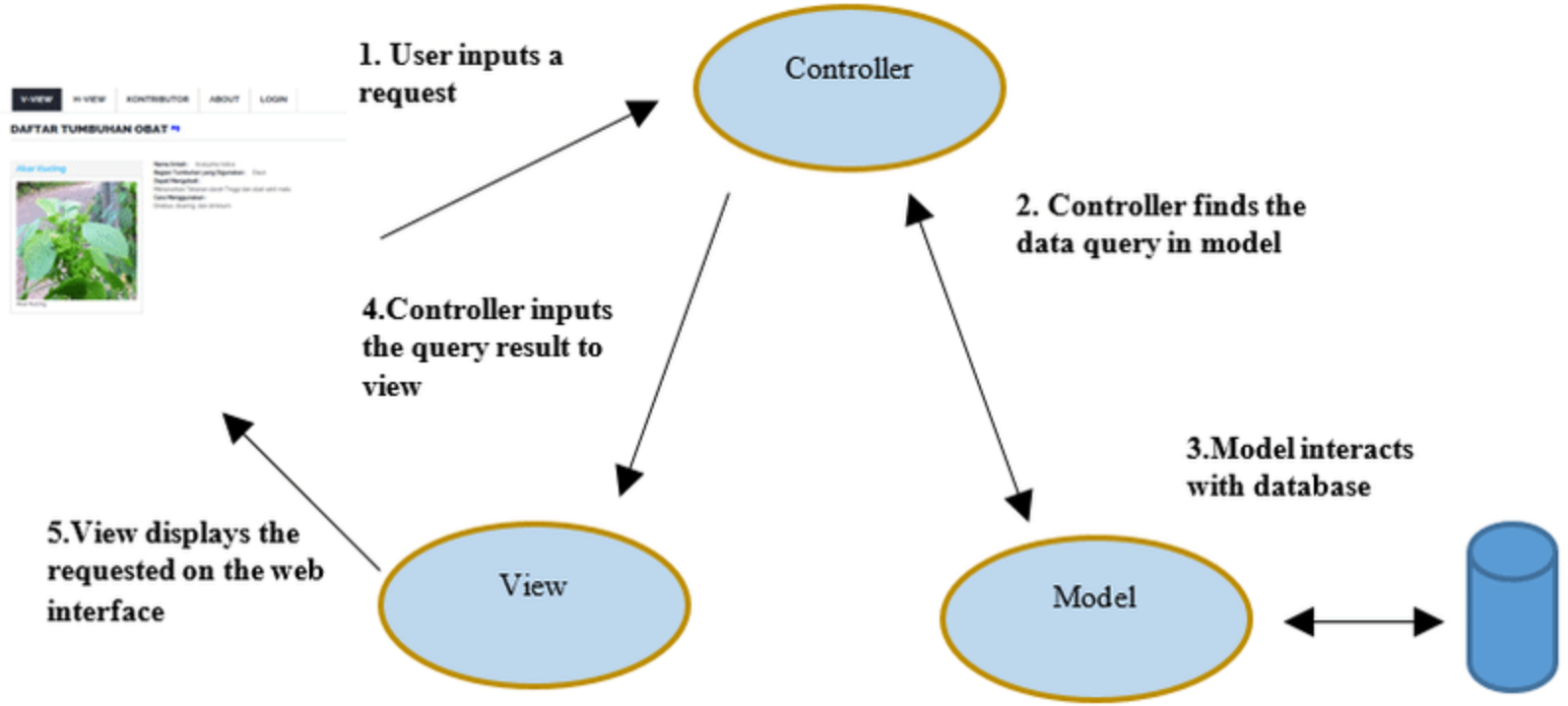
Programming Language

A Real Time Project Scenario…



Model View Controller (MVC)

- Model View Controller (**MVC**) is a software **architecture** pattern, commonly used to implement user interfaces: it is therefore a popular choice for architecting web apps. In general, it separates out the application logic into three separate parts, promoting modularity and ease of collaboration and reuse



Python Basics:

Literal Constants
Variables and Identifiers
Data Types
Input/Output Operations
Comments
Reserved Words
Indentation

Working with Python Tool

There are two ways to use the Python interpreter:

1. shell mode

In shell mode, you type Python expressions into the **Python shell**, and the interpreter immediately shows the result.

Ex:

```
>>> 2 + 3
5
>>>
```

The >>> is called the **Python prompt**.

The interpreter uses the prompt to indicate that it is ready for instructions

2. ***program mode***

create a source code file named **firstprogram.py**

Ex:

```
print("My first program adds two numbers, 2 and 3:")
print(2 + 3)

My first program adds two numbers, 2 and 3:
5
```

Windows users:

The best way to program in Windows is using the IDLE GUI (from the shortcut above.) To create a new program, run the command File->New Window (CTRL+N) – this will open up a blank editing window.

In general, save your program regularly; after saving, you can hit F5 to run your program and see the output.

Write a Program to display Hello World! Message on the screen

>>> print(“ Hello, World! ")

Hello World!

Note: The quotation marks in the program mark the beginning and end of the value. They don’t appear in the result.

```
>>>print(”Hello World!”)
>>>print(”Hello Again”)
>>>print(”I like typing this”)
>>>print(”This is funny”)
>>>print(’yay! Its Printing.’)
>>>print(”I’d much better than you ‘not’.”)
>>>print(’I “said” do not touch this.’)
```

English like commands

```
>>> name =“RGUKT-Basar”
>>> print (name)
```

One-liners

Ex: Swapping 2 variables

Ex: c : Swap x&y

```
int temp =x;
x=y;
y=temp;
```

Ex: Python

```
>>> x,y=y,x
```

Dynamically Typed Language

Ex:

C:

```
int  x=1;
x =(int)x/2;
```

O/p: 0 [can never be equal to 0.5]

Where as in Python

```
>>>x=1
>>> x=x/2
```

Now it is equal to 0.5 [Floating Point Variable]

Examples: Python as Calculator

>>>print("5+2", 5+2)	7
>>>print("5-2",5-2)	3
>>>print("5*2",5*2)	10
>>>print("5/2",5/2)	2.5
>>>print("5%2",5%2)	1
>>>print("5**2",5**2)	25
>>>print("5//2",5//2)	2
>>>print("1+2-3*2= ", 1+2-3*2)	-3
>>>print("(1+2-3)*2=",(1+2-3)*2)	0
>>> 2**100	1267650600228229401496703205376 L
>>> 2**1000	

Python Basics: Literal Constants

The value of a literal constant can be used directly in programs.

For example: 7, 3.9, 'A', and "Hello" are literal constants.

Numbers refers to a numeric value. You can use four types of numbers in Python program - integers, long integers, floating point and complex numbers.

• Numbers like 5 or other whole numbers are referred to as ***integers*.**

•Bigger whole numbers are called ***long integers*.**

For example, 535633629843L is a long integer.

• Numbers like are 3.23 and 91.5E-2 are termed as ***floating point numbers***.

• Numbers of a + bi form (like -3 + 7i) are ***complex numbers***.

Examples:

- print(10+7)
- print(50+40-35)
- print(12*10)
- print(96/12)
- print((-30*4)+500)
- print(78//5)
- print(78%5)
- print(152.78//3.0)
- print(152.78%3.0)
- print(5**3)
- print(121**0.5)

Output:

```
>>>
17
55
120
8.0
380
15
3
50.0
2.7800000000000001
125
11.0
>>>
```

Python Basics: Literal Constants

Strings

A *string* is a group of characters.

- ***Using Single Quotes (‘):*** For example, a string can be written as 'HELLO'.
- ***Using Double Quotes (“):*** Strings in double quotes are exactly same as those in single quotes. Therefore, 'HELLO' is same as "HELLO".
- ***Using Triple Quotes (‘ ‘ ‘ ‘ ‘ ‘):*** You can specify multi-line strings using triple quotes.

You can use as many single quotes and double quotes as you want in a string within triple quotes.

Examples:

```
print( ‘ Python Programming ')

print( " Python Programming " )

print( ''' Python Programming ''')
```

Escape Sequences

•An *escape character is a special character that is preceded with a backslash (\), appearing* inside a string literal.

•When a string literal that contains escape characters is printed, the escape characters are treated as special commands that are embedded in the string.

Some of Python's escape characters

Escape Character	Effect
\n	Causes output to be advanced to the next line.
\t	Causes output to skip over to the next horizontal tab position.
\‘	Causes a single quote mark to be printed.
\"	Causes a double quote mark to be printed.
\\	Causes a backslash character to be printed.

```
print('One\nTwo\nThree')
print('Mon\tTues\tWed')
print('Thur\tFri\tSat')
print("Your assignment is to read \“PYTHON\" by tomorrow.")
print('I\'m ready to Start.')
print('The path is C:\\temp\\data.')
```

Escape Sequences

Some characters (like ", \) cannot be directly included in a string.

Such characters must be escaped by placing a backslash before them.

Example:

```
>>> print("The boy replies, \"My name is Aaditya.\"")
The boy replies, "My name is Aaditya."
```

Escape Sequence	Purpose	Example	Output
\\	Prints Backslash	print("\\")	\
\'	Prints single-quote	print("\'")	'
\"	Prints double-quote	print("\"")	"
\a	Rings bell	print("\a")	Bell rings
\f	Prints form feed character	print("Hello\fWorld")	Hello World
\n	Prints newline character	print("Hello\nWorld")	Hello World
\t	Prints a tab	print("Hello\tWorld")	Hello World
\o	Prints octal value	print("\o56")	.
\x	Prints hex value	print("\x87")	+

Raw Strings

If you want to specify a string that should not handle any escape sequences and want to display exactly as specified then you need to specify that string as a *raw string*.

A raw string is specified by prefixing r or R to the string.

Example:

```
print (r"What \'s your name")
```

O/P:What's your name

Python Basics: Variables and Identifiers

Variable means its value can vary.

You can store any piece of information in a variable.

Variables are nothing but just parts of your computer's memory where information is stored.

To be identified easily, each variable is given an appropriate name.

Identifiers are names given to identify something.

Identifier is the name given to entities like class, functions, variables etc. in Python. It helps differentiating one entity from another.

For naming any identifier, there are some basic rules like:

- **The first character of an identifier must be an underscore ('_') or a letter (upper (A to Z) or lowercase(a to z))**

Python Basics: Variables and Identifiers

Names like myClass, var_1 and print_this_to_screen, all are valid example.

An identifier cannot start with a digit. 1variable is invalid, but variable1 is perfectly fine.

- Identifier names are case-sensitive.

For example, myvar and myVar are not the same.

- Punctuation characters such as @, $, and % are not allowed within identifiers.

Python Basics: Variables and Identifiers

- *Examples of valid identifier names* are :

  sum, __myvar, num1, r, var_20, First, etc.

- *Examples of invalid identifier names* are :

  1num, my-var, %check, Basic Sal, H#R&A, etc.

Python Variables

A variable is a location in memory used to store some data (value). The rules for writing a variable name is same as the rules for writing identifiers in Python.

We don't need to declare a variable before using it. In Python, we simply assign a value to a variable and it will exist. We don't even have to declare the type of the variable. This is handled internally according to the type of value we assign to the variable.

Variable assignment

We use the assignment operator (=) to assign values to a variable. Any type of value can be assigned to any valid variable.

Example

```
a = 5
b = 3.2
c = "Hello"
```

Note: Every Variable in Python is a Object

Keywords cannot be used as identifiers.

```
>>> global = 1
File "<interactive input>", line 1
global = 1
^
SyntaxError: invalid syntax
```

Python Statement

Instructions that a Python interpreter can execute are called statements. For example, a = 1 is an assignment statement.

Multi-line statement

In Python, end of a statement is marked by a newline character. But we can make a statement extend over multiple lines with the line continuation character (\). For example:

```
a = 1 + 2 + 3 + \
4 + 5 + 6 + \
7 + 8 + 9                print(a)
```

Multiple Statements on a Single Line

If you want to specify more than one statement in a single line, then use a Semicolon(;) to separate the two statements

Ex: msg="hello";print(msg)

msg="hw r u";print(msg)

Assigning or Initializing Values to Variables

- In Python, programmers need not explicitly declare variables to reserve memory space. The declaration is done automatically when a value is assigned to the variable using the equal sign (=).
- The operand on the left side of equal sign is the name of the variable and the operand on its right side is the value to be stored in that variable.

Assignments

- Assign a **value** to a **name**

  i = 5

  j = 2*i

  j = j + 5
- Left hand side is a name
- Right hand side is an expression
- Operations in expression depend on type of value
- **Ex:**
- >>>student_id =121021
- >>> student_name=“Ravi”
- >>>student _avg_marks=78.67
- >>>print(student_id,student_name,student_avg_marks)

Multiple Assignments

In Python, multiple assignments can be made in a single statement as follows:

```
a, b, c = 5, 3.2, "Hello"
```

If we want to assign the same value to multiple variables at once, we can do this as

```
x = y = z = "same"
```

This assigns the "same" string to all the three variables.

Note: Datatype of a variable can change during program execution compared to other strongly typed languages such as C,C++,Java

Variable Reassignment

We can override the value of variable with different values known as variable Reassignment

Ex:

```
X=10
print(x)
o/p: 10

X=20
Print(x)
o/p:20
```

Assigning or Initializing Values to Variables

Example:

```
num = 7
amt = 123.45
code = 'A'
pi = 3.1415926536
population_of_India = 10000000000
msg = "Hi"

print("NUM = "+str(num))
print("\n AMT = "+ str(amt))
print("\n CODE = " + str(code))
print("\n POPULATION OF INDIA = " + str(population_of_India))
print("\n MESSAGE = "+str(msg))
```

OUTPUT

```
NUM = 7
AMT = 123.45
CODE = A
POPULATION OF INDIA = 10000000000
MESSAGE = Hi
```

Example:

```
num=7
amt=123.45
code='A'
pi=3.1415926536
population=10000000000000
msg="hi"
print "num",num
print "amt",amt
print "code",code
print "pi",pi
print "population",population
print "msg",msg
```

- **Output:**

```
num 7
amt 123.45
code A
pi 3.1415926536
population 10000000000000
msg hi
```

Data types in Python

Every value in Python has a datatype.

Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.

There are various data types in Python. Some of the important types are listed below.

- **Python Numbers**
- **Python List**
- **Python Tuple**
- **Python String**
- **Python Set**
- **Python Dictionary**

Data Types in Python

Category	Data Type	Example
Integer Type	int	348
	long	38432456876765291 8L
	complex	2+5i
Floating Type	float	349.876
Textual	char	C
	String	Python
Logical	Boolean	True , False

- **Integer :** An integer is a number without a fractional part, e.g. -4, 5, 0, -3.
- **Float :** Any rational number, e.g. 3.432.
- **String :** Any sequence of characters.
- **Boolean :** Variables of this type can be either True or False.

Python Basics: Data Types

- **Data Type Boolean**
- Boolean is another data type in Python. A variable of Boolean type can have one of the two values- True or False. Similar to other variables, the Boolean variables are also created while we assign a value to them or when we use a relational operator on them.

Examples:

>>>Boolean_var = True >>>print(Boolean_var) True	>>> 20 == 30 False	>>>"Python" == "Python" True
>>> 20 != 20 False	>>>"Python"! = "Python3.4" True	>>>30 > 50 False
>>> 90 <= 90 True	>>>87 == 87.0 False	>>>87 > 87.0 False
>>>87 < 87.0 False	>>>87 >= 87.0 True	>>>87 <= 87.0 True

Programming Tip: <, > operators can also be used to compare strings lexicographically.

Datatypes in Python

Numeric Values

- Numbers come in two ways

  int – integers

  float – fractional numbers

Ex: 172 , -5 , 23453654 are values of type **int**

42.63 , -0.02 , 37.33232 are values of type **float**

Ex:

The following expressions would result in an error.

```
>>>“Hello” / 123
>>>“Hi” + 5
>>>“5” + 7
```

Ex:

```
>>>x=5.3+0.9j
>>>print(x.real)  # prints 5.3
>>>print(x.imag+3)   # prints   3.9
```

Operations on numbers

- Normal arithmetic operators : +, - ,* , /
- Note that / always produces a float
- 7/3.5 is 2.0 , 7/2 is 3.5
- Quotient and Remainder : // and %
- 9//5 is 1, 9%5 is 4
- Exponentiation : **
- 3**4 is 81 [3*3*3*3]

- log( ), sqrt( ),pow( ), sin( ), …
- We must include math " library"
    - from math import*

Names, Values and Types

•Names can be assigned values of different types as the program evolves

```
i = 5    #  i is int
i = 7*1   # i is still int
j = i/3  # j is float,  / creates float
```

•**type(e)** returns type of expression e

Ex:

```
>>> i =5
>>> type(i)
<class 'int'>
>>> j= 7.5
>>> type(j)
    < class 'float'>
```

```
>>> i = 2*j
>>> i
15.0
>>> type(i)
<class 'float')
```

Note: Not a good style to assign mixed type values to same name

To print ASCII to Char & Char to ASCII

```
x=65
print (chr(x))  # From Number to char


 y='a'
print(ord(y))  # From Char to ASCII
```

Python Basics: Input / Output Operations

- **Input Operation**
- To take input from the users, Python makes use of the input( ) function. The input( ) function prompts the user to provide some information on which the program can work and give the result.
- However, we must always remember that the input function takes user's input as a string. So whether you input a number or a string, it is treated as a string only

- **input( )** – input function is used to take input from the user
- **raw_input( )** – it takes the input in the form of a string and reads the data line by line or line at a time

Example: Program to read variables from the user

```
name=input("what's your Name?")
age=input("Enter your Age:")
DOB=input("Enter your DOB:")
print(name+",you are "+age+" \
years old.Your DOB is "+DOB)
```

```
name=raw_input("What is your name")
age=input("how old are you")
print "hi {} your age is {}".format(name,age)
```

Python Output Using print( ) function

We use the print( ) function to output data to the standard output device (screen).

Example

```
>>>print('This sentence is output to the screen')
Output: This sentence is output to the screen

>>>a = 5
>>>print('The value of a is', a)

# Output: The value of a is 5
```

Output formatting

- Sometimes we would like to format our output to make it look attractive. This can be done by using the **str.format()** method.
- This method is visible to any string object.

Example:

```
x = 5; y = 10
 print('The value of x is {} and y is {}'.format(x,y))
```

Output:- The value of x is 5 and y is 10

Note: Here the curly braces {} are used as placeholders. We can specify the order in which it is printed by using numbers (tuple index).

Example

```
print('I love {0} and {1}'.format('bread','butter'))
# Output: I love bread and butter
print('I love {1} and {0}'.format('bread','butter'))
# Output: I love butter and bread
```

We can even use keyword arguments to format the string.

print('Hello {name}, {greeting}'.format(greeting = 'Goodmorning', name = 'John'))

O/P: Hello John, Goodmorning

```
>num = input("Enter a number: ")
 >print (num)
```

 # program that uses the input function to read two strings as input from the keyboard.

```
# Get the user's first name.
first_name = input('Enter your first name: ')

# Get the user's last name.
last_name = input('Enter your last name: ')

# Print a greeting to the user.
print('Hello', first_name, last_name)
```

Reading Numbers with the input Function

Data Conversion Functions

int(item) – You pass an argument to the int() function and it returns the argument's value converted to an int.

float(item) - You pass an argument to the float() function and it returns the argument's value converted to a float.

```
>string_value = input('How many hours did you work? ')
>hours = int(string_value)

>hours = int(input('How many hours did you work? '))

>num = int(input('Read a number from keyboard'))

> m1=int(input("enter value of m1"))
>print (m1)

>pay_rate = float(input('What is your hourly pay rate? '))
```

```
# Get the user's name, age, and income.

name = input('What is your name? ')
age = int(input('What is your age? '))
income = float(input('What is your income? '))

# Display the data.
print('Here is the data you entered:')
print('Name:', name)
print('Age:', age)
print('Income:', income)
```

#Get three test scores and assign them to the test1, test2, and test3 variables.

```
test1 = float(input('Enter the first test score: '))
test2 = float(input('Enter the second test score: '))
test3 = float(input('Enter the third test score: '))

# Calculate the average of the three scores and assign the result to the average variable.
average = (test1 + test2 + test3) / 3.0
# Display the average.
print('The average score is', average)
```

Displaying Multiple Items with the + Operator

+ operator is used with two strings, it performs string concatenation.

This means that it appends one string to another.

```
> print('This is ' + 'one string.')
```

O/P : This is one string.

```
> print('Enter the amount of ' + \
'sales for each day and ' + \
'press Enter.')
```

O/p : Enter the amount of sales for each day and press Enter.

Python Import

- When our program grows bigger, it is a good idea to break it into different modules.
- A module is a file containing Python definitions and statements.
- Python modules have a filename and end with the extension .py.
- Definitions inside a module can be imported to another module or the interactive \ interpreter in Python. We use the import keyword to do this.

For example, we can import the math module by typing in import math.

Example

```
>import math
print(math.pi)
```

```
>import math
 math.sqrt(5)
```

```
>import sys
 print(sys.version)
```

```
>import time
time.ctime()
```

```
> import keyword
   print(keyword.kwlist)
```

```
>import os
os.system("notepad")
```

```
> import os
  os.system("cls")
```

```
>import os
  os.mkdir(" Python Stuff")
```

Built-in function : id( )

id (object)

It returns identity of an object.

It is the address of object in memory

It will be unique and constant throughout the lifetime of an object

Ex:

```
a=20
b=a
print (“value of a and b before incrementation”)
print(“id of a:”,id(a))
print(“id of b:”,id(b))
b=a+1
print(“Value of a and b after incrementation”)
print(“id of b: “ , id(b))
```

Built-in function :getsizeof( )

```
# size
import sys
x=1   # int 28
print(sys.getsizeof(x))
z=0.23
print(sys.getsizeof(z))
y=''
print(sys.getsizeof(y))
r=""
s="Ravikanth"
print(sys.getsizeof(r))
#r=[8]
#print(sys.getsizeof(r))
```

Type Conversion

- In Python, it is just not possible to complete certain operations that involves different types of data. For example, it is not possible to perform "2" + 4 since one operand is an integer and the other is of string type.

Example:

```
>>>"20"+"30"
'2030'
```

```
>>> int("2") + int("3")
5
```

Function	Description
int(x)	Converts x to an integer
long(x)	Converts x to a long integer
float(x)	Converts x to a floating point number
str(x)	Converts x to a string
tuple(x)	Converts x to a tuple
list(x)	Converts x to a list
set(x)	Converts x to a set
ord(x)	Converts a single character to its integer value
oct(x)	Converts an integer to an octal string
hex(x)	Converts an integer to a hexadecimal string
chr(x)	Converts an integer to a character
unichr(x)	Converts an integer to a Unicode character
dict(x)	Creates a dictionary if x forms a (key-value) pair

Built-in function : type( )

It is usedto identity the type of the object

Ex:

```
int_a =10
print("Type of 'int_a':",type(int_a))
```

Output:

Type of 'int_a':<class 'int'>

```
str_b="Hello"
print("Type of 'str_b':", type(str_b))
```

Type of 'str_b':<class 'str'>

```
list_c=[ ]
print("Type of 'list_c':",type(list_c))
```

Type of 'list_c':<class 'list'>

Type Casting vs Type Coercion

- we have done explicit conversion of a value from one data type to another. This is known as ***type casting*.**
- However, in most of the programming languages including Python, there is an implicit conversion of data types either during compilation or during run-time. This is also known ***type coercion*.**
- For example, in an expression that has integer and floating point numbers such as **(21 + 2.1 gives 23.1),** the python interpreter will automatically convert the integer into floating point number so that fractional part is not lost.

Operations on Strings

When a string is multiplied with an integer n, the string is repeated n times. The * operator is also known as String Repetition Operator

Examples:

```
>>> print("Missile Man of India" + " - Sir APJ Abdul Kalam")
Missile Man of India - Sir APJ Abdul Kalam
```

```
>>> print("Hello " * 5)
Hello Hello Hello Hello Hello
```

```
str = 'Hello '
print(str + '4')
print(str * 5)

OUTPUT
Hello 4
Hello Hello Hello Hello Hello
```

Python Basics: Comments

- Comments are the non-executable statements in a program. They are just added to describe the statements in the program code. Comments make the program easily readable and understandable by the programmer as well as other users who are seeing the code. The interpreter simply ignores the comments.
- In Python, a hash sign (#) that is not inside a string literal begins a comment. All characters following the # and up to the end of the line are part of the comment

Python Comments

Comments are very important while writing a program. It describes what's going on inside a program so that a person looking at the source code does not have a hard time figuring it out. In Python, we use the hash (#) symbol to start writing a comment.

It extends up to the newline character. Comments are for programmers for better understanding of a program. Python Interpreter ignores comment.

Example

```
#This is a comment
#print out Hello
print('Hello')
```

Multi-line comments

If we have comments that extend multiple lines, We can use triple quotes, either ''' or """. These triple quotes are generally used for multi-line strings. But they can be used as multi-line comment as well.

Example

```
"""This is also a
perfect example of
multi-line comments"""
```

Example:

```
# This is a comment
print("Hello")  # to display hello
# Program ends here

OUTPUT

Hello
```

Python Basics: Reserved Words

Keywords in Python

Keywords are the reserved words in Python. We cannot use a keyword as variable name, function name or any other identifier. They are used to define the syntax and structure of the Python language.

In Python, keywords are case sensitive. There are 33 keywords in Python. All the keywords except True, False and None are in lowercase and they must be written as it is. The list of all the keywords are given below.

Keywords in Python programming language

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

Python Basics: Indentation

- Whitespace at the beginning of the line is called indentation. These whitespaces or the indentation are very important in Python. In a Python program, the leading whitespace including spaces and tabs at the beginning of the logical line determines the indentation level of that logical line.

Python Indentation

Most of the programming languages like C, C++, Java use curly braces { } to define a block of code. Python uses indentation.

A code block (body of a function, loop etc.) starts with indentation and ends with the first unindented line. The amount of indentation is up to you, but it must be consistent throughout that block.

Generally four whitespaces are used for indentation and is preferred over tabs.

The delimiter followed in python is a **colon (:)** and indented spacesor tabs

Example

```
for i in range(1,11):
    print(i)
```

print a table of perfect squares

```
>>> for n in [1,2,3,4,5,6,7,8,9,10]:
        print (n**2)


1
4
9
16
25
36
49
64
81
100
```

print values 1 to 10 in sequence

```
>>> for i in range(1,11):
        print(i)
1
2
3
4
5
6
7
8
9
10
```

print values and break at 5

```
>>> for i in range(1,10):
        print(i)
    `````if i==5:
            break
    1
    2
    3
    4
    5
```
```

Example

```
age = 21
    print("You can vote") # Error! Tab at the start of the line
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 2
    print("You can vote")
    ^
IndentationError: unexpected indent
```

Operators and Expressions:

Arithmetic
Comparison
Assignment
Relational
Unary
Bitwise
Shift
Logical
Membership
Identity
Operator Precedence and Associativity
Expressions

Python Operators

Operators are special symbols in Python that carry out arithmetic or logical computation. The value that the operator operates on is called the operand.

Operators are used to perform specific operations on one more operands(variables) and provides a result

Arithmetic operators
Comparison / Relational Operators
Logical Operators
Assignment Operators
Bitwise Operators
Unary Operators
Membership Operators
Identity Operators

Arithmetic Operators

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc.

Arithmetic operators in Python

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)

Operator	Description	Example	Output
+	Addition: Adds the operands	`>>> print(a + b)`	`300`
-	Subtraction: Subtracts operand on the right from the operand on the left of the operator	`>>> print(a – b)`	`-100`
*	Multiplication: Multiplies the operands	`>>> print(a * b)`	`20000`
/	Division: Divides operand on the left side of the operator with the operand on its right. The division operator returns the quotient.	`>>> print(b / a)`	`2.0`
%	Modulus: Divides operand on the left side of the operator with the operand on its right. The modulus operator returns the remainder.	`>>> print(b % a)`	`0`

//	Floor Division: Divides the operands and returns the quotient. It also removes the digits after the decimal point. If one of the operands is negative, the result is floored (i.e.,rounded away from zero towards negative infinity).	`>>> print(12//5)`
		`>>> print( 12.0//5.0)`
		`>>> print(-19//5)`
		`>>> print(-20.0//3)`
			`2`
			`2.0`
			`-4`
			`-7.0`

**	Exponent: Performs exponential calculation, that is, raises operand on the right side to the operand on the left of the operator.	`>>> print(a**b)`	100^{200}

Ex:

```
x = 15
y = 4
print('x + y =',x+y)
print('x - y =',x-y)
print('x * y =',x*y)
print('x / y =',x/y)
Print('x % y =',x%y)
print('x // y =',x//y)
print('x ** y =',x**y)
```

O/p:

```
x + y = 19
x - y = 11
x * y = 60
x / y = 3.75
x %y = 3
x // y = 3
x ** y = 50625
```

Relational / Comparison operators

Comparison operators are used to compare values. It either returns True or False according to the condition. Used in conditional Statements

Comparision operators in Python

Operator	Meaning	Example
>	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

Relational / Comparison operators

Operator	Description	Example	Output
==	Returns True if the two values are exactly equal.	>>> print(a == b)	False
!=	Returns True if the two values are not equal.	>>> print(a != b)	True
>	Returns True if the value at the operand on the left side of the operator is greater than the value on its right side.	>>> print(a > b)	False
<	Returns True if the value at the operand on the right side of the operator is greater than the value on its left side.	>>> print(a < b)	True
>=	Returns True if the value at the operand on the left side of the operator is either greater than or equal to the value on its right side.	>>> print(a >= b)	False
<=	Returns True if the value at the operand on the right side of the operator is either greater than or equal to the value on its left side.	>>> print(a <= b)	True

- **Example :**

```
x = 10
y = 12
print('x > y is',x>y)
print('x < y is',x<y)
```

Output:

```
('x > y is', False)
('x < y is',True)
```

Logical Operators

- Logical AND (&&) operator is used to simultaneously evaluate two conditions or expressions with relational operators. If expressions on both the sides (left and right side) of the logical operator are true, then the whole expression is true. For example, If we have an expression (a>b) && (b>c), then the whole expression is true only if both expressions are true. That is, if b is greater than a and c.
- Logical OR (||) operator is used to simultaneously evaluate two conditions or expressions with relational operators. If one or both the expressions of the logical operator is true, then the whole expression is true. For example, If we have an expression (a>b) || (b>c), then the whole expression is true if either b is greater than a or b is greater than c.
- Logical not (!) operator takes a single expression and negates the value of the expression. Logical NOT produces a zero if the expression evaluates to a non-zero value and produces a 1 if the expression produces a zero. In other words, it just reverses the value of the expression. For example, a = 10, b b = !a; Now, the value of b = 0. The value of a is not zero, therefore, !a = 0. The value of !a is assigned to b, hence, the result.

Logical operators

Logical operators are the and, or, not operators.

Based on Boolean Algebra

Returns results as either True or False

Logical operators in Python

Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

Example

```
x = True
y = False

# Output: x and y is False
print('x and y is',x and y)
# Output: x or y is True
print('x or y is',x or y)
# Output: not x is False
print('not x is',not x)
```

Assignment operators

Assignment operators are used in Python to assign values to variables. a = 5 is a simple assignment operator that assigns the value 5 on the right to the variable a on the left.

Multiple Assignments: Same value can be assigned to more than one variable

Operator	Example	Equivatent to
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
|=	x |= 5	x = x | 5
^=	x ^= 5	x = x ^ 5
>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

Bitwise Operators

- It performs bit by bit operation on bits

Operators	Description
&	Binary AND
|	Binary OR
^	Binary XOR
~	Binary Ones Complement
<<	Binary Left Shift
>>	Binary Right Shift

Bitwise Operators

- The truth tables of these bitwise operators are given below.

A	B	A&B	A	B	A|B	A	B	A^B	A	!A
0	0	0	0	0	0	0	0	0	0	1
0	1	0	0	1	1	0	1	1	1	0
1	0	0	1	0	1	1	0	1		
1	1	1	1	1	1	1	1	0		

Shift Operators

- Python supports two bitwise shift operators. They are shift left (<<) and shift right (>>).
- These operations are used to shift bits to the left or to the right.
- The syntax for a shift operation can be given as follows:

Examples:

```
if we have x = 0001 1101, then
x << 1 gives result = 0011 1010
```

```
if we have x = 0001 1101, then
x << 4 gives result = 1010 0000
```

```
if we have x = 0001 1101, then
x >> 1 gives result = 0000 1110.
Similarly, if we have x = 0001 1101 then
x << 4 gives result = 0000 0001
```

Unary Operators

- Unary operators act on single operands. Python supports unary minus operator. Unary minus operator is strikingly different from the arithmetic operator that operates on two operands and subtracts the second operand from the first operand. When an operand is preceded by a minus sign, the unary operator negates its value.
- For example, if a number is positive, it becomes negative when preceded with a unary minus operator. Similarly, if the number is negative, it becomes positive after applying the unary minus operator. Consider the given example.

  b = 10 a = -(b)

- The result of this expression, is a = -10, because variable b has a positive value. After applying unary minus operator (-) on the operand b, the value becomes -10, which indicates it as a negative value.

Special operators

Python language offers some special type of operators like the identity operator or the membership operator.

Membership operators

- in and not in are the membership operators in Python.

- They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).

Operators	Description	Example
in	Returns to true if it finds a variable in given sequence else false	5 in x
not in	Returns to true if it does not find a variable in given sequence else false	5 not in x

Identity Operators

- is and is not are the identity operators in Python.
- They are used to check if two values (or variables) are located on the same part of the memory. [used to compare memory locations of 2 objects]
- Two variables that are equal does not imply that they are identical.

Identity operators in Python		
Operator	**Meaning**	**Example**
is	Returns to true if variable on either side of operator are referring to same object else false	x is True
is not	Returns to false if variable on either side of operator are referring to same	x is not True

Example

```
x1 = 5                  y1 = 5
x2 = 'Hello‘            y2 = 'Hello'
x3 = [1,2,3]            y3 = [1,2,3]
# Output: False
print(x1 is not y1)
# Output: True
print(x2 is y2)
# Output: False
print(x3 is y3)
```

Here, we see that x1 and y1 are integers of same values, so they are equal as well as identical. Same is the case with x2 and y2 (strings).

But x3 and y3 are list. They are equal but not identical. Since list are mutable (can be changed), interpreter locates them separately in memory although they are equal.

Example

```
x = 'Hello world'
y = {1:'a',2:'b'}
# Output: True
print('H' in x)
# Output: True
print('hello' not in x)
# Output: True
print(1 in y)
# Output: False
print('a' in y)
```

Here, 'H' is in x but 'hello' is not present in x (remember, Python is case sensitive). Similarly, 1 is key and 'a' is the value in dictionary y. Hence, 'a' in y returns False.

Python Operators Precedence

The following table lists all operators from highest precedence to lowest.

Operator	Description
( )	Parentheses
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

NOTE: There is an exception to the left-to-right rule. When two ** operators share an operand, the operators execute right-to-left. For example, the expression 2**3**4 is evaluated as 2**(3**4).

Expressions

- An **expression** is any legal combination of symbols (like variables, constants and operators) that represents a value. In Python, an expression must have at least one operand (variable or constant) and can have one or more operators. On evaluating an expression, we get a value. *Operan*d is the value on which operator is applied.
- ***Constant Expressions*:** One that involves only constants. **Example: 8 + 9 – 2**
- ***Integral Expressions:*** One that produces an integer result after evaluating the expression. **Example: a = 10**
- ***Floating Point Expressions:*** One that produces floating point results. **Example: a * b / 2**
- ***Relational Expressions:*** *One that returns either true or false value.* **Example: c = a>b**
- ***Logical Expressions:*** One that combines two or more relational expressions and returns a value as *True* or *False*. **Example: a>b && y! = 0**
- ***Bitwise Expressions****:* One that manipulates data at bit level. **Example: x = y&z**
- ***Assignment Expressions****:*One that assigns a value to a variable. **Example:c = a + b or c = 10**

```
a = 20
b = 10
c = 15
d = 5
e = 0

e = (a + b) * c / d #( 30 * 15 ) / 5
print ("Value of (a + b) * c / d is ", e )

e = ((a + b) * c) / d # (30 * 15 ) / 5
print ("Value of ((a + b) * c) / d is ", e )

e = (a + b) * (c / d); # (30) * (15/5)
print ("Value of (a + b) * (c / d) is ", e)

e = a + (b * c) / d; # 20 + (150/5)
print ("Value of a + (b * c) / d is ", e )
```

Output

```
Value of (a + b) * c / d is  90
Value of ((a + b) * c) / d is  90
Value of (a + b) * (c / d) is  90
Value of a + (b * c) / d is  50
```

Decision Control Statements:

Selection/Conditional Branching Statements –

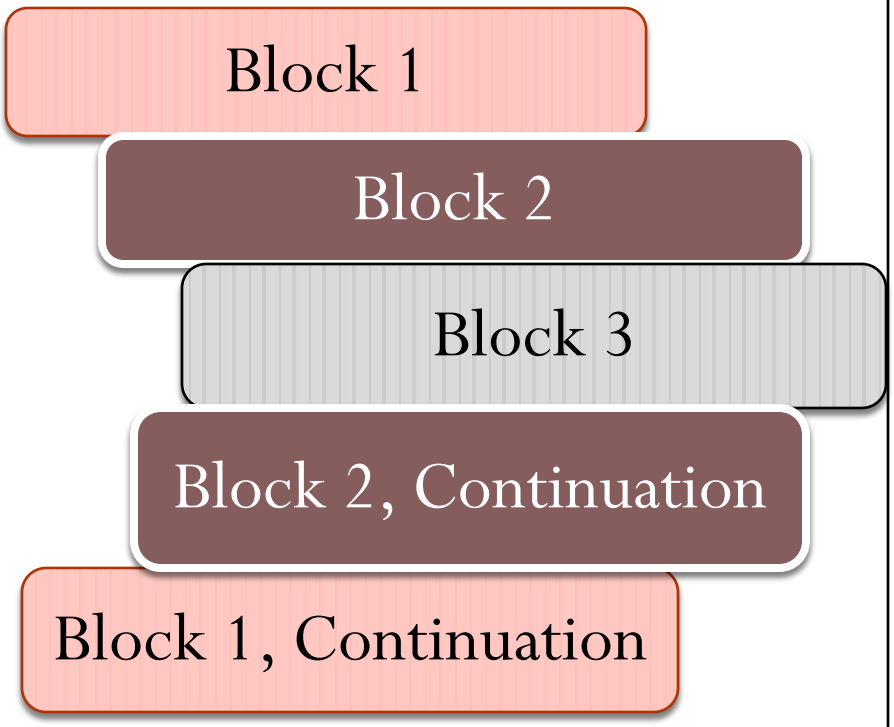
if
if-else
Nested if
if- elif-else statements

Control Structures/ Control Statements

- A control structure is a logical design that controls the order in which a set of statements execute.

(or)

- A control statement is a statement that determines the control flow of a set of instructions, i.e., it decides the sequence in which the instructions in a program are to be executed.



Control Statements

- **Types of Control Statements :**

- **Sequential Control:**

  A Python program is executed sequentially from the first line of the program to its last line.

- **Selection Control:**

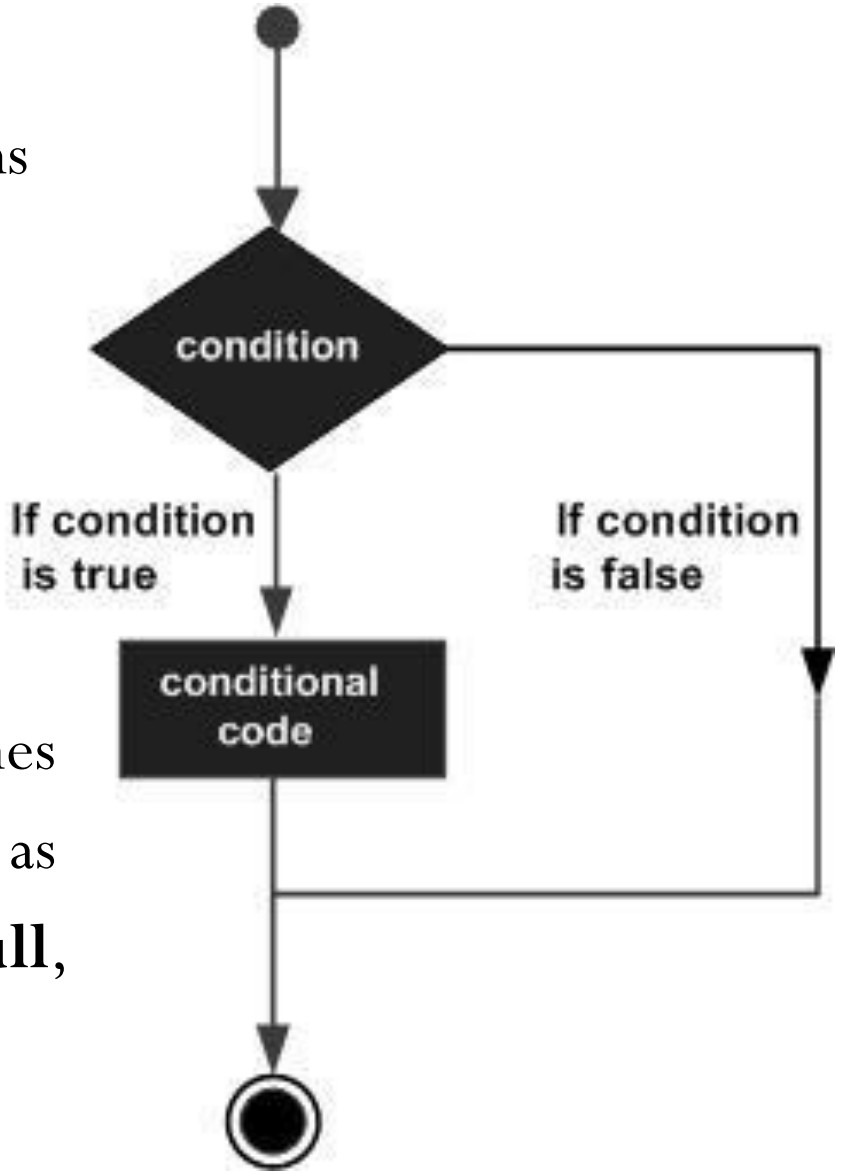
  To execute only a selected set of statements.

- **Iterative Control:**

  To execute a set of statements repeatedly.

- **Decision Making Statements**

  - executes the given set of instructions and returns either **TRUE or FALSE** based on the evaluating the expression



- Python programming language assumes any **non-zero** and **non-null** values as **TRUE**, and if it is either **zero** or **null**, then it is assumed as **FALSE** value.

Python programming language provides following types of decision making statements.

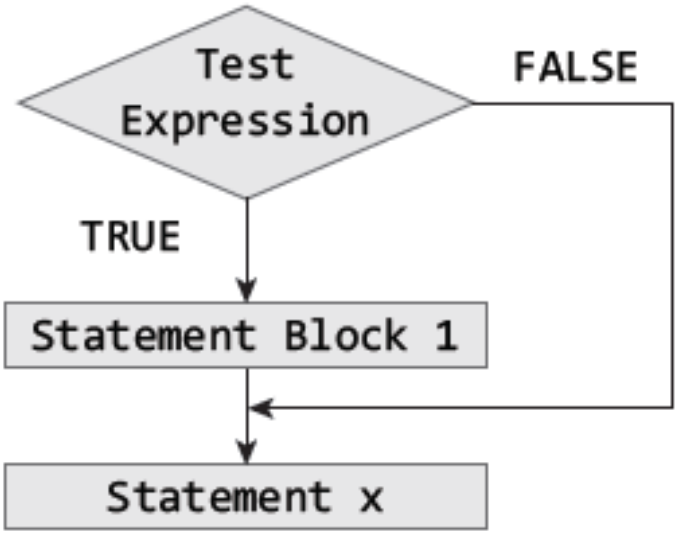
Statement	Description
if statements	if statement consists of a boolean expression followed by one or more statements.
if...else statements	if statement can be followed by an optional else statement, which executes when the boolean expression is FALSE.
nested if statements	You can use one if or else if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

If Statement

- If stmt checks for a condition and if that is found True a particular set of instructions gets executed
- **Syntax**

```
if test_expression:
        statement(s)

        statement (n)
statement x;
```



Explanation :

If the boolean expression evaluates to TRUE, then the block of statement(s) inside the if statement is executed.

If boolean expression evaluates to FALSE, then the first set of code after the end of the if statement(s) is executed.

Example:

```
>>>x=8
        if x<10:
              print("Value of x is %d" %x)        # Check if value of x=15
```

O/P: value of x is 5

```
>>>var =10
        if var>5:
               print("Hi")
        print("Iam out of if")
```

O/p: Hi

Iam out of if

```
# If the number is positive, we print an appropriate message
num = 3
if num > 0:
   print(num, "is a positive number.")
   print("This is always printed.")

num = -1
if num > 0:
   print(num, "is a positive number.")
print ("This is also always printed.")
```

O/P: ???

Example:

```
x = 10          #Initialize the value of x
if(x>0):        #test the value of x
    x = x+1       #Increment the value of x if it is > 0
print(x)        #Print the value of x
```

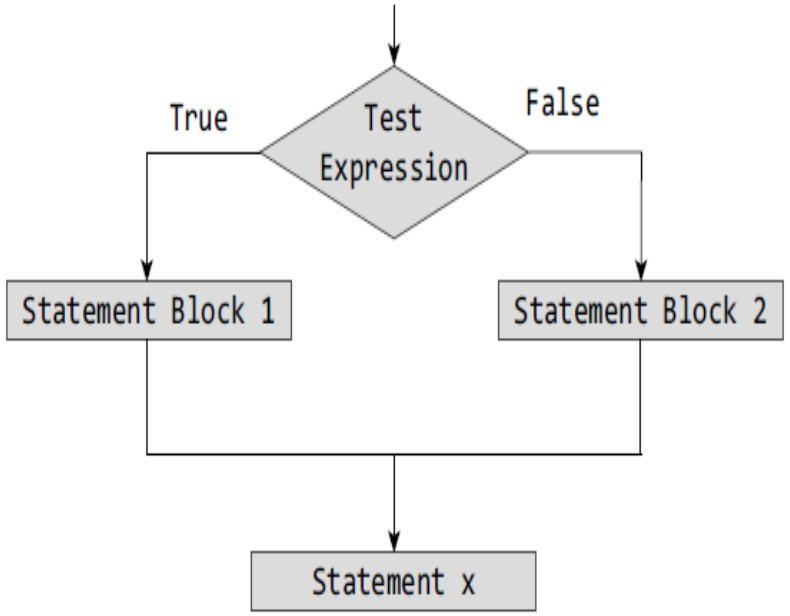
OUTPUT

```
x = 11
```

If…else Statement

- **Syntax**

```
if test expression:
        Body of if  (ST B1)
else:
        Body of else (ST B2)
statement x
```



Explanation:

- The if..else statement evaluates test expression and will execute body of if only when test condition is True.

If the condition is False, body of else is executed.

Indentation is used to separate the blocks.

Example:

```
x=5
if x>2:
    print ("True")
    print( "value of x is ",x)
else:
    print("False")
print("out of if Block")
```

O/P:

```
True
value of x is  5
out of if Block
```

WAP to check given year is Leap or not

```
year=int(input('enter the year:'))
if(year%4==0):
        print("It is a Leap Year")
else:
        print("It is not a Leap Year")
```

Ex: # To check whether Even or Odd

```
 num=int(input('Read the Number:'))
if((num%2)==0):
    print("Num is Even")
else:
    print("Num is Odd")
```

```
# Program checks if the number is positive or negative
# And displays an appropriate message

num = 3
# Try these two variations as well.
# num = -5
# num = 0
if num >= 0:
                print("Positive or Zero")
else:
                print("Negative number")
```

WAP to find largest number among the two numbers

```
a=20
b=30
if  a>b:
   print("x is greater")
else:
    print("y is greater")
```

Example:

```
age=input("Enter your age")
if(age >= 18):
        print("You are eligible to vote")
else:
        years=18-age print("you have to wait for {} years to cast\
        your vote".format(years))
```

Output:

Enter your age12
you have to wait for 6 years to cast your vote
Enter your age23
You are eligible to vote

Nested if Statements

- A statement that contains other statements is called a compound statement.
- To perform more complex checks, if statements can be nested, that is, can be placed one inside the other.
- In such a case, the inner if statement is the statement part of the outer one.
- Nested if statements are used to check if more than one conditions are satisfied.

Example:

```
num = int(input("Enter any number from 0-30: "))
if(num>=0 and num<10):
            print("It is in the range 0-10")
elif(num>=10 and num<20):
            print("It is in the range 10-20")
elif(num>=20 and num<30):
            print("It is in the range 20-30")
```

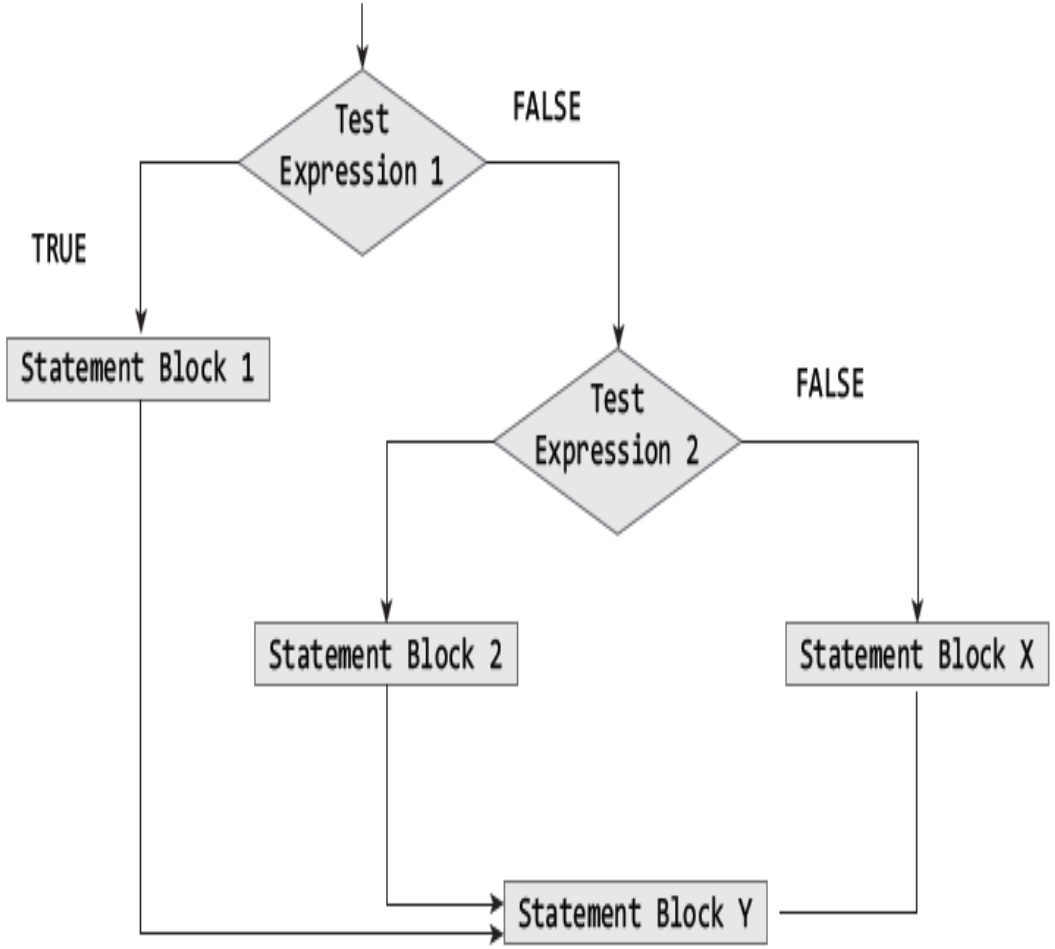
OUTPUT

```
Enter any number from 0-30: 25
It is in the range 20-30
```

if…elif…else Statement

Syntax

```
if test expression1:
        Body of if
elif test expression2:
        Body of elif
elif test expression3:
        Body of elif
else:
        Body of else
```



Explanation:

- The elif is short for else if. It allows us to check for multiple expressions.If the condition for if is False, it checks the condition of the next elif block and so on.
- If all the conditions are False, body of else is executed.Only one block among the several if...elif...else blocks is executed according to the condition.The if block can have only one else block. But it can have multiple elif blocks.
- Python does not provide **switch or case statements** as in other languages, but we can use if..elif...statements to simulate switch case

Example:

```
var = 10   # check for diff var values
if var > 10:
    print("Hello")
    print(var)
elif var < 10:
    print(" Hello Hai Hello ")
    print (var)
else:
    print("Hai")
    print(var)
print("This is the End of Program, Bye!")
```

Ex:

```
a=10
if a>=20:
   print "Condition is True"
else:
   if a>=15:
      print "Checking second value"
   else:
      print "All Conditions are false"
```

WAP to find greater value among 3 numbers

```
x=10, y=20,z=30
if x>y:
    if x>z:
        print("x is grater")
    else:
        if y>z:
            print("y is greater")
        else:
            print(" z is greater")
```

```
# In this program, we check if the number is positive or negative or zero and  display an appropriate message

num = 3.4
if num > 0:
                print("Positive number")
elif num == 0:
                print("Zero")
else:
                print("Negative number")
```

```
#Python Program to Check if a Number is Odd or Even

num = int(input("Enter a number: "))
if (num % 2) == 0:
                print("{0} is Even".format(num))
else:
                print("{0} is Odd".format(num))
```

```
#Python Program to Check Leap Year for year = 2000
# To get year (integer input) from the user
year = int(input("Enter a year: "))
if (year % 4) == 0:
    if (year % 100) == 0:
        if (year % 400) == 0:
                print("{0} is a leap year".format(year))
                    else:
                            print("{0} is not a leap
                                                year".format(year))
                else:
                            print("{0} is a leap year".format(year))
else:
        print("{0} is not a leap year".format(year))
```

#Write a program to display calculator operations(addition, subtraction, #multiplication and division) for taking any two numbers thorough the #keyboard.

```
print("+: ADDITION\n-:SUBTRACTION\n*: MULTIPLICATION\n/: DIVISION")
a =input("Enter the value of a:")
b =input("Enter the value of b:")
CHOICE = raw_input("Enter the operator:")
if CHOICE == "+":
        print ("Addition of {} and {} is: {}".format(a,b,a+b))
elif CHOICE == "-":
        print ("Subtraction of {} and {} is: {}".format(a,b,a-b))
elif CHOICE == "*":
        print ("multiplication of {} and {} is: {}".format(a,b,a*b))
elif CHOICE == "/":
        print ("Division of {} and {} is: {}".format(a,b,float(a)/float(b)))
else:
        print ("Invalid Number" )
```

Example

```
num = int(input("Enter any number from 0-30: "))
if(num>=0 and num<10):
            print("It is in the range 0-10")
elif(num>=10 and num<20):
            print("It is in the range 10-20")
elif(num>=20 and num<30):
            print("It is in the range 20-30")
```

OUTPUT

```
Enter any number from 0-30: 25
It is in the range 20-30
```

Basic Loop Structures/IterativeStatements:

- while
- for
- Nested loops
- continue
- break
- pass statements

Iterative Statements

- Loop Statements
  - Allows us to execute a statement or group of statements multiple times
    - **While Loop**
    - **For Loop**
    - **Range**
- Loop Control Statements
  - Are used to change flow of execution from its normal sequence
    - **Break**
    - **Continue**
    - **Pass**

while Loop

The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.
We generally use this loop when we don't know beforehand, the number of times to iterate.

Syntax

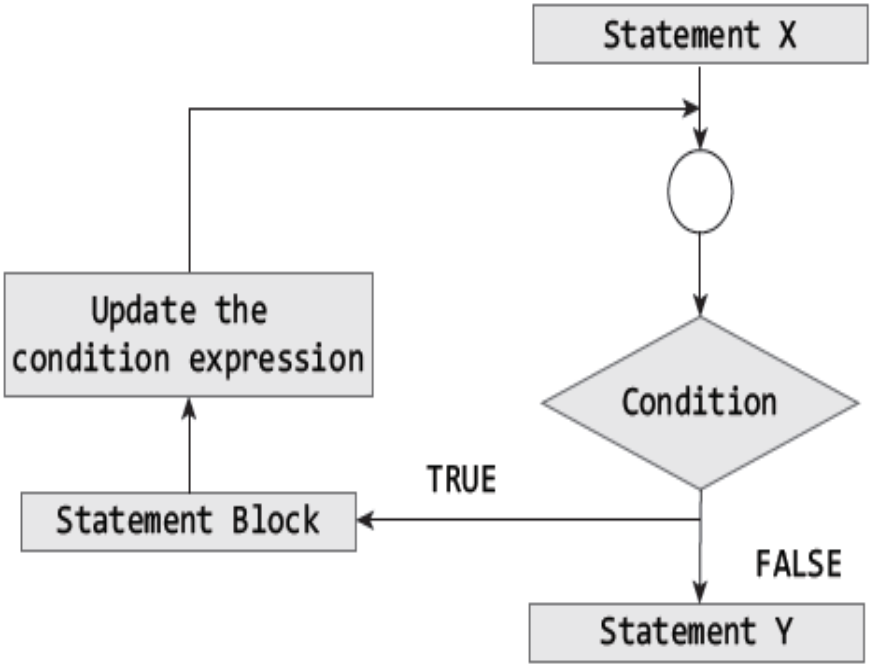
```
while test_expression:
        Body of while
statement (y)
```

In while loop, test expression is checked first. The body of the loop is entered only if the test_expression evaluates to True. After one iteration, the test expression is checked again. This process continues until the test_expression evaluates to False.

In Python, the body of the while loop is determined through indentation. Body starts with indentation and the first unindented line marks the end. Python interprets any non-zero value as True. None and 0 are interpreted as False.

Example

```
n = 10
# initialize sum and counter
sum = 0
i = 1
while i <= n:
        sum = sum + i
        i = i+1
          # update counter
        # print the sum
        print("The sum is", sum)
```



Ex:

```
n=5
result=0
counter=1
while counter<=n:
        result=result+counter
        counter +=1
print(“ sum of 1 until %d:%d” %(n,result))
```

Output: Sum of 1 until 5: 15

```
#Python Program to Check Armstrong Number
# initialize sum
sum = 0
# find the sum of the cube of each digit
temp = num
while temp > 0:
        digit = temp % 10
        sum += digit ** 3
        temp //= 10
# display the result
if num == sum:
        print(num,"is an Armstrong number")
else:
        print(num,"is not an Armstrong number")
```

#Write a program to print your name 10 times.

```
name="RGUKT"
i=0
while i<10:
        print (name)
        i=i+1
```

#Write a program to print numbers from 10 to 0.

```
num=int(input("Enter a valid integer: "))
#num=10
while(num>=0):
        print (num)
        num=num-1
```

#Write a program to print all even numbers below 20 like 2,4,6,.....18,20.(Use if statement before print statement)

```
num=1
while(num<=20):
        if(num%2==0):
                print num
        num=num+1
```

Write a program to print all odd numbers below x.

```
num=int(raw_input("Enter a num to get odd num:  "))
x=1
while(x<=num):
        if(x%2!=0):
                print( x)
        x=x+1
```

- **Example**
- **# Program to display the Fibonacci sequence up to n-th term where n is provided by the user**

```
x,y,i=0,1,0
print( x)
while i<50:
        print(y)
        x,y = y,x+y
        i=i+1
```

- **Python program to check if the number provided by the user is an Armstrong number or not # take input from the user**

```
num = input("Enter a number: ")
 # initialize sum addition = 0
# find the sum of the cube of each digit
temp = num
while temp > 0:
        digit = temp % 10
        addition += digit ** 3
        temp /= 10 # display the result
if num == addition:
        print(num,"is an Armstrong number")
else:
        print(num,"is not an Armstrong number")
```

```
#Python Program to Print the Fibonacci sequence
n1 = 0          # first two ter-ms
n2 = 1
count = 2
# check if the number of terms is valid
if nterms <= 0:
        print("Please enter a positive integer")
elif nterms == 1:
        print("Fibonacci sequence upto",nterms,":")
print(n1)
else:
        print("Fibonacci sequence upto",nterms,":")
        print(n1,",",n2,end=', ')
while count < nterms:
        nth = n1 + n2
        print(nth,end=' , ')
# update values
        n1 = n2
        n2 = nth
        count += 1
```

Example:

```
i = 0
while(i<=10):
    print(i,end=" ")
    i = i+1
```

OUTPUT

```
0 1 2 3 4 5 6 7 8 9 10
```

for Loop

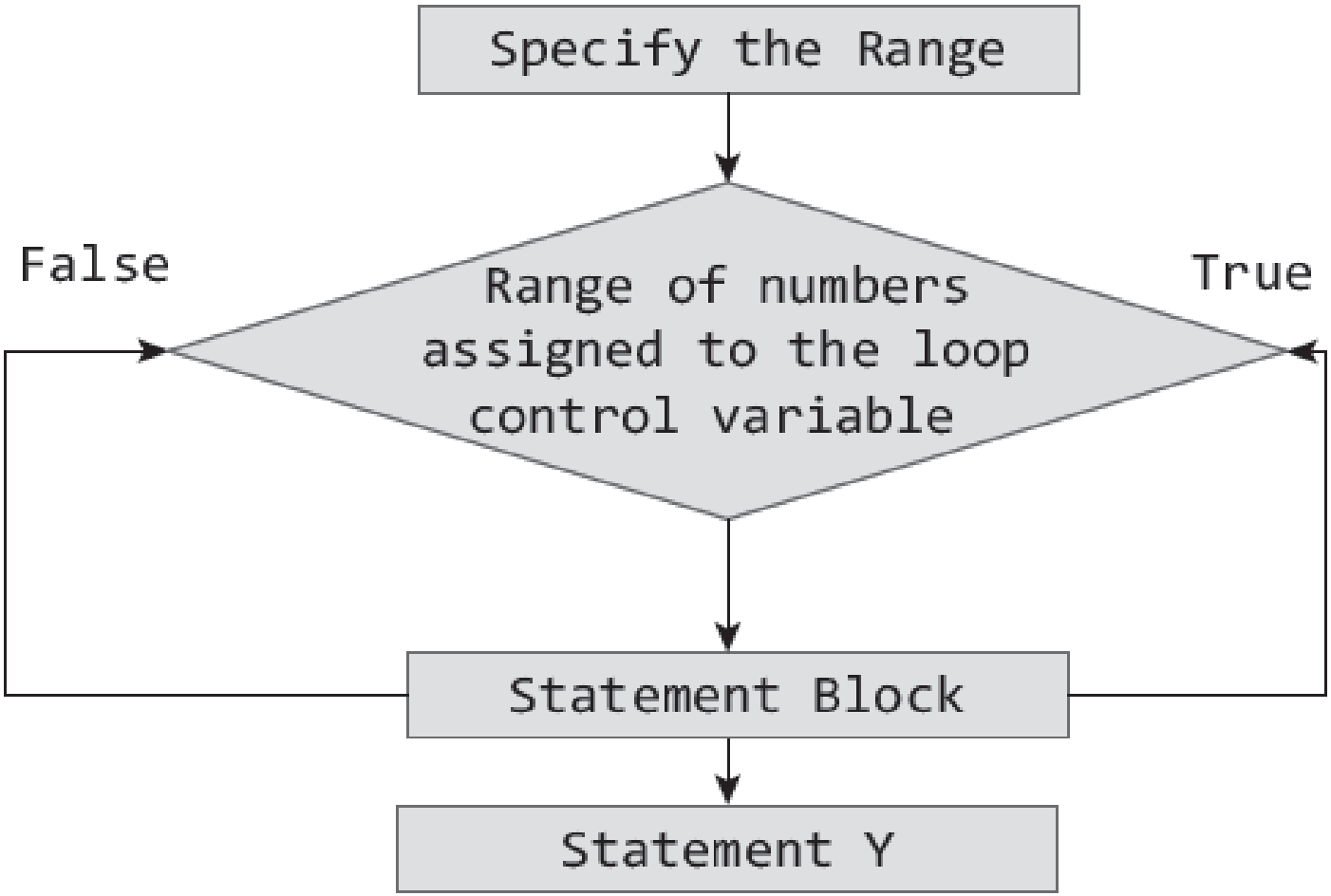
The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects. Iterating over a sequence is called traversal.

Syntax

```
for val in sequence:
        Body of for
```

Here, val is the variable that takes the value of the item inside the sequence on each iteration. Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.

Flow of Statements in for loop



Ex:

```
for counter in 1,2,”RGUKT”, 3,”IIIT”,4:
        print(counter)
```

O/p:

```
1
2
RGUKT
3
IIIT
4
```

print a table of perfect squares

```
>>> for n in [1,2,3,4,5,6,7,8,9,10]:
        print (n**2)


1
4
9
16
25
36
49
64
81
100
```

print values 1 to 10 in sequence

```
>>> for i in range(1,11):
        print(i)
1
2
3
4
5
6
7
8
9
10
```

print values and break at 5

```
>>> for i in range(1,10):
        print(i)
        if i==5:
            break
1
2
3
4
5
```

#write a program to print all numbers below the given number 'x'.

```
num=int(raw_input("Enter a number: "))
for x in range(0,num):
        print (x)
```

 # Write a program to print all the multiples 3, 5 and 7 below 100

```
for i in range(1,10):
        if(i%3==0 and i%5==0 and i%7==0):
                print (i)
```

#Python Program to Find the Factorial of a Number

```
num = 7
# uncomment to take input from the user
#num = int(input("Enter a number: "))
factorial = 1
# check if the number is negative, positive or zero
if num < 0:
        print("Sorry, factorial does not exist for negative
                numbers")
elif num == 0:
        print("The factorial of 0 is 1")
else:
        for i in range(1,num + 1):
                factorial = factorial*i
        print("The factorial of",num,"is",factorial)
```

range() function

We can generate a sequence of numbers using range() function. range(10) will generate numbers from 0 to 9 (10 numbers).

We can also define the start, stop and step size as range(start,stop,[step size]). By default, every number in the range is incremented by 1 but we can specify a different increment using step. It can be both negative and positive, but not zero.
This function does not store all the values in memory, it would be inefficient. So it remembers the start, stop, step size and generates the next number on the go.To force this function to output all the items, we can use the function list().

Examples:

```
for i in range(1, 5):
    print(i, end= " ")
```

Print numbers in the same line

OUTPUT

```
1 2 3 4
```

beg step

```
for i in range(1, 10, 2):
  print(i, end= " ")
```

end

OUTPUT

```
1 3 5 7 9
```

- Range function in loops

Used in case the need is to iterate over a specific number of times within a given range in steps/intervals mentioned

Syntax:

range(lower limit, upper limit, Increment/Decrement by)

Loop	Output	Remarks
for value in range(1,6): print(value)	1 2 3 4 5	Prints all the values in given range exclusive of upper limit
for value in range(0,6,2): print(value)	0 2 4	Prints values in given range in increments of 2
for value in range(6,1,-2): print(value)	6 4 2	Prints values in given range in decrements of 2
for ch in "Hello World": print(ch.upper( ))	H E L L O W O R LD	Prints all the characters in the string converting them to upper case

Example

```
# Output: range(0, 10)
print(range(10))
# Output: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(list(range(10)))
# Output: [2, 3, 4, 5, 6, 7]
print(list(range(2, 8)))
# Output: [2, 5, 8, 11, 14, 17]
print(list(range(2, 20, 3)))
```

Example:

 # Program to iterate through a list using indexing

```
genre = ['pop', 'rock', 'jazz']
# iterate over the list using index
for i in range(len(genre)):
        print("I like", genre[i])
```

Output:

```
('I like', 'pop')
('I like', 'rock')
('I like', 'jazz')
```

Nested Loops

- Python allows its users to have nested loops, that is, loops that can be placed inside other loops. Although this feature will work with any loop like while loop as well as for loop.
- A for loop can be used to control the number of times a particular set of statements will be executed. Another outer loop could be used to control the number of times that a whole loop is repeated.
- Loops should be properly indented to identify which statements are contained within each for statement.

Example:

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

**********

**********

**********

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

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

Example2

```
# Program to iterate through a list using indexing

genre = ['pop', 'rock', 'jazz']
# iterate over the list using index
for i in range(len(genre)):
        print("I like", genre[i])
```

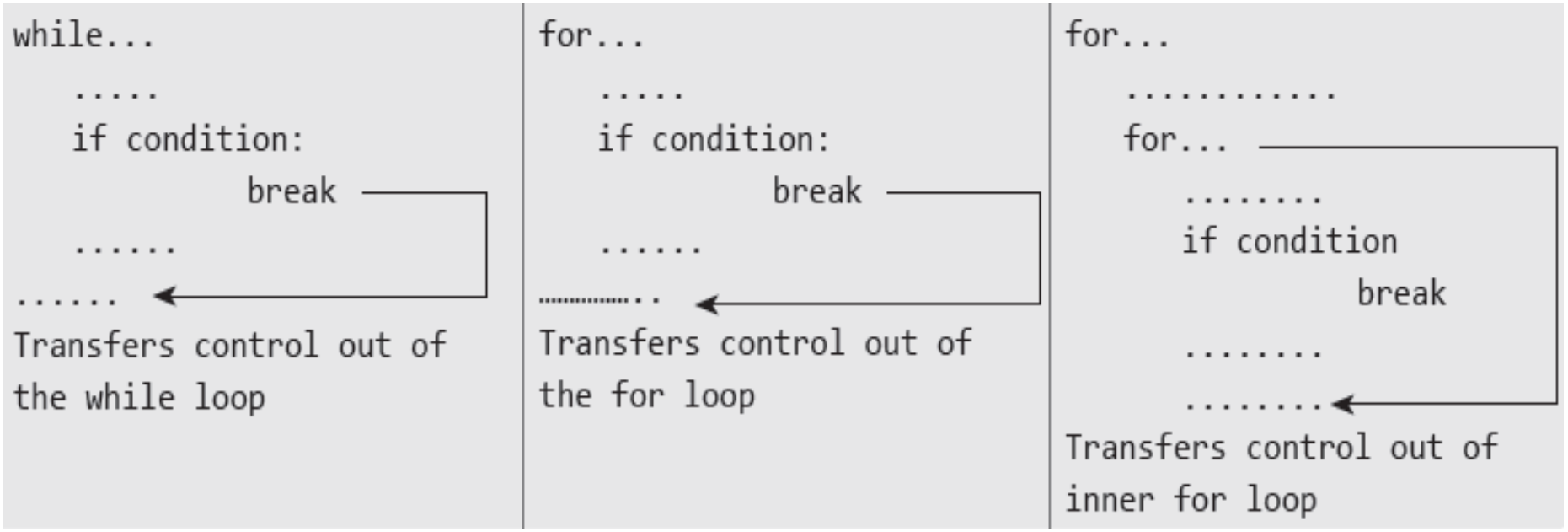
break statement

The break statement is used to terminate the execution of the nearest enclosing loop in which it appears. The break statement is widely used with for loop and while loop. When compiler encounters a break statement, the control passes to the statement that follows the loop in which the break statement appears.

If break statement is inside a nested loop (loop inside another loop), break will terminate the innermost loop.

Syntax :

```
break
```



Example:

```
i = 1
while i <= 10:
        print(i, end=" ")
        if i==5:
                break
        i = i+1
print("\n Done")
```

OUTPUT

```
1 2 3 4 5
Done
```

```
Var=3              #   check if var=5 ?
while var>0:
        print("iam in Iteration", var)
        var-=1
        if var==2:
                break
                print("Iam still in while")
print("iam out of while loop")
```

Output: Iam in Iteration 3

iam out of while loop

 # Python program to check if the input number is prime or not

```
num = 407
# take input from the user
# num = int(input("Enter a number: "))
# prime numbers are greater than 1
if num > 1:
# check for factors
for i in range(2,num):
        if (num % i) == 0:
                print(num,"is not a prime number")
                print(i,"times",num//i,"is",num)
                break
        else:
                print(num,"is a prime number")
# if input number is less than
# or equal to 1, it is not prime
else:
print(num,"is not a prime number")
```

continue statement

Like the break statement, the continue statement can only appear in the body of a loop. When the compiler encounters a continue statement then the rest of the statements in the loop are skipped and the control is unconditionally transferred to the loop-continuation portion of the nearest enclosing loop.

Syntax :

```
continue
```

```
while(...)
    ...
    If condition:
   continue
     ...
...
Transfers control to the condition
expression of the while loop
```

```
for(...)
    ...
    if condition:
        continue
    ...
...
Transfers control to the condition
expression of the for loop
```

```
for(...)
    ...
    for(...)
          ...
          if condition:
                continue
          ...
    ...
Transfers control to the condition
expression of the inner for loop
```

Example:

```
for i in range(1,11):
    if(i==5):
        continue
    print(i, end=" ")
print("\n Done")
```

OUTPUT

```
1 2 3 4 6 7 8 9 10
Done
```

```
var=3
while var>0:
        print("iam in iteration", var)
        var-=1
        if var==2:
                continue
                print("iam still in if block")
        print("iam still in while")
print("iam out of while loop")
```

Output:

```
iam in iteration 3
iam in iteration 2
iam still in while
iam in iteration 1
iam still in while
iam out of while loop
```

Example

Program to show the use of continue statement inside loops

```
for val in "string":
        if val == "i":
                continue
        print(val)
print("The end")
```

Exercise

#Write a program to print n numbers and break the program when n==10.

```
n=input("Enter the value")
for n in range(0,n):
        if(n==10):
                break
        print n
```

#Write a program to print the n numbers and use the continue statement to when n=10.(Note n should be more than 10)

```
n=input("Enter the value:  ")

if(n>=10):
        for n in range(0,n):
                print (n)
elif(n<10):
        for n in range(0,n):
                continue
print ("Continue is used“)
```

•pass statement

Pass statement is used when a statement is required syntactically but no command or code has to be executed. It specified a *null* operation or simply No Operation (NOP) statement.

Nothing happens when the pass statement is executed.

Difference between comment and pass statements In Python programming, pass is a null statement.

The difference between a comment and pass statement is that while the interpreter ignores a comment entirely, pass is not ignored.

Comment is not executed but pass statement is executed but nothing happens.

Example:

```
for letter in "HELLO":
        pass      #The statement is doing nothing
        print("Pass : ", letter)
print("Done")
```

OUTPUT

```
Pass :  H
Pass :  E
Pass :  L
Pass :  L
Pass :  O
Done
```

- Pass statement is never executed
- Used when a statement is required syntactically but do not want any command or code to execute or if the code need to be implemented in future
- Behaves like a placeholder for future scope

Ex: x= " RGUKT"

```
if x = = " RGUKT_BASAR"
    print( " Name", x)
elif x==" RGUKT":
    pass
else:
    print( " in else")
```

Output: No Output

Example

```
# pass is just a placeholder for
# functionality to be added later.

sequence = ['p', 'a', 's', 's']
for val in sequence:
   # print(sequence)
    pass
```

The Else Statement Used With Loops

- Unlike C and C++, in Python you can have the *else* statement associated with a loop statements. If the else statement is used with a *for* loop, the *else* statement is executed when the loop has completed iterating. But when used with the *while* loop, the *else* statement is executed when the condition becomes false.

Examples:

```
for letter in "HELLO":
        print(letter, end=" ")
else:
         print("Done")
```

OUTPUT

```
H E L L O Done
```

```
i = 1
while(i<0):
        print(i)
        i = i - 1
else:
        print(i, "is not negative so
loop did not execute")
```

OUTPUT

```
1 is not negative so loop did not execute
```

Exercise

- 1. Write a program to display your name?(take input from keyboard)?
- 2. Write a program to swap two numbers without using a temporary variable?
- 3. Write a program to calculate salary of an employee given his basic pay(to be entered by the user),HRA=10 percent of basic pay, TA=5 percent of basic pay. Define HRA and TA as constant and use them to calculate the salary of the employee?
- 4. Write a program that calculates number of seconds in a day?
- 5.Test if a number is a multiple of 3,5 or 7.
- 6. Write a program to check which grade the student got marks. If student A marks=300, student B Marks=450 and Student C Marks=550. (Grades 550 marks==A grade, 450 marks==B grade, 300 marks==C grade)

- **Exercise**
- 7. Write a Python program to calculate a dog's age in dog's years.

  Note: For the first two years, a dog year is equal to 10.5 human years. After that, each dog year equals 4 human years.
- 8. Write a program that determies whether a student is eligible for PG course or not. To be eligible, the student must have obtained more than 80% in X and XII examination, and 70% plus marks in graduation. If the stident changes his stream(Science,commerce or arts)then deduct 5% form his graduation score.
- 9 Write a program that accept the current date and the date of birth of the user. Then calncualte the age of the user and display it as dd/mm/yy format on the screen?

- 10. Write a program to clculate parking charges of a vehicle. Enter the type of vehicle as a character (like c for car, b for bus etc..)and number of hours ,then calculate charges as given given below:

Exercise

Truck/Bus-20Rs per hour

car-10Rs per hour

Scooter/Cycle-%Rs per hour

- 11. Write a Python program that reads a month and prints the season for that month?
- 12. Find the income tax to be paid (In Indian Rupees) and the total salary after the income tax deduction as per the details given in below table.

Gross Salary (In Indian Rupees)	Income Tax percentage
Below 5,000	Nil
5,001 to 10,000	10 %
10,001 to 20,000	20%
More than 20,000	30%

Exercise

- 13.Write a program to print your name 10 times.
- 14. Write a program to print numbers from 10 to 0.
- 15. Write a program to print all numbers from x to y using while loop.
- 16.. Write a program to print all even numbers below 20 like 2,4,6,.....18,20.(Use if statement before print statement)
- 17.. Write a program to print all odd numbers below x.
- 18.. Write a program to count all even numbers in between x and y.(read x and y from user).
- 19. Write a program that accept any number and print the number of digits and sum of digits in that number?

Exercise

- 20. Write a program to print all the multiples 3, 5 and 7 below 100.

- 21. Write a program that prints a number, its square, and cube repeatedly in the range?

- 22. Write a program that prompts the user to enter five words. If the length of any word is less than 6 characters , then it asks the user to enter it again. However , if the word is of 6 or more characters, then it displays it on the screen

- 23.Write a program to print n numbers and break the program when n==10.

Exercise

- 24. Write a program that prompts user to enter a number. If the number is equal to 99, print “congratulations”. If the number is less than 99, print enter again and aim higher else print enter again a low number. The program should run until the user guess the correct number that is 99

- 25. Write a program that prompts user to enter number. Once the user enter -1, it displays the count ,sum,and average of even numbers and that of odd numbers

For Details Contact Me @ :

9247448766

ravikanth27787@gmail.com

python

Programming Language