MAR GREGORIOS COLLEGE OF ARTS & SCIENCE

Block No.8, College Road, Mogappair West, Chennai – 37

Affiliated to the University of Madras
Approved by the Government of Tamil Nadu
An ISO 9001:2015 Certified Institution

DEPARTMENT OF COMMERCE COMPUTER APPLICATION

SUBJECT NAME: PYTHON PROGRAMMING

SUBJECT CODE : CZ221

SEMESTER: II

PREPARED BY: PROF. P. UMAESWARI

SYLLABUS

UNIT - I

Computer systems – Python Programming Language Computational Thinking – Python Data Types: Expressions, Operator, Variables, and Assignments – Strings – Lists – Objects & Classes – Python standard library.

UNIT - II

Imperative programming: Python modules – Built-in-function: print() function –eval() function – user-defined function & assignments -parameter passing.

UNIT - III

Text Data, Files & Exceptions: Strings, revisited – formatted output – files – errors & Exceptions – Execution control Structures: decision control & the IF statement

UNIT - IV

For LOOP & Iteration Patterns – two-dimensional list- while loop – more loop patterns – additional iteration control statements – Container and Randomness: Dictionaries – other built-in container types – character encodings & strings – module random.

UNIT - V

Namespaces – encapsulation in functions – global vs local namespaces exceptional flow control – modules as namespaces.

UNIT I

PROGRAMMING LANGUAGE INTRODUCTION

A programming language is a systematic notation by which we describe computational processes to others. Computational process in present context means a set of steps that a machine can perform for solving a problem. Computer science is fundamentally about computational problem solving.

The definition of computer science as computational problem solving begs the question:

What is computation?

One characterization of computation is given by the notion of an *algorithm*. For now, consider an algorithm to be a series of steps that can be systematically followed for producing the answer to a certain type of problem.

We look at fundamental issues of computational problem solving next.

COMPUTATIONAL PROBLEM

What is meant by computational problem?

A computational problem is a mathematical object representing a collection of questions that computers might be able to solve.

The mathematical object is "a collection of questions". A "collection" is another term that represent as a "set". Sets as one kindof mathematical object.

The connection between the two definitions is: each instancerepresents a slightly different question that a computer might be ableto solve.

For example, the problem of factoring

"Given a positive integer n, find a nontrivial prime factor of n."The Essence of Computational

Problem Solving,

In order to solve a problem computationally, two things are needed:

- **Representation** that captures all the relevant aspects of theproblem,
- **Algorithm** that solves the problem by use of the representation.

Limits of Computational Problem Solving

Once an algorithm for solving a given problem is developed or found, an important question is, “Can a solution to the problem be found in a reasonable amount of time?” If not, then the particular algorithm is of limited practical use.

Another problem The Traveling Salesman problem is a classic computational problem in computer science. The problem is to find the shortest route of travel for a salesman needing to visit a given set of cities. In a brute force approach, the lengths of all possible routes would be calculated and compared to find the shortest one. For five cities, the number of possible routes is 5! , Forten cities, the number of possible routes is 10!and so on.

Any algorithm that correctly solves a given problem must solve the problem in a reasonable amount of time, otherwise it is oflimited practical use.

COMPUTER ALGORITHM

What is Algorithm?

An algorithm is a set of instructions designed to perform a specific task. This can be a simple process, such as multiplying two numbers, or a complex operation, such as playing a compressed video file.

For example:Task:

To make a cup of tea.

Algorithm:

Step 1: Add water and milk to the kettle.Step 2: Boil it, add tea leaves.

Step 3: Add sugar, and the serve it in a cup.

What is Computer Algorithm?

Computer algorithms are central to computer science. They provide step-by-step methods of computation that a machine can carry out. Having high-speed machines (computers) that can consistently follow and execute a given set of instructions provides areliable and effective means of realizing computation.

"a set of steps to accomplish or complete a task that is describedprecisely enough that a computer can run it".

Described precisely: It's difficult for a machine to know how much water to be added in the above-mentioned algorithm. These algorithms run on computers or any computational gadgets like GPS,Hangouts etc.

Characteristics of an Algorithm:

- ⇒ Must take an input.
- ⇒ Must give some output.
- ⇒ Definiteness – instructions are clear and unambiguous.
- ⇒ Finiteness – algorithm terminates after a finite number ofsteps.

⇒ Effectiveness – every instruction must be understandable andsimple.

Computer Hardware

Computer hardware comprises the physical part of a computer system. It includes the all-important components of the central processing unit (CPU) and main memory .It also includes peripheral components such as a keyboard, monitor, mouse, and printer.

Fundamental Hardware Components:

Computer hardware is a collection of several components working together. Some parts are essential and others are added advantages. Computer hardware is made up of CPU and peripherals.

The central processing unit (CPU) is the “brain” of a computer system, containing digital logic circuitry able to interpret and execute instructions.

Main memory is where currently executing programs reside, which the CPU can directly and very quickly access.

Main memory is volatile; that is, the contents are lost whenthe power is turned off. In contrast, secondary memory is nonvolatile, and therefore provides long-term storage of programs and data.

This kind of storage, for example, can be magnetic (hard drive), optical (CD or DVD), or nonvolatile flash memory (such as in a USB drive).

Input/output devices include anything that allows for input (such as the mouse and keyboard) or output (such as a monitor or printer). Finally, buses transfer data between components within a computer system, such as between the CPUand main memory.

An operating system acts as the “middle man” between the hardware and executing application program. For example, it controls the allocation of memory for the various programs that may be executing on a computer. Operating systems also provide a particular user interface. Thus, it is the operating system installed on a given computer that determines the “look and feel” of the user interface and how the user interacts with the system, and not the particular model computer.

‘An operating system is software that has the job of managing the hardware resources of a given computer and providing a particular user interface’.

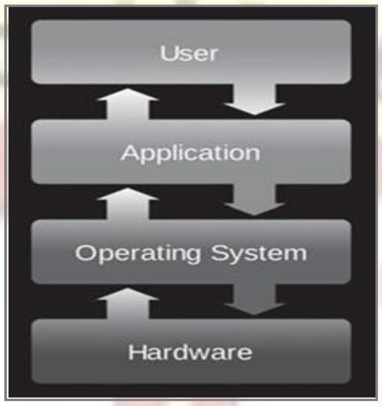
Computer Software

A set of instructions that drives computer to do stipulatedtasks is called a program.

Software instructions are programmed in a computer language, translated into machine language, and executed by computer. Software can be categorized into two types

What is Software?

Software is a logical execution and a set of instructions thatdrives computer to do stipulated tasks is called a program.



What is computer software?

Software instructions are programmed in a computer language, translated into machine language, and executed by computer. Computer software is a set of commands that tells the computer how to work.

Computer software is a set of program instructions, including related data and documentation, that can be executed by computer.

Software can be categorized into two types

I. System Software

II. Application Software

I. System Software

System software is a type of computer program that is designed to run a computer's hardware and application programs. It is the interface between the hardware and user applications.

"System Software is a set of programs that control and manage the operations of computer hardware."

Examples of system software:Windows, Linux etc.

Software generally follows three main parts as

1. Syntax

2. Semantics
3. Program Translation

1. **Syntax**

Programming languages are languages just as “natural languages” such as English. Syntaxand semanticsare important concepts that apply to all languages.

Syntax errors are caused by invalid syntax. The syntax of a language is a set of characters and the acceptable sequences of those characters. English, for example, includes the letters of the alphabet, punctuation, and properly spelled words and sentences.

The following is a syntactically correct sentence in English

Print(“hello world!!”)

The following, however, is not syntactically correct,

“prnt(“hello world”)”

In this sentence, the sequence of word “prnt” is not a word in the English language.

2. **Semantics**

The semantics of a language is a set of characters thatmake meaningful words. Semantic errors are caused by errors in program logic. It is referred as logic errors.

In other words, The semantics of a language is the meaning associated with each syntactically correct sequenceof characters.

Now consider the following sentence, “green sleep furiously.”

This sentence is syntactically correct, but it is semantically incorrect, and thus has no meaning in the sentence.

3. **Program Translation**

A central processing unit (CPU) is designed to interpret and execute a specifi c set of instructionsrepresented in binary form (i.e., 1s and 0s) called machine code.

The source code which is written by the programmer needs to be translated. When translated, the source code becomes object code which is understandable by the computer.

There are three main types of translators as follows

1. Assembler
2. Compilers

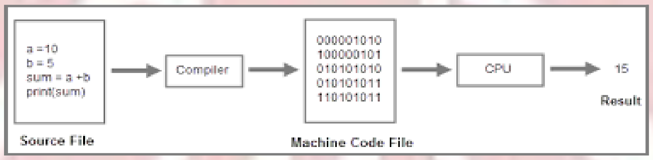
3. Interpreters

The three main types of translators

Type	**Description**
Assembler	Assemblers convert assembly language mnemonicsinto machine code.
Interpreters	Interpreters convert each instruction of the source code into the object code as the program is being run. This gives a better interactive environment butis slower.
Compilers	A compiler converts the entire source code into machine code so that it can be run on the machine without further translation.

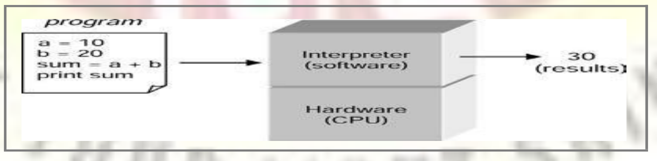
Most of the programs are written in a “high-level” programming language such as Python. Since the instructions of such programs are not in machine code that a CPU can execute, a translator program must be used. There are two fundamental types of translators.

One, called a compiler , it translates programs directly into machine code to be executed by the CPU



Program Execution by a Computer

Another called interpreter, which executes program instruction. Thus, an interpreter can immediately execute instructions as they are entered. This is referred as interactive mode



Interpreter

II. Application Software

An application is any program designed for the end user. It is capable of dealing with user inputs and helps the user to complete the task. Application softwares are installed according to user's requirements. Applications software includes database programs, word processors, browsers and spreadsheets.

Application Software Type	Examples
Word Processing Software	MS Word, WordPad and Notpad
Database Software	Oracle, MS Access etc.
Spreadsheet Software	Apple Numbers, Microsoft Excel
Multimedia Software	Real Player, Media Player

SYSTEM SOFTWARE	APPLICATIONSOFTWARE
The software which provides a platform for the user to interact with the hardware of a computer are known as system software	These software which runs on an operating system(os is a system software) serving specific purpose are called application softwares
System softwares are needed torun application softwares	Application softwares are not needed to run system softwares
Run in the background and act asa platform	Runs in the foreground and interact with the user
Example: language processors, operating system and disk drivers.	**Example:** video players, text editors and browser

Difference between Software and Hardware

Software Vs Hardware

Collection of programs to bringcomputer hardware system	Physical components of computer system
Software products evolve by adding new features to existing programs to support hardware	Hardware design is based on architectural decisions to make it work over a range of environmental conditions and time.
Software cannot be executed without hardware	Hardware cannot perform any task without software.
Software is debugged in case ofproblem	Hardware is repaired in case ofproblem
It includes numbers, alphabets, alphanumeric symbols, identifiers, keywords, etc.	It consists of electronic components like ICs, diodes, registers, crystals, boards, insulators, etc

The Process of Computational Problem Solving

The process of computational problem solving involves understanding the problem, designing a solution and writing the solution. It is a process, with programming being only one of the steps. Before a program is written, a design for the program must be developed. And before a design can be developed, the problem to be solved must be well understood. Once written, the program must bethoroughly tested.

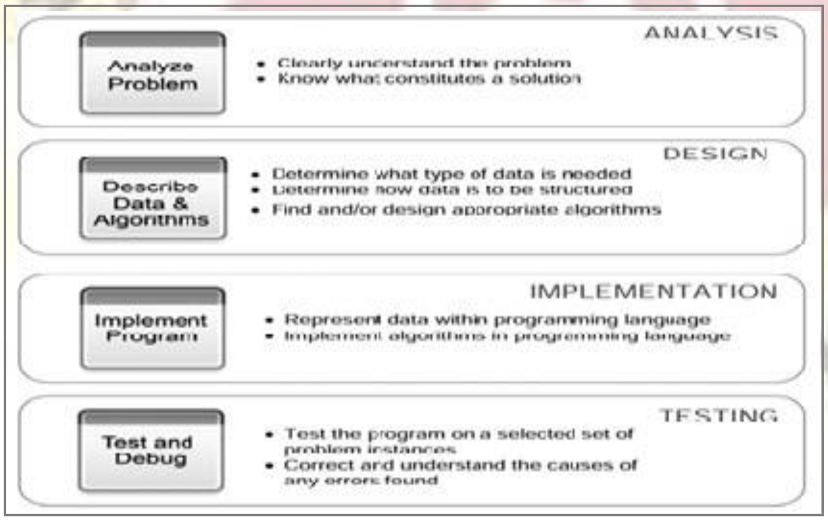

ANALYSIS
Analyze Problem
• Clearly understand the problem
• Know what constitutes a solution
DESIGN
Describe Data & Algorithms
• Determine what type of data is needed
• Determine how data is to be structured
• Find and/or design appropriate algorithms
IMPLEMENTATION
Implement Program
• Represent data within programming language
• Implement algorithms in programming language
TESTING
Test and Debug
• Test the program on a selected set of problem instances
• Correct and understand the causes of any errors found

Fig 1.5 Process of problem solving

1. **Problem Analysis Understanding the Problem**

Once a problem is clearly understood, the fundamental computational issues for solving. For each of the problems the representation is straightforward. For example,

The calendar month problem, need to store the month andyear, the number days in each month, and the names of the days of week. It obtained by *direct calculation* by use of the algorithm.

For (MCGW) problem, need to store the current state of the problem. A brute-force algorithmic approach of trying all possible solutions works very well, and only a relatively small number of steps for reaching a solution.

For both the Traveling Salesman problem and the game of chess, the brute-force approach is infeasible. To understand and finding solutions of the problem. For some problems, there is only one solution. For others, there may be a number (or infinite number) of solutions. Thus, a program may be stated as finding,

- A solution
- An approximate solution
- A best solution
- All solutions

Possible Solutions are

1. Direct Solution
2. Brute-force Solution
3. Clever Solution

2. Program design Data and algorithm

For the MCGW problem, a list can be used to represent the correct location (east and west) of the man, cabbage, goat, and wolf as discussed earlier, reproduced below,

man cabbage goat wolf[W, E, W, E]

For the Calendar Month problem, the data include the month and year (entered by the user), the number of days in eachmonth, and the names of the days of the week. A useful structuring of the data is given below,

[*Month* ,*year*]

[31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

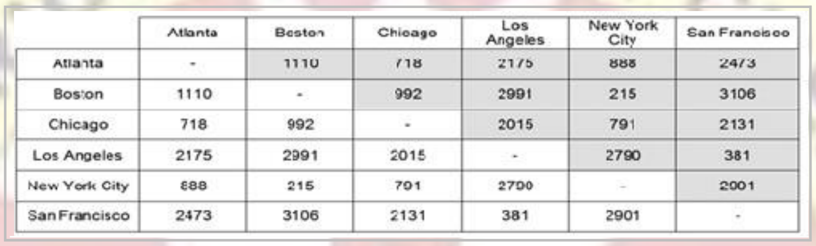
[‘Sunday’, ‘Monday’, ‘Tuesday’, ‘Wednesday’, ‘Thursday’,‘Friday’, ‘Saturday’]

The month and year are grouped in a single list since they are naturally associated. Similarly, the names of the days of the week and the number of days in each month are grouped. Finally, the first day of the month, as determined by the algorithm it can be represented by a single integer,

0 – Sunday, 1 – Monday, . . ., 6 – Saturday

For the Traveling Salesman problem, the distance between each pair of cities must be represented.

One possible way of structuring the data is as a table



	Atlanta	Boston	Chicago	Los Angeles	New York City	San Francisco
Atlanta	-	1110	718	2175	888	2473
Boston	1110	-	992	2991	215	3106
Chicago	718	992	-	2015	791	2131
Los Angeles	2175	2991	2015	-	2790	381
New York City	888	215	791	2790	-	2901
SanFrancisco	2473	3106	2131	381	2901	-

Travelling salesman- data table Describing the Needed Algorithms

When solving a computational problem, either suitable existing algorithms may be found or new algorithms must be developed.

For the MCGW problem, there are standard search algorithms that can be used.

For the calendar month problem, a day of the week algorithm already exists.

For the Traveling Salesman problem, there are various (nontrivial) algorithms that can be utilized.

Finally, for the game of chess, since it is infeasible to look ahead at the final outcomes of every possible move, there are algorithms that make a best guess at which moves to make.

3. **Program implementation**

Decisions are made after analyzing an describing the dataand algorithm. To take decision for the implementation phase in Python programming, the implementation needs to be expressed in a syntactically correct and appropriate way, using the instructions and features available in Python.

4. **Program testing**

Writing com~~puter programs is difficult and challenging. Given this fact, s~~oftware testing is crucial part of softwaredevelopment . testing is done incrementally as a program is being developed, when the program is complete, and when the program needs to be updated. **History of Python:**

❖ Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the

Netherlands.

- ❖ Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.
- ❖ Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).
- ❖ Python is now maintained by a core development team at theinstitute, although Guido van Rossum still holds a vital role in directing its progress.

Definition:

- ❖ Python is a high-level, interpreted, interactive and object- oriented scripting language.
- ❖ Python is designed to be highly readable.
- ❖ It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.
- ❖ Python is a great Software working in Web Development Domain.

Why Learn Python?

- ❖ Python is easy to learn. Its syntax is easy and code is very readable.
- ❖ Python has a lot of applications. It's used for developing webapplications, data science, rapid application development, and so on.
- ❖ Python allows you to write programs in fewer lines of code than most of the programming languages.
- ❖ The popularity of Python is growing rapidly. Now it's one of the most popular programming languages.

Some of the key advantages of Python:

- ❖ Python is Interpreted − Python is processed at run time by the interpreter. You do not need to compile your program beforeexecuting it. This is similar to PERL and PHP.
- ❖ Python is Interactive − You can actually sit at a Python prompt and interact with the interpreter directly to write yourprograms.
- ❖ Python is Object-Oriented − Python supports Object- Oriented style or technique of programming that encapsulates code within objects.
- ❖ Python is a Beginner's Language − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simpletext processing to WWW browsers to games.

Characteristics of Python:

- ❖ It supports functional and structured programming methods as well as OOP.

- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Features of Python:

- 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 visibleto the 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, Windows,and Macintosh.
- Interactive Mode: Python has support for an interactive mode which allows interactive testing and debugging of snippets ofcode.
- 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 orcustomize 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 andwindows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- Scalable: Python provides a better structure and support forlarge programs than shell scripting.

Some other features:

- It supports functional and structured programming methodsas well as OOP.
- It can be used as a scripting language or can be compiled tobyte-code for building large applications.
- It provides very high-level dynamic data types and supportsdynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX,CORBA, and Java.

Applications of Python:

Python is in use since 1991. During this time period, Pythonis used for a variety of software for different purposes. So, let’s have a look at all python application examples that are available in the market.

⇒ Graphical User Interface(GUI)

⇒ Web Frameworks & Applications

⇒ Enterprise and Business Applications

⇒ Operating Systems

⇒ Language Development

⇒ Prototyping

⇒ Software Development Applications

⇒ Console Based Applications

⇒ 3-D CAD Applications

⇒ Applications for Images

Printing and Reading in python
Printing to the Screen:

The simplest way to produce output is using the print statement where you can pass zero or more expressions separated by commas. This function converts the expressions you pass into a string and writes the result to standard output as follows −

Example:

```
print ("Python is really a great language")
```

Output:

```
Python is really a great language
```

Reading Keyboard Input:

Python provides two built-in functions to read a line of text from standard input, which by default comes from the keyboard. These functions are −

❖ raw_input

❖ input

The raw_input Function:

The raw_input([prompt]) function reads one line from standard input and returns it as a string (removing the trailing newline). This prompts you to enter any string and it would displaysame string on the screen.

Example:

```
str = raw_input("Enter your input: ")

print ("Received input is : ", str)
```

Output:

```
Enter your input: Hello Python

Received input is : Hello Python
```

The input Function :

The input([prompt]) function is equivalent to raw_input, except that it assumes the input is a valid Python expression and returns the evaluated result to you.

Example:

```
str = input("Enter your input: ")

print ("Received input is : ", str)
```

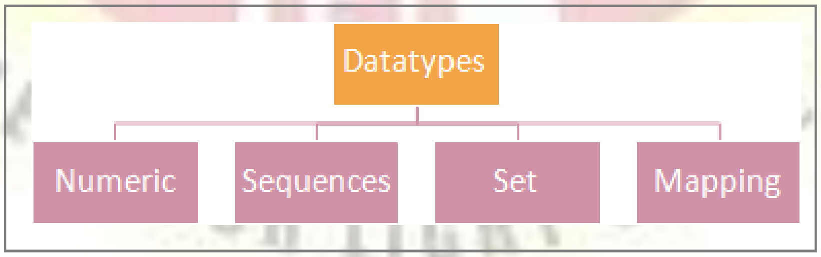
Output:

Enter your input: welcome to python Received input is : welcome to python

Data Types

What is Data Types in python :

Each variable stored in memory has a datatype. Data types are actually classes and if you create any variable of a specific datatype, it is an object. Python's standard data types can be groupedinto mainly four different classes

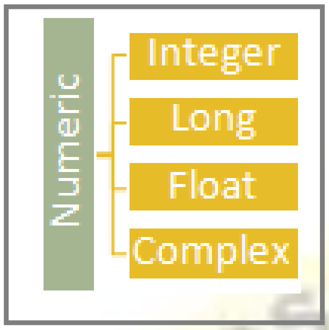


Types of datatypes :

1. **Numeric :**

a) **Integer (int)**: In python, the value of an integer can be ofunlimited length.(it only depends on the available memory)

b) **Long (long)**: Long integers of *unlimited length*. But existsonly in python 2.x.



c) **Float (float)**: Floating point numbers. Integers and floating point numbers are separated by a decimal point. The maximum no. of places after the decimal for a float is *15* in python.

d) **Complex numbers (complex)**: Complex numbers arerepresented as (x + yj) where x is the real part and y is the imaginary part.

Integer, long, float and complex all are immutable types.

Use type() function to check which class a variablebelongs to.

Example Program:

```
a = 15
print("a type ", type(a))
b = 15.5
print("a type ", type(b))
c = 15 + 6j
print("c type ", type(c))
```

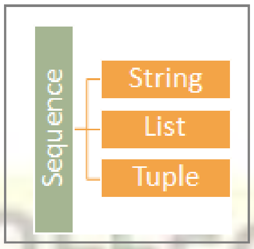
Output:

```
a type <class 'int'>
a type <class 'float'>
c type  <class 'complex'>
```

2. **Sequences :**

a) **Python String (str) :**

The string is actually a sequence of 8-bit characters (in Python 2.x) or a sequence of Unicode characters (in Python 3.x). Strings are immutable i.e. we cannot change any character of a String. To represent a String in python, single quote or double quote is used. For multiline strings, atriple quote is used. To print a particular character inside a String, use stringName[position]. Similarly, to print characters in a range, use stringName [firstPosition,lastPosition + 1].



Example program :

```
str1 = 'This is a String'
print (str1)
str2 = "This is also a String"
print (str2)
str3 = ''' This is a multiline
String '''
print (str3)
str4 = """ This is also a multiline
String """
print (str4)
print("Printing the first character of str1 "+str1[0])
print("Printing the first word of str1 "+str1[0:4])
```

Output:

```
This  is a String
This is also a String
This is a multiline
String
This is also a multiline
String
Printing the first character of str1 T
Printing the first word of str1 This
```

b) Python list :

Lists are ordered sequence of variables. The list is mutable, i.e. you can alter any item of a list. Also, list can contain items of different types. To declare a list bracket [ ]is used. If you want to print or alter a variable inside a list, you can use listname[variablePosition].

Example Program:

```
myList = [1,"one",1.0]
print (myList)
print ("first element of the list is: "+str(myList[0]))
myList[0] = "2"
print ("first element is changed:")
print ("now , first element is :"+myList[0])
```

Output:

```
[1, 'one', 1.0]
first element of the list is: 1
first element is changed:
now , first element is :2
```

c) **Python Tuple :**

Python tuple is same as a list, the only difference is that it is immutable. To represent tuple in python, we use parentheses ().

Example Program for a tuple :

```
myList = (1,"one",1.0)
print (myList)
print ("first element of the list is: "+str(myList[0]))
#myList[0] = "2"
print ("Any Element in tuple can not able to changed ")
print ("now , first element is: "+str(myList[0]))
```

Output:

```
(1, 'one', 1.0)
first element of the list is: 1
Any Element in tuple can not able to changed
now , first element is: 1
```

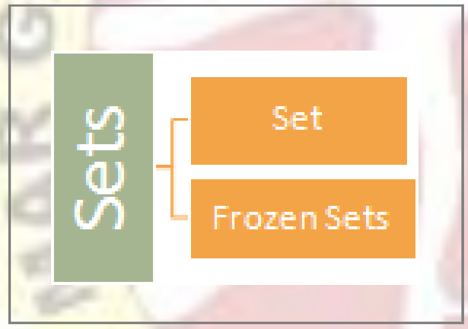
It will throw an error on myList[0] = “2” line aswe cannot change any values in the tuple.

3. **Python Sets :**

a) **Set :**

Set is an unordered collection of unique objects. Eachitem is separated by a comma inside braces { }.

We can also pass a list to the set function to create anew set.



Example Program:

```
mySet1 = set("python")
print (mySet1)
myList = (1,"one","two")
mySet2 = set(myList)
print (mySet2)
mySet3 = set([1,1,1,1,1,1])
print (mySet3)
```

Output :

```
{'h', 'o', 'y', 't', 'n', 'p'}
{1, 'two', 'one'}
{1}
```

mySet3-contains only one “1” as all values of a setshould be unique.

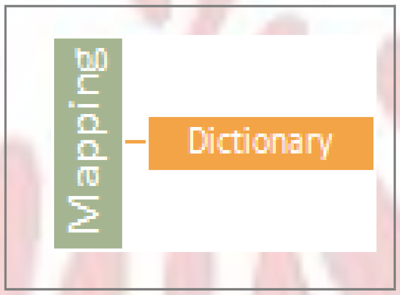
We can perform set operations like add, remove,intersection, union etc on sets.

b) Frozen Sets :

The frozen set is similar as sets but they areimmutable.

4. Python mapping :Python Dictionary

Python Dictionary is a collection of key-value pairs. Dictionary is an unordered collection. To get any value from the dictionary, we must know the key. Dictionaries are definedwithin braces {}. Each element or key-value pairs can be of anytype.



Example Program :

```
D = {"sun":'Sunday',"k2":2}
print("Element for Key 1 is:"+D["sun"])
print ("Element for key key2: "+str(D["k2"]))
```

Output:

```
Element for Key 1 is:Sunday
Element for key key2: 2
```

Python Typecasting

We need to convert the values from one data type to anotherdata type. The process of converting a value from one data type to another data type is called Typecasting or simply Casting. In Python,the typecasting is performed using built-in functions. As all the data types in Python are organized using classes, the type casting is performed using constructor functions. The following are the constructor functions used to perform typecasting.

S.No.	Function	Description
1	int( )	It is used to convert an integer literal, float literal, and string literal (String must represent a whole number) to an integer value.
2	float( )	It is used to convert an integer literal, float literal, and string literal (String must represent a whole number) to a float value.
3	str( )	It is used to convert a value of any data type including strings, integer literals and float literals to a string value.

Literals

Literal is a raw data given in a variable or constant. InPython, there are various types of literals they are as follows:

1. Numeric Literal
2. String literals
3. Boolean literals
4. Special literals
5. Literal Collections

1. Numeric Literal:

Numeric Literals are unchangeable. Numeric literals can belong to **3** different numerical types are : **Integer Literals, Float Literals , Complex.**

i) Integer Literals : It contain whole values in number. Integerliterals types are **Binary Literals(0b) , Decimal Literal, Octal Literal(0o), Hexadecimal Literal(0x)**. Integer Literals Examples are When print the variables, all the literals are converted into decimal values.

```
a = 0b1010 #Binary Literals print 10

b = 100 #Decimal Literal print 100

c = 0o310 #Octal Literal print 200

d = 0x12c #Hexadecimal Literal print 300
```

ii) Float Literals : It contain Whole value with fractional part.

For Examples
float_1 = 10.5 #print 10.5
float_2 = 1.5e2 #print 150.0 1.5e2 are floating-point literals. 1.5e2 is expressed with exponential and is equivalent to 1.5 *102

iii) Complex : To assigned a complex literal to create imaginaryand **real** part of complex number.

```
For  Example

 x = 3.14j

print(x, x.imag, x.real) #3.14j in variable x. Then we use imaginary literal (x.imag) and real literal (x.real) output  is 3.14j 3.14 0.0
```

2. String literals: A string literal is a sequence of characters surrounded by quotes. We can use both single, double or triple quotes for a string. And, a character literal is a single character surrounded by single or double quotes.

For Example

```
strings = "This is Python"

char = "C"

multiline_str = """This is a multiline string with more thanone line code."""

unicode = u"\u00dcnic\u00f6de"

raw_str = r"raw \n string"
```

```
print(strings)#This is Python

print(char)#C

print(multiline_str)# This is a multiline string with more than one line code

print(unicode)#Ünicöde

print(raw_str)#raw \n string
```

The value with triple-quote""""assigned in the multiline_str is multi-line string literal. The u"\u00dcnic\u00f6de" is a unicode literal which supports characters other than English and r"raw \n string" is a raw string literal.

3. Boolean literals: A Boolean literal can have any of the twovalues: True or False.
For Example

```
x = (1 == True)
y = (1 == False)
a = True + 4    #True=1(1+4=5)
b = False + 10 #False=0(0+10=10)
print("x is", x) #x is True
print("y is", y) #y is False
print("a:", a)    #a: 5
print("b:", b) #b: 10
```

4. **Special literals:** Python contains one special literal i.e.None. Weuse it to specify to that field that is not created .

5. **Literal Collections:** There are 4 different literal collections List Literals, Tuple Literals, Dict Literals, and Set Literals. They represent more complex data and helps to provide extendibility to Python programs.

   Let us use an example to see how these Literals function:-

```
colors = ["red", "green", "yellow"] #list
numbers = (101, 202, 304) #tuple
student = {'name':'John Doe', 'address':'California',
'email':'john@doe.com'} #dictionary
vowels = {'a', 'e', 'i' , 'o', 'u'} #set
print(colors)
print(numbers)
print(student)
print(vowels)
```

Variable

❖ A variable is a named location used to store data in the memory. when to create a variable reserve some space in memory. The value of Variable which can be changed later throughout programming.

To assigning different

❖ Data types into variable can store integers, decimals or characters in variable.

❖ The assignment operator (=) is used to assign a value to a variable. Example: a=1

Assigning Values to Variables:

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable.

For example

Output:

```
10
1000.0
Python
```

Multiple Assignment:

Python allows to assign a single value to several variablessimultaneously.

For example :**a = b = c = 5**

Here, an integer object is created with the value 5, and all three variables are assigned to the same memory location, can also assign multiple objects to multiple variables.

For example: **a,b,c = 5,10,"Python"**

Here, two integer objects with values 5 and 10 are assigned to variables a and b respectively, and one string object with the value"Python" is assigned to the variable c.

Constants

A constant is a type of variable **whose value cannot** be **changed**. It is helpful to think of constants as containers that hold information which cannot be changed later.

Assigning value to a constant in Python:

In Python, constants are usually declared and assigned on a module. Here, the module means a new file containing variables, functions etc which is imported to main file. Inside the module, constants are written in all capital letters and underscores separatingthe words.

Example 3: Declaring and assigning value to a constant

```
#Create a constant.py
PI = 3.14
GRAVITY = 9.8
```

#Create a main.py

```
import constant
print(constant.PI)
print(constant.GRAVITY)
```

Output:

```
3.14
9.8
```

Rules and naming convention for variables and constants:

1. Should not use special symbols like !, @, #, $, %, etc. in a variable name.
2. Do not start the variable name with a digit
3. Use capital letters where possible to declare a constant. Forexample PI, GRAVITY etc.
4. Constants are put into Python modules and meant not be changed.
5. Constant and variable names should have a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0to 9) or an underscore (_).

Identifier

- ❖ An identifier is a name given to entities like class, functions,variables etc. in Python, It helps to differentiate one entity from another.
- ❖ Identifier begins with a letter a to z or A to Z or an underscore (_) trailed by zero or more letters, underscores, and digits (0 to 9).

Rules naming conventions for Python identifiers:

1. Identifier begins with a letter a to z or A to Z or an underscore (_) trailed by zero or

more letters, underscores, and digits (0 to 9). Ex : Acc_no1=100.

2. Cannot use keywords as an identifier name. Ex int=10 – invalid (int is keyword)
3. An identifier cannot start with a digit. Ex: 5A - invalid identifier, however, digits can be added after the variable name Ex:A5 (valid).
4. Cannot use special symbols like !,@, #, $, % etc. in our identifier. Ex : A@=10 – invalid.
5. An identifier can be of any length.
6. Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
7. Starting an identifier with a single leading underscore indicates that the identifier is private.
8. Starting an identifier with two leading underscores indicates a strongly private identifier.
9. If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

Operators

- ❖ An Operators are special symbols in Python that carry out arithmetic or logical computation.
- ❖ The value that the operator operates on is called the operand. For Example : 4 + 5 = 9. Here, 4 and 5 are called operands and + is called operator.

Types of Operator

Python language supports the following types of operators.

1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Assignment Operators
4. Logical Operators
5. Bitwise Operators
6. Membership Operators
7. Identity Operators

1. **Arithmetic Operators:**

Assume variable a holds 10 and variable b holds 20, then –

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operandfrom left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand byright hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returnsremainder	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. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity) –	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

Example Program: CALCULATOR - USING ARITHMATICOPERATOR

```
a=int(input("Enter the A value:"))
op=input("Enter the operator:")
b=int(input("Enter the B value:"))
if op=="+":
print("Addition of A and B values are:",a+b)
elif op=="-":
print("Subtraction of A and B values are:",a-b)
elif op=="*":
print("Multiplication of A and B values are:",a*b)
elif op=="/":
print("division of A and B values are:",a/b)
```

```
elif op=="%":
print("Modulation of A and B values are:",a%b)else:
print("Invalid operator")
```

Output:

```
Enter the A value:5
Enter the operator:+
Enter the B value:4
Addition of A and B values are:9
Enter the A value:5
Enter the operator:&
Enter the B value:4
Invalid operator
```

2. **Comparison or Relational Operators:**

These operators compare the values on either sides of them and decide the relation among them. They are also called Relational operators.

Assume variable a = 10 and b = 20, then −

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 Program: BIGGEST OF TWO NUMBERS

```
a=int(input("Enter the A value:"))
b=int(input("Enter the B value:"))
if a>int b:
print("A is Biggest")
else:
print("B is Biggest")
```

/**Output:**

```
Enter the A value:50

Enter the B value:30

A is Biggest
```

3. **Assignment Operators:**

The Python Assignment Operators are handy to assign the values to the declared variables. Equals (=) operator is the most commonly used assignment operator in Python. For example: a=10, assign the value 10 into a.

The below table displays the list of available assignment operators in Python language.

PYTHON ASSIGNMENT OPERATORS	EXAMPLE	EXPLANATION
=	x = 25	Value 25 is assigned to x
+=	x += 25	This is same as x = x + 25
-=	x -= 25	Same as x = x – 25
*=	x *= 25	This is same as x = x * 25
/=	x /= 25	Same as x = x / 25
%=	x %= 25	This is identical to x = x % 25
//=	x //= 25	Same as x = x // 25
**=	x **= 25	This is same as x = x ** 25
&=	x &= 25	This is same as x = x & 25
|=	x |= 25	This is same as x = x | 25
^=	x ^= 25	Same as x = x ^ 25
<<=	x <<= 25	This is same as x = x << 25
>>=	x >>= 25	Same as x = x >> 25

Example Program: Arithmetic Operation using ShorthandAssignment Operator

```
a=int(50)
b=int(5)
c=a+b
print("Addition of A , B and assign to c=",c)
a+=b
print ("Addition of A , B and assign to A = ",a)
a-=b
print ("Subtraction of A , B and assign to A = ",a)
a*=b
print ("Multiplication of A , B and assign to A = ",a)
a/=b
print ("Division of A , B and assign to A = ",a)
a%=b
print ("Modulation of A , B and assign to A = ",a)
```

Output:

```
Addition of A , B and assign to c=55 #c=50+5=55
Addition of A , B and assign to A = 55 #a=50+5=55(now a=55)
Subtraction of A , B and assign to A = 50 #a=55-5=50(now a=50)
Multiplication of A , B and assign to A = 250 #a=50*5=250(now
a=250)
Division of A , B and assign to A = 50.0 #a=250/5=50(now a=50)
Modulation of A , B and assign to A = 0.0 #a=50%5=0(now a=0)
```

4. **Logical Operators:**

The logical operation is mainly done with conditional statements. These are mainly used with two logical operands if the value of logical operands is either True or False. The result of the logical operator is used for the final decision making. Three different types of logical operators are available in python:

- OR or Logical OR
- AND or Logical AND
- NOT or Logical NOT

Logical OR :

The output of logical OR will be False only if both operands are False. If either of them has a True value, it will result True. The syntax ‘or’ is used for logical OR operation. Following are the input and result of different OR operations :

Operand1	Operend2	Result
FALSE(0)	FALSE(0)	FALSE(0)
FALSE(0)	TRUE(1)	TRUE(1)
TRUE(1)	FALSE(0)	TRUE(1)
TRUE(1)	TRUE(1)	TRUE(1)

The final result is ‘False’ only if both operands are False.
Else, it is True always.

Logical AND :

The output of logical AND will be True only if both operands are True. If anyone of them is False, the result will be False. Syntax ‘and’ is used for logical AND operation. Input andresults for different AND operations are as follow :

Operand1	Operend2	Result
FALSE(0)	FALSE(0)	FALSE(0)
FALSE(0)	TRUE(1)	FALSE(0)
TRUE(1)	FALSE(0)	FALSE(0)
TRUE(1)	TRUE(1)	TRUE(1)

The final output is True only if both operands are True.
Else, it is False.

Logical NOT :

logical NOT is simple. It will just reverse the value. If the input is True, it will return False and if the input is False, it willreturn True.

Operand	Result
TRUE	FALSE
FALSE	TRUE

Example:Assume a= 10 and b = 20 then

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

Example Program: To Find given year is Leaf Year Or NotUsing logical operator

```
y=int(input("Enter the Year:"))
if((y%400==0)or(y%4==0)and (y%100!=0)):
print("Given year is Leaf year.")
else:
print("Given year is Not Leaf Year")
```

Output:

```
Enter the Year:1999
Given year is Not Leaf Year
Enter the Year:2020
Given year is Leaf year.
```

5. **Bitwise Operators:**

Bitwise operator works on bits and performs bit by bit operation. Assume if a = 60; and b = 13; Now in the binary format their values will be 0011 1100 and 0000 1101 respectively. Following table lists out the bitwise operators supported by Python language with an example each in those, we use the above two variables (a and b) as operands :

a = 0011 1100

b = 0000 110 1

----

a&b = 0000 1100

a|b = 0011 1101

a^b= 0011 0001

~a = 1100 0011

There are following Bitwise operators supported by Pythonlanguage.

Operator	Description	Example
& Binary AND	Operator copies a bit to the result if it exists in both operands	(a & b) (means 0000 1100)
| Binary OR	It copies a bit if it exists in either operand.	(a | b) = 61 (means 0011 1101)
^ BinaryXOR	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 Program:

```
a,b =60,13

print("a & b=",a&b)

print("a | b=",a|b)

print("a ^ b=",a^b)

print("~ a=",~a)
```

Output:

```
a & b=12
a  | b=61
a ^ b=49
~ a=-61
```

6. **Membership Operators** :

Python’s membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators are:

- in
- not in

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 if x 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.

Example Program:

```
a = int(input("Enter a value:"))
b = int(input("Enter b value:"))
li = [1, 2, 3, 4, 5 ]
if( ain li ):
print(" a is available in the given list")
else:
```

```
print(" a is not available in the given list")
if( bnotin li ):
print (" b is not available in the given list")
else:
print (" b is available in the given list")
```

Output:

```
Enter a value:4
Enter b value:10
 a is available in the given list
 b is not available in the given list
```

7. **Identity Operators** :

Identity operators compare the memory locations of twoobjects. There are two Identity operators as explained below –

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

Example Program :

```
a = int(input("Enter a value:"))
b = int(input("Enter b value:"))
if( ais b ):
print ("Line 1 - a and b have same identity")
```

```
else:
print ("Line 1 - a and b do not have same identity")
if( id(a) == id(b) ):
print ("Line 2 - a and b have same identity")
else:
print ("Line 2 - a and b do not have same identity")
```

Output:

```
Enter a value:20
Enter b value:20
Line 1 - a and b have same identity
Line 2 - a and b have same identity

Enter a value:20
Enter b value:10
Line 1 - a and b do not have same identity
Line 2 - a and b do not have same identity
```

Operators Precedence

The following table lists all operators from highestprecedence to lowest.

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>>< <	Right and left bitwise shift
&	Bitwise 'AND'td>
^ |	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

Expressions

- ❖ Expressions are representations of value.
- ❖ Python expressions only contain identifiers, literals, and operators.

- They are different from statement in the fact that statements do something while expressions are representation of value.
- For example any string is also an expressions since it represents the value of the string as well.
- Python has some advanced constructs through which you can represent values and hence these constructs are also called expressions.

Ex: c=a+b

Types of Python ExpressionsList comprehension:

The syntax for list comprehension is shown below:[compute(var) for var in iterable]

For example, the following code will get all the number within 10 and put them in a list.

```
print([x for x in range(10)])
```

Output:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Dictionary comprehension:

This is the same as list comprehension but will use curly braces: { k, v for k in iterable }

For example, the following code will get all the numbers within 5 as the keys and will keep the corresponding squares of thosenumbers as the values.

```
print({x:x**2for x in range(5)})

{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

Generator expression:

The syntax for generator expression is shown below: (compute(var) for var in iterable)

For example, the following code will initialize a generatorobject that returns the values within 10 when the object is called.

Example:

```
print((x for x in range(10)))

print(list(x for x in range(10)))
```

Output:

```
<generator object <genexpr> at 0x0020B4F8>
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Conditional Expressions: true_value if Condition else false_value

Example:

```
x=1
if x==1:
print("true")
else:
print("false")
```

Output:

```
true
```

LIST

List, Structure of list and Python list

List: A sequence of elements or items written or printed togetherin a meaningful single name.

For Example: list = ['one','two','three','four','five,'six']

Defining a Python list:

- ❖ In Python, a list is an ordered collection of objects.
- ❖ A list can contain different types of objects, even other lists.
- ❖ Lists are very similar to arrays.
- ❖ A list is enclosed by brackets [ ] with the first element atindex 0, where each element is separated by a comma.
- ❖ It implements the sequence protocol, and also allows you toadd and remove objects from the sequence.
- ❖ For example, you can define a list of integers as follows: list = [1,3,2,7,9,4]
- ❖ In Python, the size of the list can grow or shrink whenneeded.

Creating List:

Creating a list is as simple as putting different comma-separated values in square brackets.

```
a_list = [1,2,3,4]

b_list = ['a','b','c','d']

c_list = ['one','two','three','four','five,'six']

d_list = [1,2,'three','four']
```

Accessing List Values:

The syntax for accessing the elements of a list is the same as the syntax for accessing the characters of a string. The expression inside the brackets specifies the index. Python indexes starts its listsat 0 rather than 1.

Example 1:

```
a_list = [1,2,3,4]

num1 = a_list[0]

num2 = a_list[3]

print(num1)

print(num2)
```

Output:

```
1

4
```

Example 2:

```
d_list = [1,2,'three','four']

num = d_list[1]

str = d_list[2]

print(num)

print(str)
```

Output:

```
2

three
```

Basic List Operations:

Lists respond to the + and * operators much like strings; theymean concatenation and repetition here too, except that the result is anew list, not a string.

In fact, lists respond to all of the general sequence operationswe used on strings in the prior chapter.

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

List length: The function len( ) returns the length of a list, whichis equal to the number of its elements.

Example:

```
lt = [1,2,'three','four']

print(len(lt))
```

Output:

```
4
```

"+" operator : It is used to concatenates lists.

Example:

```
lt = [1,2,'three','four']

lt1=[5,'six',7]

lt2=lt+lt1

print ("Lists are in lt:",lt)

print("Lists are in lt1:",lt1)

print("Concatenation of two lists lt,lt1 are contain: ",lt2)
```

Output:

```
Lists are in lt: [1, 2, 'three', 'four']

Lists are in lt1: [5, 'six', 7]

Concatenation of two lists lt,lt1 are contain: [1, 2,  'three',
'four', 5, 'six', 7]
```

using the * operator repeats a list a given number of times.

Example:

```
lt = [1,2,'three']

print(lt*4)
```

Output:

```
[1, 2, 'three', 1, 2, 'three', 1, 2, 'three', 1, 2, 'three']
```

Indexing, Slicing, and Matrixes:

Lists are sequences, indexing and slicing work the same wayfor lists as they do for strings.

Assuming following input L = ['spam', 'Spam', 'SPAM!']

Python Expression	**Results**	**Description**
L[2]	SPAM!	Offsets start at zero
L[-2]	Spam	Negative: count from the right
L[1:]	['Spam', 'SPAM!']	Slicing fetches sections

Slice Elements: Python slice extracts elements, based on a start andstop.

Syntax:objectname[start,stop]

start- start the position of element slice extracts in the list.

Stop- End of the position of element slice extracts in the list.

Example 1 :

```
lt = [1,2,'three',2,3,'five',2,2]

print("Element in list:",lt)

print("slice extracts elements are : ",lt[2:6])
```

Output:

```
Element in list: [1, 2, 'three', 2, 3, 'five', 2, 2]

slice extracts elements are : ['three', 2, 3, 'five']
```

lt[2:6] - The 2 means to start at third element in the list (note that the slicing index starts at 0). The 6 means to end at the sixth element in the list, but not include it.

The colon in the middle is how Python's lists recognize that we want to use slicing to get objects inthe list.

Example 2:

```
lt = [1,2,'three',2,3,'five',2,2]
print("Element in list:",lt)
print("slice first three elements  : ",lt[:3])
print("slice from 4th element, Python starts its lists at 0
rather than 1 : ",lt[3:])
```

Output:

```
Element in list: [1, 2, 'three', 2, 3, 'five', 2, 2]
slice first three elements : [1, 2, 'three']
slice from 4th element, Python  starts its  lists at 0  rather  than
1 : [2, 3, 'five', 2, 2]
```

Built-in List Functions & Methods:

Python includes the following list functions are:

Sr.No.	Function with Description
1	cmp(list1, list2) Compares elements of both lists.
2	len(list) Gives the total length of the list.
3	max(list) Returns item from the list with max value.
4	min(list) Returns item from the list with min value.
5	list(seq) Converts a tuple into list.

cmp() function: Python list method **cmp()** compares elements oftwo lists.

Syntax:cmp(list1, list2) Parameters:

- ❖ List1− This is the first list to be compared.
- ❖ List2− This is the second list to be compared.

Return Value: If elements are of the same type, perform the compare and return the result. If elements are different types, checkto see if they are numbers.

- ❖ If numbers, perform numeric coercion if necessary and compare.
- ❖ If either element is a number, then the other element is "larger"(numbers are "smallest").
- ❖ Otherwise, types are sorted alphabetically by name.

If we reached the end of one of the lists, the longer list is "larger." If we exhaust both lists and share the same data, the result is a tie, meaning that 0 is returned.

Example:

```
lt = [1,2,3,4,5]

print("the total length of the list 1 :",len(lt))

print("Returns item from the list1 with max value:",max(lt))

print("Returns item from the list1 with minimum value:",min(lt))
```

Output:

```
the total length of the list 1 : 5

Returns item from the list1 with max value: 5

Returns item from the list1 with minimum value: 1
```

Python includes following other valuable list methods are:

Sr.No.	Methods with Description
1	list.append(obj) Appends object obj to list
2	list.count(obj) Returns count of how many times obj occurs in list
3	list.extend(seq) Appends the contents of seq to list
4	list.index(obj) Returns the lowest index in list that obj appears
5	list.insert(index, obj) Inserts object obj into list at offsetindex
6	list.pop(obj=list[-1]) Removes and returns last object orobj from list
7	list.remove(obj) Removes object obj from list
8	list.reverse() Reverses objects of list in place
9	list.sort([func]) Sorts objects of list, use compare func ifgiven

Inserting and Removing Elements:

append() - Appends adds its argument as a single element to the endof a list. The length of the list itself will increase by one.

Example:

```
lt = [1,2,'three','four']
print ("Lists Are in lt:",lt)
print("List Length=",len(lt))
lt.append(5)
print("After insert new element:",lt)
print("New List Length=",len(lt))
```

Output:

```
Lists Are in lt: [1, 2, 'three', 'four']
List Length= 4
After insert new element: [1, 2, 'three', 'four', 5]
New List Length= 5
```

Appending a list inside a list: We can also appending insideof another list.

Example:

```
lt = [1,2,'three','four']
lt1=[5,'six',7]
print ("Lists Are in lt:",lt)
print("List Length=",len(lt))
lt.append(lt1)
print("After insert list inside of another list:",lt)
print("New List Length=",len(lt))
```

Output:

```
Lists Are in lt: [1, 2, 'three', 'four']
List Length= 4
After insert new list inside of another list: [1, 2, 'three', 'four', [5, 'six', 7]]
New List Length= 5
```

Inserting elements in List given position: We can also insert anelement in given position on the list.

Syntax:objectname(position,element)

Example:

```
lt = [1,2,'four']
print("Before insert an element in list:",lt)
lt.insert(2,3)
print("After inert an element given position 2 in the list: ",lt)
```

Output:

```
Before insert an element in list: [1, 2, 'four']
After inert an element given position 2 in the list: [1, 2, 3,
'four']
```

Insert an element at end of the list: Negative(-) is used to Inserts an element into the last position of the list. **Negative** indicesstart from the end of the list.

Example:

```
lt = [1,2,'three','five']
print("Before insert an element in list:",lt)
lt.insert(-1,4)
print("After inert an element given position -1 in the list: ",lt)
```

Output:

```
Before insert an element in list: [1, 2, 'three', 'five']
After inert an element given position -1 in the list: [1,  2,
'three', 4, 'five']
```

Remove elements from List: This method is used to removethe element from the list.

Syntax:objectname.remove(value).

Example:

```
lt = [1,2,'three','five']
print("Before remove an element in list:",lt)
lt.remove('three')
print("After remove an element in the list: ",lt)
```

Output:

```
Before remove an element in list: [1, 2, 'three', 'five']
After remove an element in the list: [1, 2, 'five']
```

Clear or Emptying List: This method is used to remove allitems from the list.

Syntax:list.clear()

Example:

```
lt = [1,2,'three','four']
print("List Length=",len(lt))
print ("Lists Are in lt:",lt)
lt.clear()
print("After clear Lists, That contain empty List:",lt)
```

Output:

```
List Length= 4
Lists Are in lt: [1, 2, 'three', 'four']
After clear Lists, That contain empty List: []
```

List Count: Count()- This function is used to count the value howmany time present in the list.

Syntax: list.count(x)- return the number of times x appears in the list.

Example:

```
lt = [1,2,'three',2,3,'five',2,2]
print("Element in list:",lt)
print("Count the how many Number of times 2 is present in the list : ",lt.count(2))
```

Output:

```
Element in list: [1, 2, 'three', 2, 3, 'five', 2, 2]
Count the how many Number of times 2  is present in the list
: 4
```

List Reverse: The reverse() method in list reverse theelements of the list in place.

Syntax:list.reverse()

Parameters: NA

Return Value: This method does not return any value butreverse the given object from the list.

Example:

```
lt = [1,2,3,4,5]
lt1=['one','two','three']
print("Elements in first lists are :",lt)
lt.reverse()
print("Elements in reversed first lists are :",lt)
print("Element in Second lists are :",lt1)
lt1.reverse()
print("Element in  reversed second lists are  : ",lt1)
```

Output:

```
Elements in first lists are : [1, 2, 3, 4, 5]
Elements in reversed first lists are : [5, 4, 3, 2, 1]
Element in Second lists are : ['one', 'two', 'three']
Element in reversed second lists are  :  ['three', 'two', 'one']
```

List index():

The index() method returned the index of the first matching item.

```
Example:
lt = ['h','a','i',4,5]

print("Index value :",lt.index('a'))
```

Output:

```
Index value : 1
```

Exist “in” List and “not in” List: We can test if an itemexists in a list or not, using the keyword "in" and “not in”

Example:

```
lt = ['h','a','i',4,5]

print("Exist value in list using in:",'a'inlt)

print("Exist value in list using not in:",'a'notinlt)
```

Output:

```
Exist value in list using in: True

Exist value in list using not in: False
```

List sort and Reverse Sorting: List sort() method that performs anin-place sorting

Example:

```
lt = [5,2,7]

lt1=['c','e','b','a']

lt.sort()

lt1.sort(reverse=True)

print("sorted value in list1:",lt)

print("reverse sorted value in the list:",lt1)
```

Output:

```
sorted value in list1: [2, 5, 7]

reverse sorted value in the list: ['e', 'c', 'b', 'a']
```

Remove duplicates from a Python List: The commonapproach to get a **unique collection** of items is to use a dictionary. A **Python dictionary** is a mapping of **unique keys** to values. So, converting **Python list** to dictionary will automatically remove any duplicates because dictionaries cannot have **duplicate keys**.

Example:

```
li = [1,2,3,1,4,2,5,3]

print("Original   list:",li)

li = list(dict.fromkeys(li))

print("After removed duplicate values in the list",(li))
```

Output:

```
Original list: [1, 2, 3, 1, 4, 2, 5, 3]

After removed duplicate values in the list [1, 2, 3, 4, 5]
```

PYTHON OBJECTS

Objects and their use:

In procedural programming, functions are the primary building blocks of program design. In object-oriented programming, objects are the fundamental building blocks in which functions are acomponent.

Object as something that has a set of attributes and its relatedset of behaviours.

Python Classes and Objects

Python is an object-oriented programming language. Unlike procedure-oriented programming, where the main emphasis on functions, object-oriented programming stress on objects.

Object is simply a collection of data and methods that act onthose data.

Class is a blueprint for the object. class as a prototype of an object ex: house. It contains all the details about the doors, windows, floors etc. Based on these descriptions we build the house.House is the object.

An object is also called an instance of a class and the process of creating this object is called **instantiation**.

Class creates a user-defined data structure, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class.

Some points on Python class:

- Classes are created by keyword class.
- Attributes are the variables(data) that belong to class.
- Attributes are always public and can be accessed using dot (.)operator.

Eg.: Myclass.MyattributeSyntax

Class classname:

# statement 1

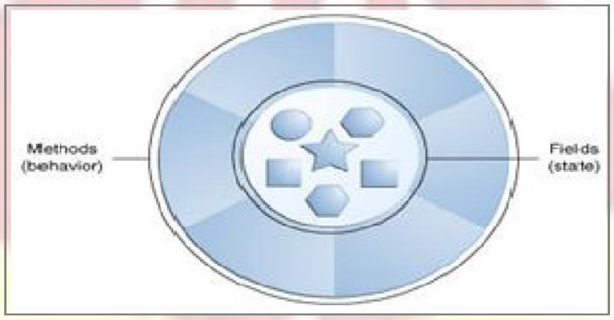
.

.

# statement N

Software Objects:

An Object is an instance of a Class. A class is like a blueprintwhile an instance is a copy of the class with *actual values*. You can



have many dogs to create many different instances, but without the class as a guide, you would be lost, not knowing what information isrequired.

An object consists of :

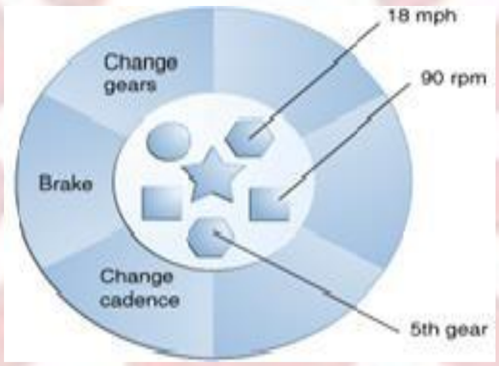
- **State :** It is represented by attributes of an object. It alsoreflects the properties of an object.
- **Behavior :** It is represented by methods of an object. It alsoreflects the response of an object with other objects.
- **Identity :** It gives a unique name to an object and enablesone object to interact with other objects.

Software objects are conceptually similar to real-world objects: they too consist of state and related behavior.

An object stores its state in *fields* (variables in some programming languages) and exposes its behavior through *methods*(functions in some programming languages).

Methods operate on an object's internal state and serve as theprimary mechanism for object-to-object communication. Hiding internal state and requiring all interaction to be performed through an object's methods is known as *data encapsulation* — a fundamental principle of object-oriented programming.

❖ Consider a bicycle, for example:



A bicycle modeled as a software object.

UNIT II

MODULE

The term module refers to the design and/or implementation of specific functionality to be incorporated into a program.

Modular programming is a software design technique to split your code into separate parts. These parts are called modules. The focus for this separation should be to have modules with no or just few dependencies upon other modules. Modules are collection of functions (or entites).

Modules in python

A module is a python object that allows logically organize the python code. Simply, Modules refers to a file containing python statements and definition. A file containing python code for example sample.py , here sample is a module name that written in python .

Let us create a module. Type the coding and save it as sample.py

# python module sample

```
def add(a, b):
   """This program adds two numbers and return the result"""


   result = a + b
   return result
```

Here, we have defined a function add() inside a module named sample. The function takes in two numbers and returns theirsum.

Import modules in Python

We can import the definitions inside a module to another module or the interactive interpreter in Python.

We use the previously `import` keyword to do this. To import our defined modulePython prompt. `example` , we type the following in the

import sample

This does not import the names of the functions definedin example directly in the current symbol table. It only imports the module nameexamplethere.Using the module name we can access the function using the dot . operator.example.add (4,6)

Output 10

Python import statement (pre defined modules)

We can import a module using the import statement andaccess the definitions inside it using the dot operator as described above. Here is an example.

```
# import statement example
# to import standard module math
import math
print("The value of pi is", math.pi)
When you run the program, the output will be:
```

```
The value of pi is 3.141592653589793
```

Rename the module:

We can import a module by renaming it as follows:

```
# import module by renaming it
import math as mt
print("The value of pi is", mt.pi)
```

We have renamed the math module as mt.Note that the module math is not recognized in our scope. Hence, math.pi is invalid, and mt.pi is the correct implementation.

Python from...import statement

We can import specific names from a module withoutimporting the module as a whole. Here is an example.

```
# import only pi from math module
from math import pi
print("The value of pi is", pi)
```

Here, we imported only the pi

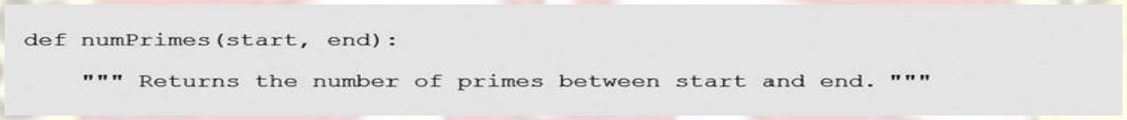
Import all names math

attribute from the module.

We can import all names(definitions) from a module usingthe following construct:

```
# import all names from the standard module math
from math import *
print("The value of pi is", pi)
```

Every module needs to provide a specification of how it is to be used. Any program code making use of a particular module is referred to as a client of the module. A module specification shouldbe sufficiently clear and complete.



```
def numPrimes(start, end):
    """ Returns the number of primes between start and end. """
```

The function's specification is provided by the line immediately following the function header, called a docstringin Python. A **docstring** is a string literal denoted by triple quotes.

LOCATING MODULES

When you import a module, the python interpreter searches forthe module in the following sequences:

- ❖ The Current directory
- ❖ If the module isn't found, Python then searches eachdirectory in the shell variable PYTHONPATH
- ❖ If all else fails, python checks the default path.

The PYTHONPATH is an environment variable, consisting of a list of directories.

Syntax of PYTHONPATH is same as that of shell variable PATH.

- ❖ For Windows, set PYTHONPATH=c:\python20\lib
- ❖ For Unix, set PYTHONPATH=/usr/local/lib/python

PYTHON MODULES:

Python modules provide all the benefits of modular software design. Usually, Python module is a file containing Python definitions and statements. When a Python file is directly executed, it is considered the main moduleof a program. Main modules are given the special name_main_. Main modules provide the basis for a complete Python program. As with the main module, imported modules may contain a set of statements. The statements of importedmodules are executed only once, the first time that the module is imported. The purpose of these statements is to perform any initialization needed for the members of the imported module. The Python Standard Library contains a set of predefined Standard (built-in) modules.

Create a Python module by entering the following in a fi le name simple.py. Then execute the instructions in the Python shell asshown and observe the results.

```
# module simple                                    import simple
print('module simple loaded')                        ???

def func1():                                       simple.func1()
print('func1 called')                                ???

def func2():                                       simple.func2()
print('func2 called')                                ???
```

Modules and Namespaces

In Python, each module has its own namespace. Namespace is a collection of names and containing all built-in names is created.A namespace is basically a system to make sure that all the names in a program are unique and can be used without any conflict. It enables programs to avoid potential name clashesby associating each identifier with the namespace from which it originates.

```
# var1 is in the global namespacevar1 = 5
def some_func():

# var2 is in the local namespacevar2 = 6
def some_inner_func():

# var3 is in the nested local# namespace
var3 = 7
```

FUNCTIONS

What is a Function?

A function is a block of code which is used to perform someaction, and it is also called as reusable code.

A function provides higher modularity and code re-usability.

What is a Python Main Function?

As Python is an interpreted language, it follows a top-down approach. Python is interpreted there is no static entry point to the program and the source code is executed sequentially and it doesn't call any methods unless you manually call it. The most important factor in any programming language is the 'modules'. The module is a program that can be included or imported to the other programs sothat it can be reused in the future without writing the same module again. However, there is a special function in Python that helps us toinvoke the functions automatically by operating the system during run-time or when the program is executed, and this is what we call asthe main function. Even though it is **not mandatory to use mainfunction** in Python, it is a good practice to use this function asit improves the logical structure of the code.

What is a function in Python?

- In Python, a function is a group of related statements thatperforms a specific task.
- Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.
- Functions are very useful features of Python to perform your **task with less coding**. It contains codes which you can add to perform certain tasks. You can call it much time to perform the same operation.
- Furthermore, it avoids repetition and makes the code reusable.
- A function is a block of organized, reusable code that is used to perform a single, related action.
- Functions provide better modularity for your application and a high degree of code reusing.

As you already know.

- Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions arecalled user-defined functions.

Types of Functions:

Basically, we can divide functions into the following two types:

- **Built-in functions**
- **User defined Function**

Built-in functions- Functions that are built into Python.Python has a set of built-in functions are:

Function	Description
abs()	Returns the absolute value of a number
eval()	Evaluates and executes an expression
exec()	Executes the specified code (or object)
filter()	Use a filter function to exclude items in aniterable object
float()	Returns a floating point number
id()	Returns the id of an object
input()	Allowing user input
int()	Returns an integer number
isinstance()	Returns True if a specified object is an instance of a specified object
issubclass()	Returns True if a specified class is a subclassof a specified object
iter()	Returns an iterator object
len()	Returns the length of an object
list()	Returns a list
map()	Returns the specified iterator with the specified function applied to each item
max()	Returns the largest item in an iterable
min()	Returns the smallest item in an iterable
next()	Returns the next item in an iterable
object()	Returns a new object
oct()	Converts a number into an octal
open()	Opens a file and returns a file object
pow()	Returns the value of x to the power of y
print()	Prints to the standard output device
round()	Rounds a numbers
set()	Returns a new set object

setattr()	Sets an attribute (property/method) of an
	object
slice()	Returns a slice object
sorted()	Returns a sorted list
str()	Returns a string object
sum()	Sums the items of an iterator
tuple()	Returns a tuple
type()	Returns the type of an object

User-defined functions:

- Functions defined by the users themselves.
- Functions that we define ourselves to do certain specific task are referred as user-defined functions.
- Functions that readily come with Python are called built-in functions. If we use functions written by others in the form oflibrary, it can be termed as library functions.
- All the other functions that we write on our own fall under user-defined functions. So, our user-defined function could be a library function to someone else.

Advantages of user-defined functions:

- By using a function on your programming, you don’t have tocreate the same code again and again.
- User-defined functions help to decompose a large program into small segments which makes program easy to understand, maintain and debug.
- If repeated code occurs in a program. Function can be used toinclude those codes and execute when needed by calling thatfunction.
- Programmers working on large project can divide the workload by making different functions.

Defining a Function:

Define functions to provide the required functionality. Hereare simple rules to define a function in Python.

- Function blocks begin with the keyword def followed by the function name and parentheses (( )).
- Any input parameters or arguments should be placed within these parentheses. You

can also define parameters inside these parentheses.

- The first statement of a function can be an optional statement - the documentation string of the function or docstring.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller.
- A return statement with no arguments is the same as return None.

Syntax:

```
def functionname( parameters ):
    "function_docstring"
function_suite
return [expression]
```

By default, parameters have a positional behavior and you need to inform them in the same order that they were defined. The following function takes a string as input parameter and prints it onstandard screen.

Example:

```
def fun( ):
print (“Wlcome Function”)
```

Here function is now fully defined, but if we run the program at this point, nothing will happen since we didn't call the function. So, outside of the defined function block, let's call the function withfun()

Calling a Function:

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures theblocks of code.

Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt. Following is the example to call fun() function

Example:

```
def fun( ): #Function definition
print (“Welcome Function”)
fun()    #function call
```

Output:

```
Welcome Function
```

fun()- is function call, It call fun function and print the string Welcome Function.

Python Function Parameters calling values:

A parameter is the variable which is part of the method’s signature (method declaration). Parameters are specified within the pair of parentheses in the function definition, separated by commas.When we call the function, we supply the values in the same way.

Returning a Value:

Not only can you pass a parameter value into a function, a function can also produce a value.

The return statement is used to return from a function.

The statement return [expression] exits a function, optionally passing back an expression to the caller.

A return statement with no arguments is the same as return None.

Syntax:

```
def fun():
    statements
    .
    .
    return [expression]
```

Example : Factorial Program

```
def fact(n): #function definition
if n==1:
returnn #function return
elif n>1:
return n*fact(n-1) # returning a value
```

```
n=int(input("Enter the number :"))
print("Factorial of ",n ," = ",fact(n))    # function calling
value or passing parameter n
```

Output:

```
Enter the number : 5 Factorial of     5 = 120
```

Function Calling Non-value- Returning Function:

Returning Multiple Values

In Python, we can return multiple values from a function. Following are different ways.

❖ **Using Object (Return by object):** create a class to holdmultiple values and return an object of the class.

Example:

```
class Test:
def__init__(self):
self.str = "Welcome python"
self.x = 20
# This function returns an object of Test
def fun():
returnTest()
t = fun()
print(t.str)
print(t.x)
```

Output:

```
Welcome python
20
```

Using a list: A list is like an array of items created using square brackets. They are different from arrays as they can contain items of different types. Lists are different from tuples as they are mutable.

Example:

```
def fun():
    str1 = "welcome python"
    x = 20
return [str1, x];
list1 = fun()
print(list1)
```

Output: ['welcome python', 20]

Using a Dictionary: A Dictionary is similar to hash or mapin other languages.

Example:

```
def fun():
    d = dict();
    d['str'] = "welcome to python"
    d['x'] = 20
return d
d = fu006E()
print(d)
```

Output: {'str': 'welcome to python', 'x': 20}

Parameter passingFunction Arguments:

We can call a function by using the following types of formalarguments –

- Required arguments
- Keyword arguments
- Default arguments

❖ Variable-length arguments

Required arguments:

Required arguments are the arguments passed to a function in correct positional order. Here, the number of arguments in thefunction call should match exactly with the function definition.

Example:

```
def fun(st):
print(st)
fun()
```

To call the function fun(), you definitely need to pass oneargument, otherwise it gives a syntax error as like below

```
Traceback (most recent call last):
File "D:\workspace\HelloWorld\gayathri\function.py", line 3,
in <module>
fun()
TypeError: fun() missing 1 required positional argument: 'st'
```

Passing number of correct argument into function:

```
def fun(st):
print(st)
st="Welcome Function"
fun(st) # passing one string argument
```

Output:

```
Welcome Function
```

Keyword arguments:

Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.

This allows you to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters. You can also make keyword calls to the fun ( ) functionin the following ways :

Example:

```
def fun( str ): # str is keyword but used argument

nameprint (str)

return;

fun( str = "My string")
```

Output:

```
My string
```

Default arguments:

A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument.The main advantage of default argument is that we can give values to only those parameters to which we want to, provided that the other parameters have default argument values. It assign the value right to left.

Example : It prints default age if it is not passed

```
def printinfo( name, age = 35 ):

print("Name: ", name)

print("Age ", age)

printinfo( age=50, name="python" )

printinfo( name="phthon" )# age is not passed but assigned
default argument age is 35
```

Output:

```
Name: python

Age   50

Name:  python

Age 35
```

Variable-length arguments:

You may need to process a function for more arguments thanyou specified while defining the function. These arguments are called variable-length arguments and are not named in the function definition, unlike required and default arguments.

Syntax:

```
def functionname([formal_args,] *var_args_tuple ):
   "function_docstring"
function_suite
   return [expression]
```

Example:

```
def fun( arg1, *vartuple ):
print ("Output is: ")
print(arg1) #printed first argument
for var invartuple:
print (var)# it is printing remaining argument
fun( 10,20 )
fun( 70, 60, 50 )
```

Output:

```
Output is:
10
20
Output is:
70
60
50
```

The Anonymous Functions:

These functions are called anonymous because they are not declared in the standard manner by using the def keyword. You canuse the lambda keyword to create small anonymous functions.

- Lambda forms can take any number of arguments but returnjust one value in the form of an expression. They cannot contain commands or multiple expressions.
- An anonymous function cannot be a direct call to print because lambda requires an expression
- Lambda functions have their own local namespace and cannot access variables other than those in their parameter list and those in the global namespace.
- Although it appears that lambda's are a one-line version of a function, they are not equivalent to inline statements in C orC++, whose purpose is by passing function stack allocation during invocation for performance reasons.

The syntax of lambda functions contains only a singlestatement, which is as follows :

Syntax:

lambda [arg1 [,arg2,..............................argn]]:expression

Following is the example to show how lambda form offunction works :

```
sum1 = lambda arg1, arg2: arg1 + arg2;

print ("Value of total : ", sum1( 10, 20 ))

print ("Value of total : ", sum1( 20, 20 ))
```

Output:

```
Value of total :  30

Value of total :  40
```

Scope of Variables

All variables in a program may not be accessible at alllocations in that program. This depends on where you have declared a variable. The scope of a variable determines the portion of the program where you can access a particular identifier. There are twobasic scopes of variables in Python : They are

- Global variables
- Local variables

Global vs. Local variables:

Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.

This means that local variables can be accessed only inside the function in which they are declared, whereas global variables can be accessed throughout the program body by all functions.

When you call a function, the variables declared inside it arebrought into scope.

Example :

```
total = 0; # This is global variable.
# Function definition is here
def sum1( arg1, arg2 ):
    # Add both the parameters and return them."
    total = arg1 + arg2 # Here total is local variable
45+55=100.
print("Inside the function local total : ", total)
return total
sum1( 45, 55 )
print ("Outside the function global total : ", total)
```

Output:

```
Inside the function local total : 100
Outside the function global total : 0 #It print 0 because now
reassigned global variable
```

Inner Functions:

A function contain inside of another function called inner function.The main advantage of inner functions is that it protect them from anything happening outside of the function, meaning thatthey are hidden from the global scope.

Example:

```
def calc(x,y):#outer function
def findSum(x,y):#innner function
returnx+y
    sum1 = findSum(x,y)
print("Sum of ", x, " + ", y, " is ", sum1)
calc(10,20)
```

Output:

```
Sum of 10 + 20 is 30
```

Assign functions to variables:

When you assign a function to a variable you don't use the ()but simply the name of the function.

Example:

```
def findSum(x,y):
returnx+y
f1 = findSum(10,20)#function call and assign to f1 variable
print("f1=",f1)
f2 = findSum(40,20)#function call and assign to f2 variable
print("f2=",f2)
```

Output:

```
f1= 30
f2= 60
```

Python recursive functionsDefinition:

- When a function call itself is knows as recursion.
- Recursion works like loop but sometimes it makes more sense to use recursion than loop. You can convert any loop torecursion.
- You'd imagine such a process would repeat indefinitely if not stopped by some

condition.

- This condition is known as base condition.
- A base condition is must in every recursive programs otherwise it will continue to execute forever like an infinite loop.

Example:

Factorial is denoted by number followed by (!) sign i.e4!,2!,1!

4! = 4 * 3 * 2 * 1

2! = 2 * 1

1! = 1

Overview of how recursive function works:

1. Recursive function is called by some external code.
2. If the base condition is met then the program do something meaningful and exits.
3. Otherwise, function does some required processing and then call itself to continue recursion. Here is an example of recursive function used to calculate factorial.

Why use recursion in programming?

We use recursion to break a big problem in small problems and those small problems into further smaller problems and so on. At the end the solutions of all the smaller subproblems are collectively helps in finding the solution of the big main problem.

Example: ***Factorial using recursive function***

```
def fact(n):
if n==1:
return n
elif n>1:
return n*fact(n-1)
n=int(input("Enter the number :"))
print("Factorial of  ",n ," = ",fact(n))
```

Output:

```
Enter the number  :5

Factorial of  5 = 120
```

Advantages of recursion:

Recursion makes our program:

1. Easier to write.
2. Readable – Code is easier to read and understand.
3. Reduce the lines of code – It takes less lines of code to solvea problem using recursion.

Disadvantages of recursion:

1. Not all problems can be solved using recursion.
2. If you don’t define the base case then the code would run indefinitely.
3. Debugging is difficult in recursive functions as the function is calling itself in a loop and it is hard to understand which call is causing the issue.
4. Memory overhead – Call to the recursive function is not memory efficient.

UNIT III

Reading and Writing Text file What is a file?

- File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory(e.g. hard disk).
- Since, random access memory (RAM) is volatile which losesits data when computer is turned off, we use files for future use of the data.
- When we want to read from or write to a file we need to openit first. When we are done, it needs to be closed, so that resources that are tied with the file are freed.

Types of file in python:

- **Text file**
    - **Binary FileText file:**

Text files are structured as a sequence of lines, where each line includes a sequence of characters. Each line is terminated with aspecial character, called the EOL or **End of Line** character. There are several types, but the most common is the comma {,} or newlinecharacter. It ends the current line and tells the interpreter a new one has begun. A backslash character can also be used, and it tells the

interpreter that the next character – following the slash – should be treated as a new line. This character is useful when you don't want tostart a new line in the text itself but in the code.

Examples: Python source code, HTML file, text file, markdown fileetc.

Binary File

A binary file is any type of file that is not a text file. It is important to note that inside the disk both types of files are stored as a sequence of 1's and 0's. Because of their nature, binary files can only be processed by an application that know or understand the file's structure.

Example Binary files: executable files, images, audio etc.

File Operation :

1. Open a file
2. Read or write (perform operation)
3. Close the file

1. Opening the file - open() function:

The open() built-in function is used to open the file.

Syntax :

open(filename, mode) -> file object

On success, open() returns a file object. On failure, itraises IOError or it's subclass.

Filename - Absolute or relative path of the file to be opened.

Mode - (optional) mode is a string which refers to the processingmode (i.e read, write, append etc;) and file type.

The following are the possible values of mode.

Mode	Description
r	Open the file for reading (default).
w	Open the file for writing.
rb	Reading in binary format.
wb	Writing in binary format.
r+	Open the file for both reading and writing.
w+	Open the file for both reading and writing. Overwrites thefile if the file exits otherwise creates a new one.
a	Open the file in append mode i.e add new data to the end ofthe file.
a+	Open a file for both appending and reading.
ab	Open a file for appending in binary format.
X	Open the file for writing, only if it doesn't already exist.

We can also append t or b to the mode string to indicate the type of the file we will be working with. The t is used for text file and b for binary files. If neither specified, t is assumed by default.

The mode is optional, if not specified then the file will beopened as a text file for reading only.

This means that the following three calls to open() are equivalent:

```
f = open("test.txt")      # equivalent to 'r' or 'rt'
f = open("test.txt",'w') # write in text mode
f = open("img.bmp",'r+b') # read and write in binary mode
```

Note that before you can read a file, it must already exist, otherwise open() will raise FileNotFoundError exception.However, if you open a file for writing (using mode such as w, a,or r+), Python will automatically create the file for you. If the file already exists then its content will be deleted. If you want to prevent that open the file in x mode.

2. **Closing a file using close()**

When we are done with operations to the file, we need toproperly close the file.

Closing a file will free up the resources that were tied with the file and is done using Python close() method.

Python has a garbage collector to clean up unreferenced objects but, we must not rely on it to close the file.

```
f = open("test.txt",encoding = 'utf-8')
```

```
# perform file operations

f.close()
```

This method is not entirely safe. If an exception occurs when we are performing some operation with the file, the code exits without closing the file.

A safer way is to use a try...finally block.

```
try:
   f = open("test.txt",encoding = 'utf-8')
   # perform file operations
finally:
f.close()
```

This way, we are guaranteed that the file is properly closed even if an exception is raised, causing program flow to stop. The best way to do this is using the with statement. This ensures that the file is closed when the block inside with is exited. **Don't need to explicitly call the close()** method. It is done internally.

```
with open("test.txt",encoding = 'utf-8') as f:
   # perform file operations
```

3. Reading files using read(), readline() and readlines():

To read a file in Python, we must open the file in reading mode. There are various methods available for this purpose.To read data, the file object provides the following methods:

Method	**Argument**
read([n])	Reads and returns n bytes or less (if there aren't enough characters to read) from the file as a string. Ifn not specified, it reads the entire file as a string andreturns it.
readline()	Reads and returns the characters until the end of theline is reached as a string.
readlines()	Reads and returns all the lines as a list of strings.

When the end of the file (EOF) is reached the read() and readline() methods returns an empty string, while readlines() returnsan empty list ([]).

To create a text file and save example.txt

```
example.txt
1 Welcome to python
2 This is text file
3 god bless you
```

Example:

```
f =  open("example.txt",  "r")
print(f.read(3)) # read the first 3 characters
print(f.read()) # read the remaining characters in the file.
print(f.readline()) # End of the file (EOF) is reached

f.close()
```

Output:

```
Wel
come to python
This is text file
god bless you
```

Seek() and tell() method:

seek() - change our current file cursor (position) using theseek() method.

```
f = open("example.txt", "r")
print(f.read(3)) # read the first 3 characters
print(f.read()) # read the remaining characters in the file.
print(f.readline()) # End of the file (EOF) is reached
print("current file position=",f.tell())# get the current file
position
f.seek(0)
print("seek() method - to bring file cursor to initial position
again")
print(f.read())
f.close()
```

Tell() - the tell() method returns our current position (innumber of bytes)

Output:

```
Wel
come to python
This is text file
god bless you

current file position= 51
seek() method - to bring file cursor to initial position again
Welcome to python
This is text file
god bless you
```

Writing Data using write() and writelines():

In order to write into a file in Python, we need to open it in write 'w', append 'a' or exclusive creation 'x' mode. Writing a string or sequence of bytes (for binary files) is done using write() method. This method returns the number of characters written to the file. We need to be careful with the 'w' mode as it will overwrite into the file if it already exists. All previous data are erased.

Method	Description
write(s)	Writes the string sto the file and returns the numbercharacters written.
writelines(s)	Writes all strings in the sequence sto the file.

Example:

```
f = open("example.txt", "w")
f.write("new content written\n")
f.write("god is great\n\n")
f.writelines("health is wealth")
f.close()
```

Output example.txt

```
new content written
god is great

health is wealth
```

This program will create a new file named 'example.txt' if it does not exist. If it does exist, it is overwritten. We must include the newline characters ourselves to distinguish different lines.

Append : Append the content at the end of the already existing file.

Example:

```
f = open("example.txt", "a")
f.write("\nnew content append\n")
f.write("happy\n\n")
f.writelines("welcome")
f.close()
data = open("example.txt").read()
print(data)
f.close()
```

Output: example.txt

```
new content written
god is great


health is  wealth
new content append
happy

welcome
```

Python File Methods:

There are various methods available with the file object. Some of them have been used in above examples.

Here is the complete list of methods in text mode with a briefdescription.

Python File Methods:

Method	Description
close()	Close an open file. It has no effect if the fileis already closed.
detach()	Separate the underlying binary buffer fromthe TextIOBase and return it.
fileno()	Return an integer number (file descriptor) ofthe file.
flush()	Flush the write buffer of the file stream.
isatty()	Return True if the file stream is interactive.
read(n)	Read atmost n characters form the file. Readstill end of file if it is negative or

	None.
readable()	Returns True if the file stream can be readfrom.
readline(n=-1)	Read and return one line from the file. Readsin at most n bytes if specified.
readlines(n=-1)	Read and return a list of lines from the file. Reads in at most n bytes/characters if specified.
seek (offset,from=SEE	Change the file position to offset bytes, inreference to from (start, current, end).
K_SET)	
seekable()	Returns True if the file stream supports random access.
tell()	Returns the current file location.

truncate (size=None)
 	
writable()
 	
write(s)
	Resize the file stream to size bytes. If size isnot specified, resize to current location.
	
	Returns True if the file stream can be writtento.
	
	Write string s to the file and return the number of characters written.

writelines(lines)	Write a list of lines to the file.

String processing in Python

Strings are sequences of characters. It contains enclosing characters in quotes. Python treats single quotes the same as doublequotes. There are numerous algorithms for processing strings, including for searching, sorting, comparing and transforming. Python strings are "immutable" which means they cannot be changed after they are created . To create a string, put the sequence of characters inside either single quotes, double quotes, or triple quotes.

Eample:

```
print('Hellow   World!')
print("Hellow World!")
print("""Sunday
        Monday
        Tuesday""")
```

Output:

```
Hellow   World!
Hellow   World!
Sunday
        Monday
        Tuesday
```

Access characters in a string:

In order ot access characters from String, use the square brackets [] for slicing along with the index or indices to obtain your characters. Python String index starts from 0.

Example:

```
str = 'Hellow World!'
print(str [0])
print(str [7])
print(str [0:6])
print(str [7:12])
```

Output:

```
H
W
Hellow
World
```

Updating Strings: The "update" an existing string by (re)assigning a variable to another string. The new value can be related to its previous value or to a completely different string altogether.

Example:

```
var1 = 'Hello World!'
print("Current String:",var1)
print ("After Updated at 6 th position of the   String : ", var1[:6] + 'Python')
```

Output:

```
Current String: Hello World!
After Updated at 6 th position of the String :  Hello Python
```

String Special Operators:

Assume string variable a = 'Hello' and variable b ='Python', then

Operator	Description	Example
+	Concatenation - Adds values on either side of the operator	a + b will give HelloPython
*	Repetition - Creates new strings, concatenating multiple copies of the same string	a*2 will give - HelloHello
[]	Slice - Gives the character from thegiven index	a[1] will give e
[:]	Range Slice - Gives the charactersfrom the given range	a[1:4] will give ell

In	Membership - Returns true if a character exists in the given string	H in a will give 1
not in	Membership - Returns true if a character does not exist in the givenstring	M not in a will give 1
r/R	Raw String - Suppresses actual meaning of Escape characters. The syntax for raw strings is exactly the same as for normal strings with the exception of the raw string operator, the letter "r," which precedes the quotation marks. The "r" can be lowercase (r) or uppercase (R) and must be placed immediately preceding the first quote mark.	print r'\n' prints \n and print R'\n'prints \n
%	Format - Performs String formatting	See at next section

String Formatting Operator:

One of Python's coolest features is the string format operator %. This operator is unique to strings and makes up for the pack ofhaving functions from C's printf() family.

Example :

```
print ("My name is %s and weight is %d kg!"%('Zara',21))
```

Output:

```
My name is Zara and weight is 21 kg!
```

Here is the list of complete set of symbols which can be usedalong with % −

Format Symbol	Conversion

%c	character
%s	string conversion via str() prior to formatting
%i	signed decimal integer
%d	signed decimal integer
%u	unsigned decimal integer
%o	octal integer
%x	hexadecimal integer (lowercase letters)
%X	hexadecimal integer (UPPERcase letters)
%e	exponential notation (with lowercase 'e')
%E	exponential notation (with UPPERcase 'E')
%f	floating point real number
%g	the shorter of %f and %e
%G	the shorter of %f and %E

String Built in Methods:

Method	Description
capitalize()	Converts the first character to upper case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
find()	Searches the string for a specified value and returns
	the position of where it was found
islower()	Returns True if all characters in the string are lower case
isupper()	Returns True if all characters in the string are upper case
join()	Joins the elements of an iterable to the end of thestring
ljust()	Returns a left justified version of the string

lower()	Converts a string into lower case
split()	Splits the string at the specified separator, and returns a list
swapcase()	Swaps cases, lower case becomes upper case andvice versa
title()	Converts the first character of each word to uppercase

Example:

Python String Concatenation: Concatenation is the operation of joining stuff together. Python Strings can join using theconcatenation operator **+**.

Example:

```
 a='Do you see this, '
 b='$$?'
print(a+b)
```

Output:

```
Do you see this, $$?
```

Let’s take another example.

```
 a='10'
print(2*a)
```

Output:

```
1010
```

Multiplying ‘a’ by 2 returned 1010, and not 20, because ‘10’ is a string, not a number. You cannot concatenate a string to a number.

a) **len():** The len() function returns the length of a string.

```
a='book'
print(len(a))
```

Output:

```
4
```

You can also use it to find how long a slice of the string is.

```
a='book'

print(len(a[2:]))
```

Output: 2

b) **str():** This function converts any data type into a string.

Example:

```
1.print(str(2+3j))

print (str(['red','green','blue']))
```

Output:

```
(2+3j)

['red', 'green', 'blue']
```

c) **lower() and upper():**

These methods return the string in lowercase anduppercase, respectively.

Example:

```
a='BOOK is Python'

print(a.lower())

print(a.upper())
```

Output:

```
book is python

BOOK IS PYTHON
```

d) **trip():** It removes whitespaces from the beginning and end ofthe string.

```
a='      Book '
print(a.strip())
```

Output:

```
Book
```

e) **isdigit():** Returns True if all characters in a string are digits.Otherwise return False.

Example:

```
a='777'
print(a.isdigit())
b='77a'
print(b.isdigit())
```

Output:

```
True
False
```

f) **isalpha():** Returns True if all characters in a string are charactersfrom an alphabet. Otherwise return false.

Example:

```
a='abc'
print(a.isalpha())
b='ab7'
print(b.isalpha())
```

Output:

```
True
False
```

g) **isspace():** Returns True if all characters in a string are spaces.Otherwise return false.

Example:

```
a=' '
print(a.isspace())
b=' \' '
print(b.isspace())
```

Output:

```
True
False
```

h) **startswith():** It takes a string as an argument, and returns True isthe string it is applied on begins with the string in the argument.Otherwise return False.

Example:

```
a='union'
print(a.startswith('un'))
print(a.startswith('io'))
```

Output:

```
True
False
```

i) **endswith():** It takes a string as an argument, and returns True if the string it is applied on ends with the string in the argument.Otherwise return False.

Example:

```
a='therefore'
print(a.endswith('fore'))
print(a.endswith('the'))
```

Output:

```
True
False
```

j) **find():** It takes an argument and searches for it in the string on which it is applied. It then returns the index of the substring. If the string doesn't exist in the main string, then the index it returns is -1.

Example:

```
print( 'homeowner'.find('meow'))
print('homeowner'.find('wow'))
```

Output:

```
2
-1
```

k) **replace():** It takes two arguments. The first is the substring to bereplaced. The second is the substring to replace with.

Example: `print('banana'.replace('na','ha'))`

Output: bahaha

l) **split():** It takes one argument. The string is then split aroundevery occurrence of the argument in the string.

Example: `print( 'No. Okay. Why?'.split('.'))`

Output: ['No', ' Okay', ' Why?']

m) **join():** It takes a list as an argument and joins the elements in thelist using the string it is applied on.

Example: `print("*".join(['red','green','blue']))`

Output: red*green*blue

Escape Characters:

Following table is a list of escape or non-printable characters that can be represented with backslash notation. An escape character gets interpreted; in a single quoted as well as double quoted strings.

ackslash notation	exadecimal character	Description
\a	0x07	Bell or alert

\b	0x08	Backspace
\n	0x0a	Newline
\nnn		Octal notation, where n is in the range 0.7
\r	0x0d	Carriage return
\s	0x20	Space
\t	0x09	Tab
\v	0x0b	Vertical tab
\x		Character x
\xnn		Hexadecimal notation, where n is in the range 0-9, a-f, or A-F

Simple program: To check given string is palindrome or not

```
string = input("Enter the String : ")
print("Length of string is:",len(string))
if(string == string[:: - 1]):
print(string," is a Palindrome")
else:
print(string," is Not a Palindrome")
```

Output:

```
Enter the String :amma
Length of string is: 4
amma is a Palindrome
Enter the String :hai
Length of string is: 3
hai is Not a Palindrome
```

Exception

A Python program terminates as soon as it encountersan error.

Types of error :

- Syntax error .
- An exception error .

Syntax Errors: Syntax errors occur when the parser detects anincorrect statement. Observe the following **Example:**

```
print0/0) # Here Missing parentheses in call to 'print'
```

Output error:

```
File "D:\workspace\HelloWorld\gayathri\exceptionh.py", line 1
    print 0/0)
          ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(0/0))?
```

Exception error: This type of error occurs whenever syntactically correct Python code results in an error. The last line of the message indicated what type of exception error you ran into.

```
Print (0/0) # division by zero
```

Output:

```
Traceback (most recent call last):
File "D:\workspace\HelloWorld\gayathri\exceptionh.py", line 1, in <module>
    print (0/0)
ZeroDivisionError: division by zero
```

Instead of showing the message exception error, Python details what type of exception error was encountered. In this case, itwas a ZeroDivisionError.

Python comes with various built-in exceptionsas well as the possibility to create self-defined exceptions.

What is Exception?

An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error.

When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits moral

Example:

```
a = 10
b = 0
print("Result of Division: " + str(a/b))
```

Output:

```
Traceback (most recent call last):
File "D:\workspace\HelloWorld\gayathri\exceptionh.py", line 3, in <module>
print("Result of Division: " + str(a/b))
ZeroDivisionError: division by zero
```

Handling Exceptions using try and except:

Exception handling is a concept used in Python to handle the exceptions and errors that occur during the execution of any program. Exceptions are unexpected errors that can occur during code execution.

For handling exceptions in Python we use two types of blocks:

- try block
- except block.

try block:

The tryblock is used to put the whole code that is to be executed in the program(which you think can lead to exception), if any exception occurs during execution of the code inside the tryblock, then it causes the execution of the code to be directed to the exceptblock and the execution that was going on in the tryblock is interrupted. But, if no exception occurs, then the whole tryblock is executed and the exceptblock is never executed.

except block :

The try block is generally followed by the except block which holds the exception cleanup code(*exception has occurred*, *how to effectively handle the situation*) like some print statement to **print some message** or may be **trigger some**

event or **store something in the database** etc.

except block, along with the keyword except we can also provide the **name of exception class** which is expected to occur. In case we do not provide any exception class name, it catches all the exceptions, otherwise it will only catch the exception of the type which is mentioned.

Syntax: try....except...else blocks −

```
try:
   You do your operations here;
   ......................
except ExceptionI:
   If there is ExceptionI, then execute this block.
except ExceptionII:
   If there is ExceptionII, then execute this block.
   ......................
else:
   If there is no exception then execute this block.
```

Here are few important points about the above-mentionedsyntax :

- ❖ A single try statement can have multiple except statements. This is useful when the try block contains statements that may throw different types of exceptions.
- ❖ You can also provide a generic except clause, which handlesany exception.
- ❖ After the except clause(s), you can include an else-clause. The code in the else-block executes if the code in the try: block does not raise an exception.
- ❖ The else-block is a good place for code that does not need thetry: block's protection.

Example:

```
try:
    a = 10
    b = 0
print("Result of Division: ",a/b)
except:
print("You have divided a number by zero, which is not allowed.")
```

Output:

```
You have divided a number by zero, which is not allowed.
```

The except Clause with No Exceptions:

This kind of a **try-except** statement catches all the exceptions that occur. Using this kind of try-except statement is not considered a good programming practice though, because it catches all exceptions but does not make the programmer identify the root causeof the problem that may occur.

Syntax:

```
try:
   You do your operations here;
   ......................
except:
   If there is any exception, then execute this block.
   ......................
else:
   If there is no exception then execute this block.
```

Example:

```
try:
    a = int(input("Enter A value:"))
    b = int(input("Enter B value:"))
    c= a/b
except:
print("You have divided a number by zero, which is not allowed.")
else:
print("Result of Division: ",c)
```

Output:

```
Enter A value:4
Enter B value:2
Result of Division: 2.0


Enter A value:4
Enter B value:0
You have divided a number by zero, which is not allowed.
```

The except Clause with Multiple Exceptions:

If think code may generate different exceptions in different situations and want to handle those exceptions individually, then have multiple except blocks. try to handle multiple possible exception cases using multiple except blocks. Mostly exceptions occur when user inputs are involved.

```
try:
Youdo your operations here;
......................
except(Exception1[,Exception2[,...ExceptionN]]]):
If there is any exception from the given exception list,
then execute this block.
......................
else:
If there isno exception then execute this block.
```

So let's take a simple example where we will ask user for twonumbers to perform division operation on them and show them the result.

Example:

```
try:
    a = int(input("Enter A value: "))
    b = int(input("Enter B valuer: "))
print("Result of Division: ",a/b)
# except block handling division by zero
except(ZeroDivisionError):
print("You have divided a number by zero, which is not
allowed.")
# except block handling wrong value type
except(ValueError):
print("You must enter integer value")
```

Output:

```
Enter A value: 4
Enter B valuer: 2
Result of Division: 2.0

Enter A value: 4
Enter B valuer: 0
You have divided a number by zero, which is not allowed.

Enter A value: 4
Enter B valuer: x
You must enter integer value
```

The try-finally Clause:

The finally code block is also a part of exception handling. When we handle exception using the try and except block, we can include a finally block at the end. It cannot use **else** clause as well along with a finally clause.

Syntax:

```
try:
Youdo your operations here;
......................
Due to any exception,this may be skipped.
finally:
This would always be executed.
......................
```

Example:

```
try:
    a = int(input("Enter A value: "))
    b = int(input("Enter B value: "))
print("Result of Division: ",a/b)
# except block handling division by zero
except(ZeroDivisionError):
print("You have divided a number by zero, which is not allowed.")
finally:
print("Code execution Wrap up!")
# outside the try-except block
print("Will this get printed?")
```

Output:

```
Enter A value: 4
Enter B value: 0
You have divided a number by zero, which is not allowed.
Code execution Wrap up!
Will this get printed?
```

Exception Handling : raise Keyword

While the try and except block are for handling exceptions,the raise keyword on the contrary is to raise an exception.

Syntax:

```
raise EXCEPTION_CLASS_NAME
```

We want to add a new validation for restricting user from inputting negative values. Then we can simply add a new conditionand use the raise keyword to raise an exception which is already handled.

Example:

```
a = int(input("Enter A value:"))
b = int(input("Enter B value:"))
try:
   # condition for checking for negative values
if a <0or b <0:
       # raising exception using raise keyword
raiseZeroDivisionError
print(a/b)
exceptZeroDivisionError:
print("Please enter valid integer value")
```

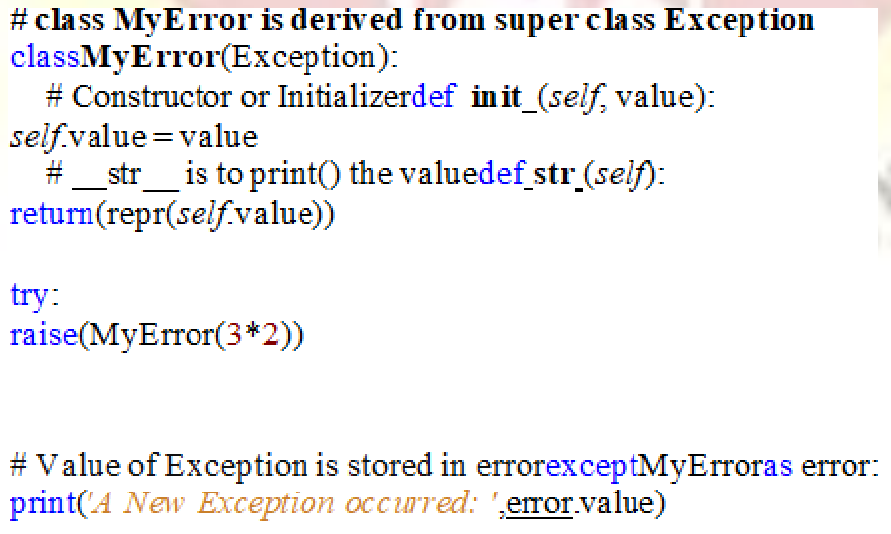
Output:

```
Enter A value:-1
Enter B value:0
Please enter valid integer value
```

User-defined Exceptions

Programmers may name their own exceptions by creating a new exception class. Exceptions need to be derived from the Exception class, either directly or indirectly. Although not mandatory, most of the exceptions are named as names that end in "Error" similar to naming of the standard exceptions in python.

Example:



```
# class MyError is derived from super class Exception
classMyError(Exception):
   # Constructor or Initializerdef init_(self, value):
self.value = value
   # __str__ is to print() the valuedef str (self):
return(repr(self.value))

try:
raise(MyError(3*2))


# Value of Exception is stored in errorexceptMyErroras error:
print('A New Exception occurred: ',error.value)
```

Output:

```
A New Exception occurred:  6
```

CONTROL STRUCTURES

- ❖ A control structure is just a decision that the computer makes.
- ❖ A control structure (or flow of control) is a block of programming that analyses variables and chooses a directionin which to go based on given parameters.
- ❖ It is the basic decision-making process in programming and flow of control determines how a computer program will respond when given certain conditions and parameters.
- ❖ A control structure is just a decision that the computer makes.
- ❖ There are two basic aspects of computer programming: dataand instructions.
- ❖ To work with data, you need to understand variables and data types; to work with instructions, you need to understand control structures and statements.
- ❖ Flow of control **three basic types** of control structures:

1. Sequential.
2. Selection .
3. Repetition.

Sequential

Sequential execution is when statements are executed oneafter another in order.

Selection

Selection used for decisions, branching - choosing between 2or more alternative paths.

1. if
2. if...else
3. switch

Repetition

Repetition used for looping, i.e. repeating a piece of codemultiple times in a row.

1. while loop
2. do..while loop
3. for loop

Boolean Expression:

Boolean represent one of two values: True or False.

Evaluate any expression will get one of two answers True or False.

Example:

```
print(10>9)
print(10 == 9)
print(10<9)
```

```
True
False
False
```

When you run a condition in an if statement, Pythonreturns True or False:

Example:

```
a=20
b=10
if b>a:
print("b is greater than a")
else:
print("b is not greater than a")
```

Output:

```
b is not greater than a
```

Evaluate Values and Variables:

The bool() function allows you to evaluate any value, andgive you True or False in return,

Example:

```
print(bool("Hello"))
print(bool(15))
```

Output:

```
True
True
```

Most Values are True:

Almost any value is evaluated to True if it has some sort ofcontent.

- ❖ Any string is True, except empty strings.
- ❖ Any number is True, except 0.
- ❖ Any list, tuple, set, and dictionary are True, except emptyones.

The following example will return True:

```
print(bool("abc"))
print(bool(123))
print(bool(["apple", "cherry", "banana"]))
```

Output:

```
True
True
True
```

Some Values are False:

In fact, there are not many values that evaluates to False, except empty values, such as (), [], {}, "", the number 0, and the value None. And of course the value False evaluates to False.

The following Example will return False:

```
print(bool(False))
print(bool(None))
print(bool(0))
print(bool(""))
print(bool(()))
print(bool([]))
print(bool({}))
```

OUTPUT

```
False
False
False
False
False
False
```

Selection Control Statement

- ❖ The selection statements are also known as decision making statements or branching statements or Conditional Statements.
- ❖ The selection statements are used to select a part of the program to be executed based on a condition.
- ❖ It require one or more conditions to be evaluated or tested by the program statement or statements to be executed if the condition is true or false. Python provides the following selection statements.

1. if statements
2. if else statements
3. if..elif..else statements
4. nested if statements
5. not operator in if statement
6. and operator in if statement
7. in operator in if statement

if statement

- ❖ If statement evaluates the test expression insideparenthesis.

- If test expression is evaluated to true (nonzero) ,statements inside the body of if is executed.
- If test expression is evaluated to false (0) , statementsinside the body of if is skipped.

Syntax:

```
if  expression:
     statements
```

Example:

```
x=20
y=10
if x >y :
print(" X is bigger ")
```

Output:

```
X is bigger
```

In this program we have two variables x and y. x is assigned as the value 20 and y is 10. In next line, the if statement evaluate the expression (x>y) is true or false. In this case the x > y is true becausex=20 and y=10, then the control goes to the body of if block and print the message "X is bigger". If the condition is false then the control goes outside the if block.

Python if..else statements

The else statement is to specify a block of code to be executed, if the condition in the if statement is false. Thus, the else clause ensures that a sequence of statements is executed.

Syntax:

```
if expression:
   statements
else:
   statements
```

Example:

```
x=10
y=20
if x >y :
print(" X is bigger ")
else :
print(" Y is bigger ")
```

Output

```
Y is bigger
```

In the above code, the if stat evaluate the expression is true or false. In this case the x > y is false, then the control goes to the body of else block , so the program will execute the code inside elseblock.

Indentation in Python

- Indentation in Python refers to the (spaces and tabs) that areused at the beginning of a statement.
- The statements with the same indentation belong to the samegroup called a suite.
- By default, Python uses four spaces for indentation, and theprogrammer can manage it.

Consider the example of a correctly indented Python codestatement mentioned below.

Example

```
a=int(input("Enter a 1 or 2 value:"))
if a==1:
print("one")
if a==2:
```

```
print("two")
print('end')
```

Output:

```
Enter a 1 or 2 value:1
one
end
```

In the above code, the first and last line of the statement is related to the same suite because there is no indentation in front of them. So after executing first "if statement", the Python interpreter will go into the next statement. If the condition is not true, it will execute the last line of the statement.

Multi Way Selection (if..elif..else statements:)

- The most commonly used multiple selection technique is acombination of if and if…else statements.
- This form of selection is often called a selection tree becauseof its resemblance to the branches of a tree.
- In this case, you follow a particular path to obtain a desiredresult.
- The elif is short for else if and is useful to avoid excessiveindentation.

Syntax:

```
if expression:
  statements
elif expression:
statements
else:
  statements
```

In the above case Python evaluates each expression one by one and if a true condition is found the statement(s) block under thatexpression will be executed. If no true condition is found the statement(s) block under else will be executed.

Example:

```
x=500
if x >500 :
print(" X is greater than 500 ")
elif x <500 :
print(" X is less than 500 ")
elif x == 500 :
print(" X is 500 ")
else :
print(" X is not a number ")
```

Output:

```
X is 500
```

Nested if statements:

In some situations you have to place an if statement insideanother statement.

Syntax:

```
if condition:
if condition:
statements
else:
statements
else:
  statements
```

Example:

```
mark = 72
if mark >50:
if mark >=80:
print ("You got A Grade !!")
elif mark>=60and mark<80 :
print ("You got B Grade !!")
else:
print ("You got C Grade !!")
else:
print("You failed!!")
```

Output:

```
You got B Grade !!
```

not operator in if statement:

By using Not keyword we can change the meaning of theexpressions, moreover we can invert an expression.

Example:

```
mark = 100
ifnot (mark == 100):
print("mark is not 100")
else:
print("mark is 100")
```

Output:

```
mark is 100
```

Write same code using "!=" operator.

Example:

```
mark = 100
if (mark != 100):
print("mark is not 100")
else:
print("mark is 100")
```

Output:

```
mark is 100
```

In operator in if statement:Example

```
color = ['Red','Blue','Green']
selColor = "Red"
ifselColorin  color:
print("Red is in the list")
else:
print("Not in the list")
```

Output:

```
Red is in the list
```

UNIT IV

Iteration Control

- Iteration statements or loop statements allow us to execute ablock of statements as long as the condition is true.
- Loops statements are used when we need to run same codeagain and again, each time with a different value.
- Loops are one of the most important features in computerprogramming languages.
- It offer a quick and easy way to do something repeated until acertain condition is reached.

Every loop has 3 parts:

- Initialization
- Condition
- Updation

In Python Iteration (Loops) statements are of three type :-

1. While Loop
2. For Loop
3. Nested For Loops

Loops

The loop construct in Python allows you to repeat a body ofcode several times.

There are two types of loops :

- ❖ Definite loops
- ❖ Indefinite loops.

Definite loops :

You use a definite loop when you know a priory how many times you will be executing the body of the loop. You use key word***for*** to begin such a loop.

Indefinite loop :

In an indefinite loop is the number of times it is going to execute is not known in advance and it is going to be executed until some condition is satisfied. Use the keyword ***while*** to begin indefinite loops.

while Statement(Infinite loop)

- ❖ while loop is a control flow statement that allows code to beexecuted repeatedly based on a given Boolean condition.
- ❖ While loop tells the computer to do something as long as thecondition is met.
- ❖ It consists of condition/expression and a block of code.
- ❖ The condition/expression is evaluated, and if the condition/expression is true, the code within the block is executed.
- ❖ This repeats until the condition/expression becomes false.

Syntax:

```
while (condition) :
 statement(s)
```

Initialize the value of a variable and set the condition, test the condition in while clause, if it holds true, the body of the loop is executed. While executing the body of loop it can update the statement inside while loop. After updating, the condition is checked again. This process is repeated as long as the condition is true and once the condition becomes false the program breaks out of the loop.

Example:

```
n=int(input("Enter the n value:"))
i=1
while(i<=n):
```

```
print(i)
i+=1
```

Output:

```
Enter the n value:5
1
2
3
4
5
```

Here the conditional of x < =5 (while(x < =5):) and x was previously declared and set equal to 1 (x=1). So, the first item printed out was 1 (print(x)), which makes sense. In the next line x+=1 means x = x+1, now the value of x = 2. After updating x , thecondition is checked again. This process is repeated as long as the condition is true and once the condition becomes false the program breaks out of the loop . Of course, once a becomes equal to 5, we will no longer run through the loop.

break and continue:

Python provides two keywords that terminate a loop iterationprematurely: break and continue.

1. break leaves a loop.
2. continue jumps to the next iteration.

break statement in Python while loop:

Sometimes it's necessary to exit from a Python while loop before the loop has finished fully iterating over all the step values. This is typically achieved by a "break" statement.

Example:

```
x=10

whileTrue:
print (x)
    x+=2;

if x >20:

break

print("After Break")
```

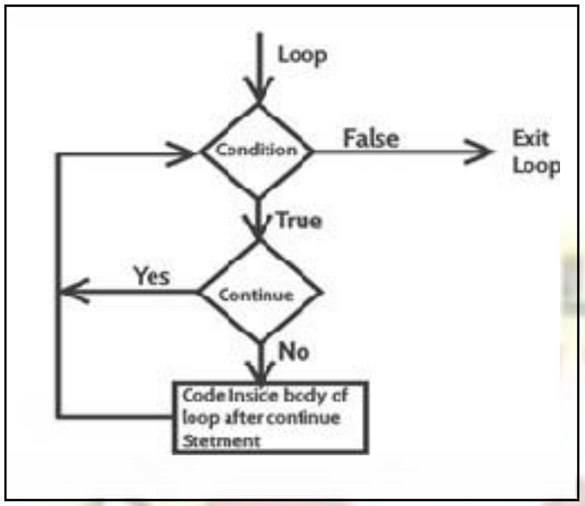
Output:

```
10
12
14
16
18
20
After Break
```

In the above example, when the condition x>20, the break statement executed and immediately terminated the while loop and the program control resumes at the next statement.

continue statement in Python while loop:

The continue statement in Python while loop is used when we want to skip one or more statements in loop's body and to transfer the control to the next iteration.



Example:

```
x=0
while x <50:
    x+=10
if x==30:
continue
print (x)
print("Loop Over")
```

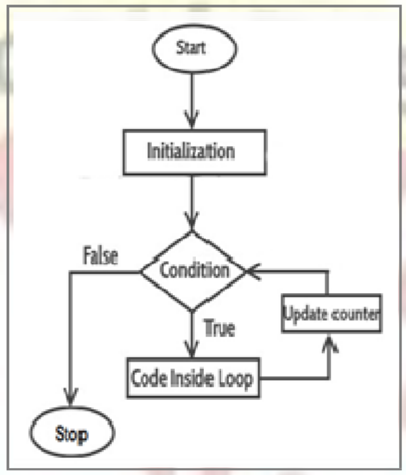
Output:

```
10
20
40
50
Loop Over
```

In the above example, we can see in the output the 30 is missing. It is because when the condition x==30 the loop encounter the continue statement and control go back to start of the loop.

for Loop (Definite loop)

A loop is a fundamental programming idea that is commonly used in writing computer programs. It is a sequence of instructions that is repeated until a certain condition is reached. A for loop has two sections: a header specifying the iterating conditions, and a bodywhich is executed once per iteration. The header often declares an explicit loop counter or loop variable, which allows the body to know which iteration is being executed.



Syntax:

```
for item in sequence:
    statements(s)
```

for loop range() function:

The range function in for loop is actually a very powerful mechanism when it comes to creating sequences of integers. It can take one, two, or three parameters. It returns or generates a list of integers from some lower bound(zero, by default) up to (but not including) some upper bound , possibly in increments (steps) of some other number (one, by default). Note for Python 3 users: There are no separate range and xrange()functions in Python 3, there is just range, which follows the design of Python 2's xrange.

range(stop)

1. range(start,stop)
2. range(start,stop,step)

It is important to note that all parameters must be integersand can be positive or negative.

Python range() function with one parameters:Syntax

range(stop) stop:

Generate numbers up to, but not including this number.

```
for n inrange(5):
print(n)
```

Output:

```
0
1
2
3
4
```

Python range() function with two parametersSyntax:

```
range(start,stop)
```

start: Starting number of the sequence.

stop: Generate numbers up to, but not including this number.

Example:

```
for n inrange(5,10):
print(n)
```

Output:

```
5
6
7
8
9
```

The range(start,stop) generates a sequence with numbersstart, start + 1, ..., stop - 1. The last number is not included.

Python range() function with three parameters:

1. start: Starting number of the sequence.
2. stop: Generate numbers up to, but not including this number.

3. step: Difference between each number in the sequence.

Example:

```
for n inrange(0,10,3):
print(n)
```

Output:

```
0
3
6
9
```

Here the start value is 0 and end values is 10 and step is 3. This means that the loop start from 0 and end at 10 and the increment value is 3.

Python Range() function can define an empty sequence, like range(-10) or range(10, 4). In this case the for-block won't beexecuted:

Example:

```
foriinrange(-10):
print('Python range()')
```

The above code won't be executed.

Also, you can use Python range() for repeat some actionseveral times:

Example:

```
foriinrange(2 ** 2):
print('Python range()!!')
```

Output:

```
Python range()!!
Python range()!!
Python range()!!
Python range()!!
```

Decrementing for loops:

If you want a decrementing for loops, you need to give therange a -1 step

Example:

```
foriinrange(5,0,-1):
print (i)
```

Output:

```
5
4
3
2
1
```

Accessing the index in 'for' loops in Python:

Python's built-in enumerate function allows developers to loop over a list and retrieve both the index and the value of each item in the containing list. It reduces the visual clutter by hiding the accounting for the indexes, and encapsulating the iterable into another iterable that yields a two-item tuple of the index and the item that the original iterable would provide.

Example:

```
months = ["January", "February", "March", "April", "May", "June", "July","August", "September", "October", "November", "December"]
foridx, mNameinenumerate(months, start=1):
print("Months {}: {}".format(idx, mName))
```

Output:

```
Months 1: January
Months 2: February
Months 3: March
Months 4: April
Months 5: May
Months 6: June
Months 7: July
Months 8: August
Months 9: September
Months 10: October
Months 11: November
Months 12: December
```

Note: The start=1 option to enumerate here is optional. If we didn't specify this, we'd start counting at 0 by default. Python enumerate function perform an iterable where each element is a tuple that contains the index of the item and the originalitem value.

So, this function is meant for:

1. Accessing each item in a list (or another iterable).
2. Also getting the index of each item accessed.

Iterate over two lists simultaneously:

In the following Python program we're looping over two listsat the same time using indexes to look up corresponding elements.

Example:

```
grades = ["High", "Medium", "Low"]
values = [0.75, 0.50, 0.25]
fori, grade in enumerate(grades):
    value = values[i]
print("{}% {}".format(value * 100, grade)
```

Output:

```
75.0% High
50.0% Medium
25.0% Low
```

Using Python zip() in for loop:

The Python zip() function takes multiple lists and returns an iterable that provides a tuple of the corresponding elements of each list as we loop over it.

Example:

```
grades = ["High", "Medium", "Low"]
values = [0.75, 0.50, 0.25]
for grade, value inzip(grades, values):
print("{}% {}".format(value * 100, grade))
```

Output:

```
75.0% High
50.0% Medium
25.0% Low
```

Nested for loop in Python:

A for loop contain inside of another for loop that is callednested for loop.

Syntax:

```
for iterating_var in sequence:
   for iterating_var in sequence:
      statements(s)
statements(s)
```

Example:

```
n=int(input("Enter the number:"))
foriin range(1,n+1):
for j in range(1,int(i)+1):
print(i, end=' ')
print("\n")
```

Output:

```
1

2 2

3 3 3

4 4 4 4

5 5 5 5 5
```

Infinite Loops

An Infinite Loop in Python is a continuous repetitive conditional loop that gets executed until an external factor interfere in the execution flow, like insufficient CPU memory, a failed feature/ error code that stopped the execution, or a new feature in the other legacy systems that needs code integration. There are a few types of Infinite Loop in Python, that includes, the While statement, the If statement, the Continue statement and the Break statement.

When are Infinite Loops Necessary?

An infinite loop may be useful in client/server programming where the server needs to run with continuity so that the client programs may communicate with the server program whenever the necessity arises. It may also be helpful if a new connection needs to be created. There is the utility of a while loop in gaming application or an application where we enter some sort of main event loop which continues to run until the user selects an action to break that infinite loop. Also, if one has to play a game and wishes the game to reset after each session. Iterations are the process of doing a repetitive taskand computer programs have always mastered this art.

How would we Run an Infinite Loop by Mistake?

It is a very simple program but loop may surely miss out onthese basic steps and have an infinite loop running in their program.

Example:

```
i=0

whilei<10:

print("Welcome")
```

Output:

```
Welcome

Welcome

Welcome

Welcome #non stop printing Welcome
```

As there is no code to increment the value of the integer, itwill continue to print that until we terminate the program.

So, to avoid the unintentional loop, we add the following lineto the code.

Example:

```
i=0

whilei<3:

print("Welcome")

i=i+1
```

Output:

```
Welcome

Welcome

Welcome
```

Definite Loop Vs Indefinite Loop

- ❖ A loop is a block of code that would repeat for a specified number of times or until some condition is satisfied.
- ❖ A **definite loop** is a loop in which the **number of times** it isgoing to execute is **known** in advance before entering the loop.
- ❖ In an **indefinite loop** is the **number of times** it is going to execute is **not known** in advance and it is going to beexecuted until some condition is satisfied.

What is a Definite Loop?

A definite loop is a loop in which the number of times it is going to execute is known in advance before entering the loop. The number of iterations it is going to repeat will be typically provided through an integer variable. In general, for loops are considered to bedefinite loops.

What is an Indefinite Loop?

In an indefinite loop, the number of times it is going to execute is not known in advance. Typically, an indefinite loop is going to be executed until some condition is satisfied. While loops and do-while loops are commonly used to implement indefinite loops. Even though there is no specific reason for not using for loops for constructing indefinite loops, indefinite loops could be organizedneatly using while loops. Some of common examples that you wouldneed to implement indefinite loops are prompting for reading an input until user inserts a positive integer, reading a password until the user inserts the same password twice in a row, etc.

What is the difference between Definite Loop and Indefinite Loop?

A definite loop is a loop in which the number of times it is going to execute is known in advance before entering the loop, while an indefinite loop is executed until some condition is satisfied and the number of times it is going to execute is not known in advance. Often, definite loops are implemented using for loops and indefiniteloops are implemented using while loops and do-while loops. But there is no theoretical reason for not using for loops for indefinite loops and while loops for definite loops. But indefinite loops could be neatly organized with while loops, while definite loops could be neatly organized with for loops.

Boolean Flags And Indefinite LoopBoolean Flags:

Flag variable is used as a signal in programming to let the program know that a certain condition has met. It usually acts as a boolean variable indicating a condition to be either true or false.

Indefinite loops:

The number of iterations is not known before we start to execute the body of the loop, but depends on when a certain condition becomes true (and this depends on what happens in the body of the loop)

Example:

While the user does not decide it is time to stop, printout a * and ask the user whether wants to stop.

- ❖ In Python, While Loops is used to execute a block of statements repeatedly until a given condition is satisfied.
- ❖ And when the condition becomes false, the line immediately after the loop in the program is executed. While loop falls under the category of indefinite iteration.
- ❖ Indefinite iteration means that the number of times the loop is executed isn’t specified explicitly in advance.

Example: To find given number is prime or not

```
n = int(input("Enter any number: "))
flag=True
foriinrange(2, n):
if (n%i)==0:
while flag:
            flag=False
break
else:
         flag=True
if flag==False:
print(n, " is a not a prime number")
else:
print(n, "is a prime number")
```

Output:

```
Enter any number: 6
6 is a not a prime number
Enter any number: 11
11 is a prime number
```

UNIT V

PYTHON NAMESPACE AND VARIABLE SCOPE RESOLUTION

What is Name in Python?

- Name (also called identifier) is simply a name given toobjects.
- Everything in Python is an object.
- Name is a way to access the underlying object.

For example, when we do the assignment a = 2, 2 is an object storedin memory and a is the name we associate it with. We can get the address (in RAM) of some object through the built-in function id().

Example:

```
# Note: You may get different values for the ida = 2
print('id(2) =', id(2))
print('id(a) =', id(a))
```

Output:

```
id(2) = 9302208
id(a) = 9302208
```

Here, both refer to the same object 2, so they have the sameid(). Let's make things a little more interesting.

Example:

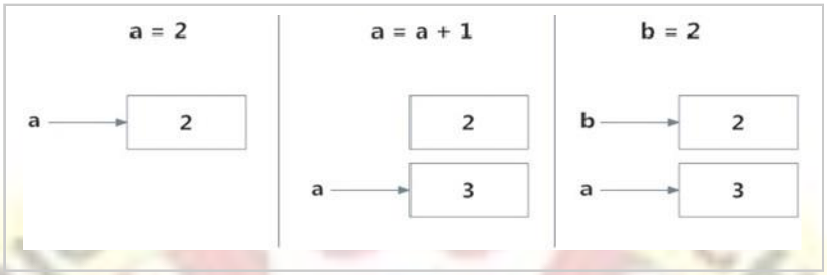
```
# Note: You may get different values for the ida = 2
print('id(a) =', id(a))a = a+1
print('id(a) =', id(a))
print('id(3) =', id(3))b = 2
print('id(b) =', id(b))
print('id(2) =', id(2))
```

Output:

```
id(a) = 9302208
id(a) = 9302240
id(3) = 9302240
id(b) = 9302208
```

id(2) = 9302208

What is happening in the above sequence of steps? Let's use a diagram to explain this:



Initially, an object 2 is created and the name a is associated with it, when we do a = a+1, a new object 3 is created and now a is associated with this object.

Note that id(a) and id(3) have the same values.

Furthermore, when b = 2 is executed, the new name b gets associated with the previous object 2.

What is Python Namespace?

- A namespace is a system to have a unique name for each andevery object in Python.
- Python namespaces are containers to map names to objects.
- An object might be a variable or a method.
- In Python, everything is an object and we specify a name to the object so that we can access it later on.
- Python itself maintains a namespace in the form of a Pythondictionary.
- You can think of namespace as a dictionary of key-value pairs where the key is the variable name and the value is theobject associated with it.

Real-time example, the role of a namespace is like a surname. One might not find a single “Alice” in the class there might be multiple “Alice” but when you particularly ask for “Alice Lee” or“Alice Clark” (with a surname), there will be only one (time being don’t think of both first name and surname are same for multiple students).

On the similar lines, Python interpreter understands what exact method or variable one is trying to point to in the code, depending upon the namespace. So, the division of the word itself gives little more information. Its Name (which means name, an unique identifier) + Space(which talks something related to scope). Here, a name might be of any Python method or variable and space depends upon the location from where is trying to access a variableor a method.

Example:

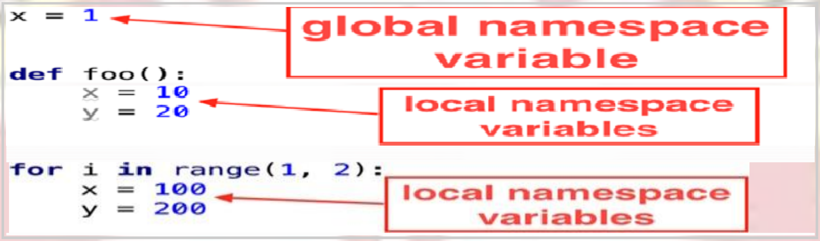
namespace = {"name1":object1, "name2":object2}

- In Python, multiple independent namespaces can exist at thesame time.
  - The variable names can be reused in these namespaces.

function_namespace = {"name1":object1, "name2":object2}for_loop_namespace =

{"name1":object3, "name2":object4}

Let's look at a simple example where we have multiplenamespaces.



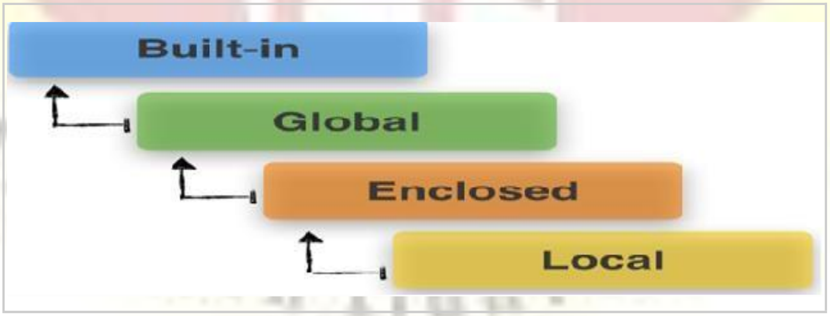
```
x = 1

def foo():
    x = 10
    y = 20

for i in range(1, 2):
    x = 100
    y = 200
```

global namespace variable
local namespace variables
local namespace variables

Namespace Types and LifecycleTypes of namespaces

Python namespaces can be divided into four types:

1. **Built-In**
2. **Global**
3. **Enclosing**
4. **Local**



❖ These have differing lifetimes.

❖ As Python executes a program, it creates namespaces asnecessary and deletes them when they're no longer needed.

❖ Typically, many namespaces will exist at any given time.

1. **Local Namespace:**

❖ A function, for-loop, try-except block are some examples ofa local namespace.

❖ The local namespace is deleted when the function or the codeblock finishes its execution.

2. **Enclosed Namespace:**

❖ When a function is defined inside a function, it creates anenclosed namespace.

❖ Its lifecycle is the same as the local namespace.

3. **Global Namespace:**

❖ The global namespace contains any names defined at the level of the main program.

❖ Python creates the global namespace when the main programbody starts, and it remains in existence until the interpreter terminates.

❖ Strictly speaking, this may not be the only global namespacethat exists.

❖ The interpreter also creates a global namespace for anymodule that your program loads with the import statement.

For further reading on main functions and modules in Python,see these resources:

❖ Defining Main Functions in Python

❖ Python Modules and Packages

4. **Built-in Namespace:**

The built-in namespace contains the names of all ofPython’s built-in objects.

These are available at all times when Python is running.

You can list the objects in the built-in namespace with the following command: built-in functions like max() and len(), andobject types like int and str.

Example:

```
>>> dir(_builtins_)

['ArithmeticError', 'AssertionError', 'AttributeError',

 'BaseException','BlockingIOError',                    'BrokenPipeError',
'BufferError',

'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError',

'ConnectionError','ConnectionRefusedError', 'ConnectionResetError',

'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError','Exception', 'False',

'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'FutureWarning',

'GeneratorExit', 'IOError','ImportError', 'ImportWarning', 'IndentationError',

'IndexError',

 'InterruptedError',                  'IsADirectoryError',    'KeyError',
'KeyboardInterrupt',

 'LookupError',                  'MemoryError',   'ModuleNotFoundError',
'NameError', 'None',

 'NotADirectoryError', 'NotImplemented', 'NotImplementedError','OSError',

'OverflowError', 'PendingDeprecationWarning', 'PermissionError',

 'ProcessLookupError',                'RecursionError', 'ReferenceError',
'ResourceWarning',
```

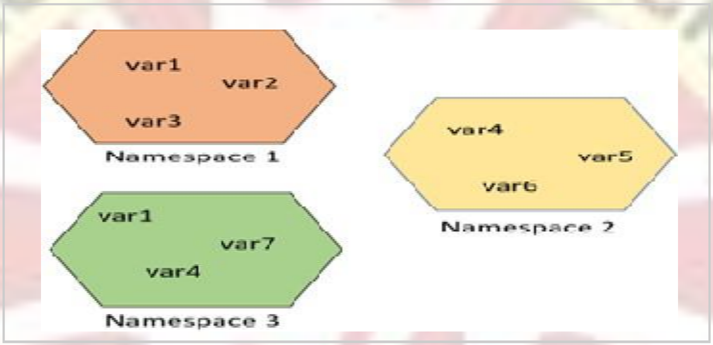
Lifetime Namespace

A lifetime of a namespace depends upon the scope of objects,if the scope of an object ends, the lifetime of that namespaces comes to an end.

Hence it is not possible to access inner namespaces's objectfrom an outer namespace.

Example:

```
# var1 is in the global namespacevar1 = 5
def some_func():
# var2 is in the local namespacevar2 = 6
def some_inner_func():
# var3 is in the nested local# namespace
var3 = 7
```



```
# global variablecount = 5
def some_method():
global count count = count + 1print(count)
some_method()
```

Output:

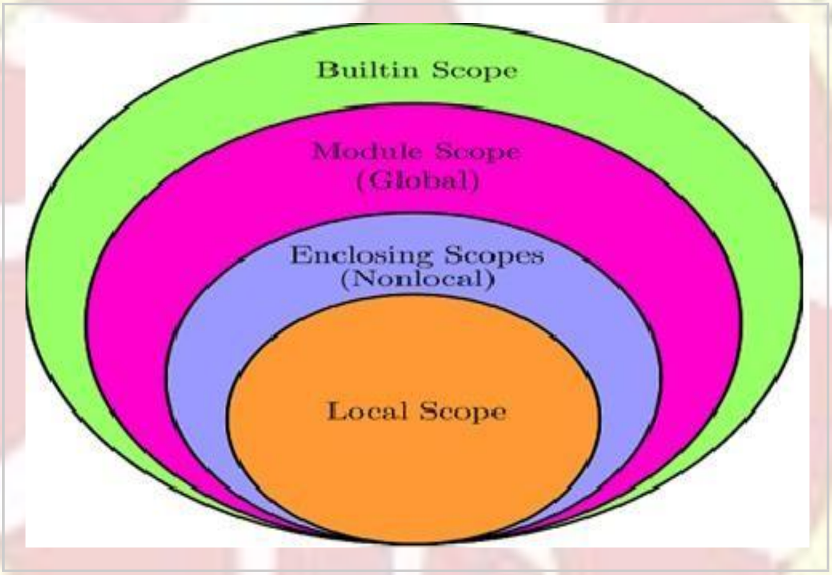
```
6
```

Python Scopes

- A variable is only available from inside the region it is created. This is called **scope**.
- In the context of Python namespaces, a "**scope**" is the collection of names associated with a particular environment.
- A scope defines the hierarchical order in which the namespaces have to be searched in order to obtain the mappings of *name-to-object*(variables).
- It is a context in which variables exist and from which they are referenced.

It defines the accessibility and the lifetime of a variable.

Python has the following scopes:



When a name is referenced in Python, the interpreter searches for it in the namespaces starting from the smallest scope in the above diagram, and progressively moves outward until Python either finds the name or raises a NameError exception.

Python Variable Scope

A scope is the portion of a program from where a namespacecan be accessed directly without any prefix.

At any given moment, there are at least three nested scopes.

1. Scope of the current function which has local names
2. Scope of the module which has global names
3. Outermost scope which has built-in names

When a reference is made inside a function, the name is searched in the local namespace, then in the global namespace and finally in the built-in namespace.

a. Local Scope

- ❖ Local scope refers to variables defined in current function.Always, a function will first look up for a variable name in its local scope.
- ❖ Only if it does not find it there, the outer scopes are checked.

Example

A variable created inside a function is available insidethat function:

Example:

```
def myfunc():x = 300
print(x)myfunc()
```

Output:

```
300
```

b. Global Scope

- ❖ A variable created in the main body of the Python code is aglobal variable and belongs to the global scope.
- ❖ Global variables are available from within any scope, globaland local.

A variable created outside of a function is global and canbe used by anyone.

Example:

```
x = 300
def myfunc():print(x) myfunc()
print(x)
```

Output:

```
300
300
```

c. Naming Variables

If you operate with the same variable name inside and outside of a function, Python will treat them as two separate variables, one available in the global scope (outside the function)and one available in the local scope (inside the function):

Example:The function will print the local x, and then thecode will print the global x:

```
x = 300
def myfunc():x = 200
```

```
print(x)myfunc()print(x)
```

Output:

```
200
300
```

d. Global Keyword

- If you need to create a global variable, but are stuck in thelocal scope, you can use the global keyword.
- The global keyword makes the variable global.

Example: If you use the global keyword, the variable belongs tothe global scope:

```
def myfunc():
global xx = 300
myfunc()print(x)
```

Output:

```
300
```

- Use the global keyword if you want to make a change to aglobal variable inside a function.
- To change the value of a global variable inside a function,refer to the variable by using the global keyword.

Example

```
x = 300
def myfunc():global x
x = 200
myfunc()print(x)
```

Output:

```
200
```

Example of Scope and Namespace in Python

```
def outer_function():a = 20
def inner_function():a = 30
    print('a =', a) inner_function()print('a =', a)
a = 10
```

outer_function()print('a =', a)

Output:

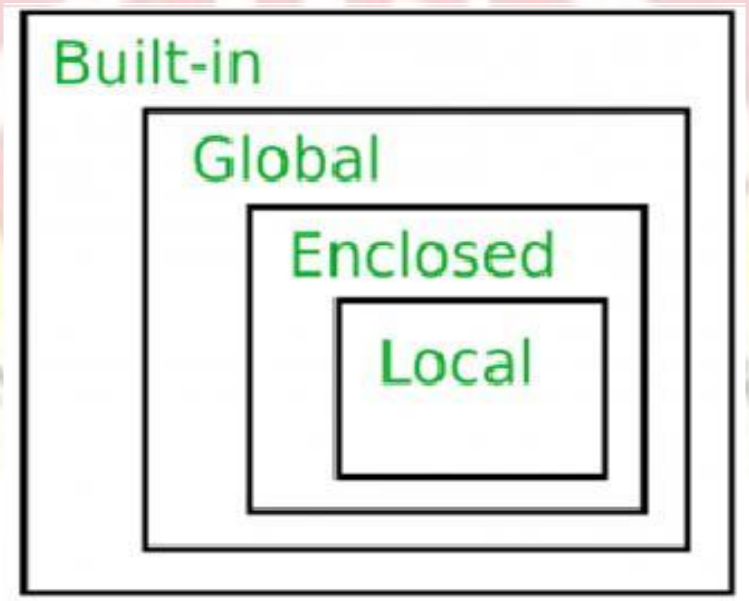
```
a = 30
a = 20
a = 10
```

Scope resolution via LEGB rule

- ❖ In Python, the LEGB rule is used to decide the order in which the namespaces are to be searched for scope resolution.
- ❖ The scopes are listed below in terms of hierarchy(highest to lowest/narrowest to broadest):

1. **Local(L)**: Defined inside function/class
2. **Enclosed(E):** Defined inside enclosing functions(Nested function concept)
3. **Global(G):** Defined at the uppermost level
4. **Built-in(B):** Reserved names in Python builtin modules

Local → Enclosed → Global → Built-in



- ❖ This is also called LEGB rule for variable scope resolution.

Python Variable Scope Resolution – LEGB Rule

- ❖ If a name is not found in the namespace hierarchy,NameError is raised.

❖ When we create an object or import a module, we create aseparate namespace for them.

❖ We can access their variables using the dot operator.

Local and Global Scopes :

If a variable is not defined in local scope, then, it is checkedfor in the higher scope, in this case, the global scope.

Example:

```
# Global Scope

pi = 'global pi variable'def inner():

pi = 'inner pi variable'print(pi)

inner() print(pi)
```

Output:

```
inner pi variable global pi variable
```

- Therefore, as expected the program prints out the value in thelocal scope on execution of inner().
- It is because it is defined inside the function and that is the first place where the variable is looked up. The pi value in global scope is printed on execution of print(pi) on line 9.

Local, Enclosed and Global Scopes :

For the enclosed scope, we need to define an outer function enclosing the inner function, comment out the local pi variable of inner function and refer to pi using the nonlocal keyword.

Example:

```
pi = 'global pi variable'def outer():

pi = 'outer pi variable'def inner():

local pi

print(pi)

inner()

outer()

print(pi)
variable'nonlocal pi
print(pi)
```

Output:

outer pi variable global pi variable

Local,Enclosed,Global and Built-in Scopes :

The final check can be done by importing pi from math module and commenting the global, enclosed and local pi variablesas shown below:

Example:

# Built-in Scope from math import pi

# pi = 'global pi variable'def outer():

# pi = 'outer pi variable'def inner():

# pi = 'inner pi variable'print(pi)

inner()

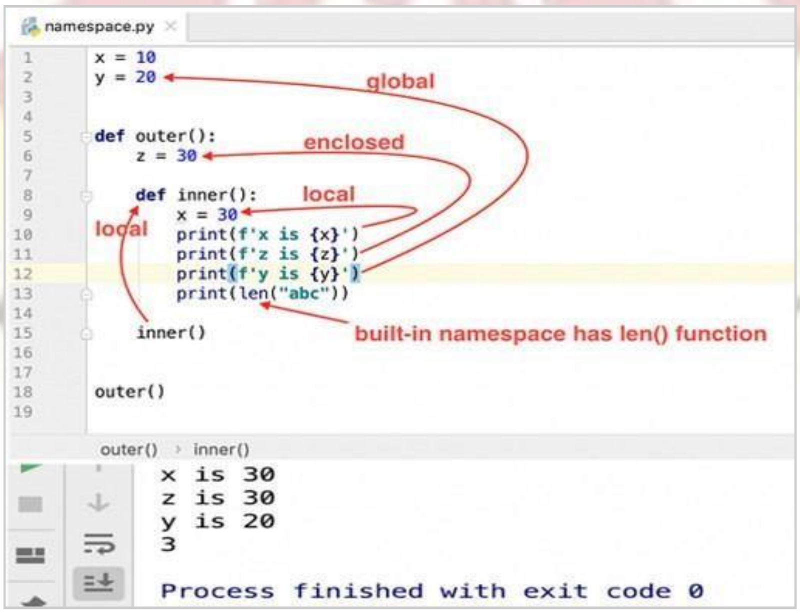
outer()

Output:

3.141592653589793

Since, pi is not defined in either local, enclosed or global scope, the built-in scope is looked up i.e the pi value imported frommath module.

The program is able to find the value of pi in the outermost scope, the following output is obtained.



```
x = 10
y = 20


def outer():
    z = 30

    def inner():
        x = 30
        print(f'x is {x}')
        print(f'z is {z}')
        print(f'y is {y}')
        print(len("abc"))

    inner()


outer()
```

```
x is 30
z is 30
y is 20
3

Process finished with exit code 0
```

Python eval()

- eval() is a built-in function or methodused in python, towhich we pass an expression.
- It parses this expression and runs

Uses of Python:

1. To allow users to enter own script to allow customization of acomplex system’s behavior.
2. To evaluate mathematical expressions in application insteadof writing an expression parser.

To evaluate a string based expression, python’s eval functionruns the following steps:

- Parse expression
- Compile it to bytecode
- Evaluate it as a python expression
- Return the result of the evaluation

In simple eval() is, when we pass any python expression as a string to the eval function, it evaluates the expression and returns theresult as an integer or float.

Syntax:

eval(expression, global, local)where,

expression- a string that will be evaluated as python code. global- Optional, a dictionary contains global parameters. local-Optional, a dictionary contains local parameters.

Example 1: passing expression to add two local variables

a=20b=30

res=eval(‘a+b’)print(res)

Output:

50

In this example , we have passed an expression ‘a+b’ to the eval() function in order to add two local variables: a,b.

Example 2: python eval() function with user input

Num1=int(input()) Num2=int(input()) Mult=eval(‘num1*num2’)

Print(‘multiplication:’,mult)

Output:

```
30
20
Multiplication: 600
```

In this example, we have accepted the input from the user and assigned the same to the variables. Further, it passed the expression for the multiplication of those two input values.