**PYTHON**

**Introduction to Python**

* Python is a high-level, general purpose, interpreted, and object-oriented programming language.
* It was developed by **Guido Van Rossum** of Netherlands’ in 1991.
* Python supports various programming approaches such as object-oriented, and functional or procedural programming like C.
* Python is easy to learn and versatile scripting language.
* It can interact with databases like Oracle, MySQL, SQL server etc.
* It is used almost in all areas from business Web sites to online games.
* Python is also used in **technical area**s - such *as*[*Machine Learning*](https://www.javatpoint.com/machine-learning)*,*[*Artificial Intelligence*](https://www.javatpoint.com/artificial-intelligence-tutorial)*, Web Development,*[*Mobile Application*](https://www.javatpoint.com/javatpoint.com/mobile-application-testing)*, Desktop Application, Scientific Calculation*, etc.
* Python is a programming language that can be applied to many different platforms ie the program developed in one operating system can be executed in another operating system. This is called platform independent.
* Python provides various Libraries which are cross-platform compatible and can run on platforms such as Windows, Linux, or macOS.
* It supports automatic garbage collection.

**Features of Python**

* 1. Simplicity
  2. Interpreted
  3. Object oriented
  4. Open source
  5. Portable
  6. Embeddable
  7. Extensible
  8. Platform independent
  9. Dynamically Typed Language
  10. Standard Libraries
  11. Expressive Language
  12. GUI support

1. **Simplicity**

* Python is easy to learn as compared with other programming languages like C, C++ and Java.
* Its syntax is similar to English language.
* There is no use of the semicolon or curly-bracket.

1. **Interpreted**

* Python is an interpreted language; it means the Python program is executed one line at a time.
* Python programs are directly executed without compilation.

1. **Object oriented**

* Python support object oriented programming also. It provides security for the data.
* In Python, we can work with classes and objects, and also Inheritance, polymorphism etc.

1. **Open source**

* It means the python code can be downloaded from anywhere and can be modified by the user.

1. **Portable**

* It is a portable language, that is, the programs written in computer can easily transfer to another computer system.

1. **Embeddable**

* The other programming language code can use in the Python source code. We can use Python source code in another programming language as well.

1. **Extensible**

* It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our Python code.
* It converts the program into byte code, and any platform can use that byte code.

1. **Platform independent**

* The code written in one platform (OS) can be executed in another platform without any modifications.
* The Python programs are equally run on any operating system like Window, Linux and Macintosh etc.

1. **Dynamically Typed Language**

In Python we are not required to declare type for variables. Whenever we are assigning the value, based on value, type will be allocated automatically. Hence Python is considered as dynamically typed language**.**

Ex. x=10 only

We don't need to write **int x = 10**.

1. **Standard Libraries**

* It provides a wide range of libraries for the various fields such as machine learning, web developer, and also for the scripting.
* Eg. Machine learning libraries - Tensor flow, Pandas, Numpy, Keras, and Pytorch, etc.

Desktop development libraries – PyQT5, Tkinter, Kivy etc.

1. **Expressive Language**

Python can perform complex tasks using a few lines of code.

Eg. **To print hello world**

In ‘C’ void main()

{

printf(“hello world”);

}

In Java class sample

{

public static void main(String args[])

{

System.out.println(“hello world”);

}

}

In Python just type - print(“hello world”)

1. **GUI support**

* Graphical User Interface is used for the developing Desktop applications.

**Applications of Python**

Python is used in almost every technical areas like:

* Data Science
* Data Mining
* Machine Learning
* Artificial Intelligence
* Desktop Applications
* Console-based Applications
* Mobile Applications
* Software Development
* Web Applications
* Enterprise Applications
* 3D CAD Applications
* Computer Vision or Image Processing Applications.
* Speech Recognitions

Some popular frameworks and libraries of Python as follows:

* **Web development (Server-side) –** Django, Flask, Pyramid, CherryPy ,etc.
* **GUIs based applications -** Tkinter, PyGTK, PyQt, PyJs, etc.
* **Machine Learning -** TensorFlow, PyTorch, Matplotlib, Scipy, etc.
* **Mathematics -** Numpy, Pandas, etc.

**Companies using Python**

* Google
* Dropbox
* Netflix
* bitTorrent
* NASA

**Career opportunities**

* Web development - -
* Big data analysis
* Web app testing
* Data scientist - *machine learning engineer, AI*
* *Smart IoT devices - used for connecting the world*

**IDEs for Python**

**IDE – Integrated Development Environment**

1. **PyCharm**
   * Is a cross-platform IDE, developed by JetBrains company of Czech
   * Facilitates Web Development along with [Django](https://www.edureka.co/blog/django-tutorial), Flask, and web2py
2. **Spider** 
   * Spyder is an **open-source**, **cross-platform**IDE developed by Pierre Raybaut in 2009.
   * It is integrated with many of the scientific Python libraries namely Scipy, NumPy, Matplotlib, Pandas etc.
3. **PyDev**
   * is basically an open-source third-party package which serves as a plug-in for Eclipse to enable it for [Python development](https://www.edureka.co/blog/python-programming-language).
4. **Rodeo**
   * Rodeo is an **open source** Python IDE developed by Yhat. It is built in particular for [machine learning](https://www.edureka.co/blog/what-is-machine-learning/) and [data science](https://www.edureka.co/blog/learn-python-for-data-science/).
5. **Sublime Text**
   * Sublime-Text is a cross-platform IDE developed in C++ and Python. In addition to Python, it provides support for other languages as well
6. **Wing**
   * This IDE was created by Wingware. It is a lightweight IDE designed to allow quick [programming](https://www.edureka.co/blog/python-programming-language).
7. **EricPython**
   * Eric is written in Python and is free software. Its source code is available freely and can -be studied and recreated by anybody.
8. **Atom**
   * Atom is an **open source** free IDE built using web technologies. Atom is based on the Electron framework which is built by [GitHub](https://www.edureka.co/blog/how-to-use-github/) which in turn is written in CoffeeScript and Less.
9. **Thonny** 
   * Thonny is an IDE developed for beginners. It provides step-by-step assistance to the programmer.
10. **IDLE**
    * completely in [Python](https://www.edureka.co/blog/python-tutorial/) and it comes as a default implementation along with Python IDLE is written

**Character set**

A character set is a collection of characters that can form words, constants, expressions etc. Python supports an ASCII character system which contain 256 characters only. Python also supports a Universal character system called UNICODE system.

Eg. Alphabets - A to Z and a to z

Digits - 0 to 9

Symbols - + - \* / % @ $

# & ^ | \ : ?

= < > ‘ “ ; .

( ) [ ] { }

**Tokens** – small units used in a program are called ‘Tokens’. The tokens are classified into various types as follows:

1. Keywords
2. Identifiers
3. Constants or Literals
4. Operators
5. Keywords

Keywords are nothing but pre-defined words. The meaning/purpose of those words is already defined in Python. They are also known as reserved words. Pythons supports 33 keywords.

They are:

Boolean type - True False None(object)

Operators - and or not is in

Conditional - if else elif

Loops - while for

Jumping statements - break return continue yield

Exceptions - try assert except raise finally

General keywords - def as class from import global nonlocal lambda with pass del

**Note: To get keywords list, type:**

**import keyword**

**print(keyword.kwlist)**

1. **Identifiers**

An identifier is the name given to the program elements like variables, classes, exceptions, collections (List, Tuple, Set, Dictionary) etc.

Rules for an identifier

* + An identifier is a group of characters
  + Start with an alphabet or \_ (underscore)
  + Symbols are not allowed (except \_)
  + Keywords are not used as identifiers
  + Spaces and commas are not used
  + It is case sensitive ie. Upper case and lower case letters are not equal ( total is not equal to Total or TOTAL).

Eg. ***valid*** ***invalid***

m 1m

\_m m$

num1 else

tot\_sal tot sal

1. **Literals** – A literal is a constant, which is a fixed value that cannot be changed at the runtime of a program. The literals are classified into various types as follows:
2. String literals
3. Numeric literals
4. Boolean literals
5. Collection literals
6. Special literals
7. **String literals**

A group of characters is called string. The string literals are enclosed either in single or double quotes.

Ex. “WELCOME”, ‘computer’, ‘Python’

In Python, the string literals are represented in two types.

1. Single line strings

Ex. 1) str=”welcome”

2)s1=’welcome to bdps’ print(s1)

1. Multi line strings – Use triple quotation marks

Ex. s2=”””welcome

To

Python

Literals”””

print(s2)

1. **Numerical literals**

They are represented by the digits 0 to 9. They can represents any type of numerical value like integer, long, float, and complex, and also binary, decimal, octal and hexadecimal. They are immutable ie can’t be changed.

They are in 3 types – Integer , float and complex

1. ***Integer literals***:

An integer is a rounded or whole number ie without fractional part. They are in 4 types.

1. *Decimal(base10)* – all integers are decimal by default.

Eg:

a=100

print(a) - 100

1. *Binary(base 2)* – contain the binary digits o or 1 only and leading with 0B/0b

Eg. b=0b10011

print(b) - 19

1. *Octal(base 8)* – contain the digits o to 7 only and leading with 0o

Eg. c=0o25

print(c) - 21

1. *Hexadecimal(base16)* – contain the digits 0 to 9, the letters a-f/A-F and leading with 0x/0X.

Here, a/A – 10, b/B – 11, c/C – 12

d/D – 13 e/E – 14 f/F – 15

Eg. a=0x100

print(a) - 256

b=0XFF

print(b) - 255

c=0XA4

print© - 164

1. ***Floating point literals*** – it has an integer part as well as fractional part. They are represented in 2 ways
2. Fractional representation

Eg. f1=10.5

print(f1)

1. Exponential representation

Eg. f2=10.5e2

print(f2)

1. ***Complex Literals*** – contain real and imaginary part ie. a+bj

Ex. c= 3+5j here c is called an object

To get real part - object.real

To get imaginary - object.imag

Eg. c=3+5j

print(c)

print(c.real)

print(c.imag)

**Special Literals** - python has only one special literal ie. None

eg. v=None

print(v)

**Boolean literals** – represents a Boolean value called either True or False

True – 1 False – 0

Eg. x=True

print(x)

print(5+True) - 6

print(10+False) - 10

**Collection Literals**

Collection means group of items. In Python the collections are defined in 4 types. They are:

1. List
2. Tuple
3. Set
4. Dictionary
   * 1. List: If the items are enclosed in [ ] then it can be called as list object.

Eg. a=[1,2,3,4,5]

print(a) o/p- [1,2,3,4,5]

* + 1. Tuple:- If the items are enclosed in ( ) , is called as tuple object.

Eg. 1) tup=(2,3,4,5,6)

print(tup) o/p- (2,3,4,5,6)

2)tup1=(1,”abc”,2,”mango”)

print(tup1) o/p- (1, ‘abc’ , 2 , ‘xyz’)

* + 1. Set: If the items are enclosed in { } is called set object. It represent the elements without duplicate values.

Eg. s={1,2,3,4,5,5}

print(s) o/p- {1,2,3,4,5}

* + 1. Dictionary: It is also similar to set object ie items are enclosed in { } and

Each item is a pair of key and value, that is, **key:value**

Eg. d={‘a’:101,’b’:’kumar’, ‘c’:25500.00}

print(d)

o/p- {‘a’:101,’b’:’kumar’, ‘c’:25500.00}

**print() function:**

It is the output function in Python. It is used to print or display output on screen.

syntax: **print(object(s), sep=separator, end=end)**

where object - represents an object to be printed

sep - objects are to be separated by sep

end - object to be print at last

Ex. x=7

print(x)

print(‘X= ‘, x)

print(‘X= ‘,x,sep=’ 000 ’)

print(‘X= ‘,x, sep=’$$$$’ , end = ‘ \n\n\n’)

print(‘Thank U’)

o/p- 7

X = 7

X= 000007

X = $$$$7

Thank U

**type() function -** This function returns the type (datatype) of a given object

Syntax: type(object)

**Id() –** This function returns the address of an object

Syntax: id(object)

Note: If two objects referred the same value then the address of the objects are also same because the same memory will be referred by the both objects.

**Datatypes in Python**

A datatype specifies the type of data a variable can stored or hold, that is, the data present inside of a variable. In python, the datatype cannot be specified in a program explicitly. It can automatically assigned to a variable based on the value provided.

Python provides the following built-in datatypes.

* 1. int
  2. float
  3. complex
  4. bool
  5. str
  6. bytes
  7. bytearray
  8. range
  9. list
  10. tuple
  11. set
  12. frozenset
  13. dict
  14. None

The datatypes in Python are mainly classified into 2 types – Mutable and Immutable.

Mutable - means values to be changed

Immutable - means values not be changed

DATATYPES

Immutable Mutable

Numbers Strings Lists Dictionaries

Tuples Sets

**Numeric Datatypes**

These datatypes can represent numeric values like integer, float, complex, decimal, octal, hexadecimal etc.

They are classified into 3 types:

1. int datatypes
2. float datatypes
3. complex datatypes

type() – It returns the datatype of an object (variable)

syntax: type(object)

1. **int datatypes:**

This datatype can return/represent an integer value ie whole numbers.

An integer may be

-decimal (base 10)

- binary(base 2)

- octal(base 8)

- hexadecimal(base 16)

***Decimal –* contain the digits 0 to 9**

Ex. a=100

B=234

***Binary* – contain the digits 0 and 1, leading by 0b/0B**

Ex. x=0B11001

Y=0b10011

Print(x,y) -25, 19

***Octal* – contain the digits 0 to 7 only, leading by 0o/0O**

Ex. x=0o234

y=0O75

print(x)

print(y)

***hexadecimal*** – **contain the digits 0 to 9, the letters a-f/A-F, leading by 0x/0X**

ex.

x=0x100

y=0XFA

print(x,y) o/p – 256 250

Note: the above all decimal, binary , octal and hexadecimal are belongs to ‘int’ datatype.

eg. a=100

b=0b11001

c=0o234

d=0xff

print(type(a)) - <class ‘int’>

print(type(b)) - <class ‘int’>

print(type(c)) - <class ‘int’>

print(type(d)) - <class ‘int’>

1. **float datatypes** - these datatypes can return/represent a floating value (decimal value)

eg. i) floating value

f1=10.5

print(type(f1)) - <class ‘float’>

ii) floating value as exponential power

Eg. f2=5e3

print(f2) – 5000.0

print(type(f2)) - <class ‘float’>

1. ***Complex datatypes*** - It represents a complex value, 3+5j.
   * + It contain real part (3) and an imaginary part (5)
     + It allows both integer and floating values

Eg. x=4+5j

Y=2.5 + 5.6j

print(type(x)) - <class ‘complex’>

print(type(y)) - <class ‘complex’>

**bool datatype**

It represents a Boolean value either True or False

In python, True – is 1 and False – is 0

Eg. y=True

print(type(y)) - <class ‘bool’>

z= False

print(type(z)) - <class ‘bool’>

**str datatype:**

It represents a string value. The string values are enclosed in either single quotes or double quotes.

Eg. s=”welcome” or s=’welcome’

print(type(s)) - <class ‘str’>

**Type casting:**

It means to convert one datatype value into another type. Python support the following conversion functions.

1. int() – converts a value into int type

syn: int(value)

ex.

a=input("enter a no.1")

b=input("enter no.2")

print(a+b)

x=int(a)

y=int(b)

print(x+y)

1. float() – converts a given to a float type

syn: float(value)

1. complex() – converts a given to complex type

syn: complex(value)

1. bool() – converts a given value to bool type

syn: bool(value)

ex.

a=1

print(bool(a))

1. str() – converts a given a value to str type

syn: str(value)

ex.

a=10

b=20

print(a,b,a+b)

c=str(a)

d=str(b)

print(c,d,c+d)

**Conversion functions**

They are used to convert one integer format to another integer format ie. Binary to octal, hexadecimal or decimal. They are – bin() ,oct() and hex()

* + 1. bin() – It converts a given value to binary format

syntax: bin(value)

ex. a=25

print(bin(a))

b=0o25

print(bin(b))

c=0xff

print(bin(c))

* + 1. oct() – converts a given value to octal format

syntax: oct(value)

ex. a=35

b=0b111011

c=0xfa

print("Decimal : ",a,b,c)

print("Octal : ",oct(a),oct(b),oct(c))

* + 1. hex() – It converts a given value to hexa decimal format

Syntax: hex(value)

Ex. a=255

print(hex(a))

b=0b1001101

print("Binary : 0b1001101")

print('Decimal : ',b)

print("Hexa : ",hex(b))

c=0o234

print("Decimal : ",c)

print("Hexa decimal : ",hex(c))

**range() datatype**

This datatype is used to represent a range of numeric values ie from start to end-1. It can be applied in a loop or collection types.

Syntax: range(start,end[,step])

If start is omitted, then range starts from o.

Ex. print("---Using range(10)----")

a= range(10)

for i in a:

print(i,end=" ")

print("\n---Using range(1,11)----")

b=range(1,11)

for i in b:

print(i,end= " ")

print("\n---Using range(1,11,2)----")

c=range(1,11,2)

for i in c:

print(i,end=" ")

**list datatype:**

This datatype can return or represent group of items. It a mutable datatype (object). In a list, the items are enclosed in [].

Syntax: obj =[val1, val2, …]

Ex.

a=[2,5,7,9,1]

print(a)

print(type(a))

b=['anil', 'kumar' , "mahesh"]

print(b)

print(type(b))

***Note: in python every variable can be treated as an object.***

**Tuple datatype:**

This datatype can also represent a group of items. In this, the items are enclosed in (). But it is an immutable datatype (object) ie values cannot be changed.

Syntax: object = (val1, val2, …)

Ex.

tup=(2,5,7,9,1)

print(tup)

print(type(tup))

a=('anil', 'kumar' , "mahesh")

print(a)

print(type(a))

**set datatype:**

It is also be used to represent a group of items (elements) without duplicates. In this, the items are enclosed in { }.

Syntax: object = {val1, val2, … }

Ex.

Note:

If an empty set is created then by default it can be treated as dictionary object ie dict.

**Dict datatype**

It is also similar to set datatype. But in this, the items are represented as ‘key:value’ pairs and enclosed in {}.

Syntax: object = { key:value1 , key:value2, …. }

Ex. a={1:100,2:200,3:300}

print(a)

print(type(a))

b={'a':'apple' , 'b':'banana','c':'mango'}

print(b)

print(type(b))

**None datatype**

This datatype does not return or represent any type of value.

Syntax: object = None

Ex. a=None

print(a)

print(type(a))

**bytes datatype –** It is used to represent a sequence of byte values from 0-255. It creates an immutable object ie the values can’t be changed.

Ex. a=[65,66,67,68]

x=bytes(a)

print(x)

print(type(x))

#x.add(90) – can’t be allowed (immutable)

#print(x)

a=[65,66,67,68]

x=bytearray(a)

print(x)

print(type(x))

x.add(90)

print(x)

**bytearray datatype** – It is also similar to bytes datatype, sequence of byte values from 0-255. But it creates a mutable object.

Ex.

a=[65,66,67,68,69]

b=bytesarray(a)

print(b)

print(type(b))

**frozenset datatype –** It is also similar to set datatype. But it is immutable object, whereas set is mutable object.

Syntax: obj=frozenset(set)

Ex.

s={10,20,30,40}

print("---Using set----(Mutable)")

for i in s:

print(i)

s.add(25)

print(s)

print("---Using Forzenset----(Immutable)")

s={11,22,33,44,55}

fs=frozenset(s)

for i in fs:

print(i)

fs.add(25)

**Input() –** This function is used to take input from the user ie to access user input. It returns a string value.

Syntax: input(“message “)

To read 2 numbers

a,b=[int(x) for x in input("enter 2 no.s").split()]

print(a,b)

to read 3 float values

x,y,z=[float(x) for x in input("Enter 2 float values : ").split(",")]

print("x =",x)

print("Y= ",y)

print("Z= ",z)

**Operators in Python**

An operator is a symbol which can perform either an arithmetical or logical operations on data (operands/variables). Python support the following operators.

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Bitwise operators
6. Special Operators
7. Identity operators
8. Membership operators

1. **Arithmetic operators -** They are used to perform the arithmetical operations like addition, subtraction etc.

Eg. +, -, \* , / , %

\*\* - exponential power

// - floor value of division

1. a=10

b= 20

c= a+b

d=a\*b

print(a,b,c,d) - 10 20 30 200

1. x=2\*\*5

print(x) - 32

print(10/3) - 3.3333333

print(10//3) - 3

1. **Relational operators -** They are used to compare any two values, also known as comparison operators. They returns either true or false when comparing.

Eg. > ,<, >= , <=

Equality operators == , !=

1. a=10>5

print(a) - True

1. b= 4<2

print(b) - False

1. **Logical operators** -

They areused to compare more than one conditions at a time. They also returns either true or false when comparing.

They are:

And, or , not

And : it returns true when all conditions are true, otherwise false

Or : it returns false when all conditions are false, otherwise true

Not : it evaluates the negation of the condition ie if a condition true then it evaluates to false.

Eg.

a,b,c=10,5,9

print(a>b and b>c) - False

print(a>c or b>c) - True

print(a>b) - True

print(not a>b) - False

1. **Assignment operators** – they are used to assign or store a value into operand. It is used to give initial value to a variable. This is called initialization.

Eg. a=10

print(a) - 10

a,b=10,20

print(a,b)

a,b,c=10,"kumar","clerk"

print(a,b,c)

The assignment operators are also applied as follows:

1. += ex. a+=b 🡪 a=a+b
2. -= a-=b 🡪 a=a-b
3. \*= a\*=b 🡪 a=a\*b
4. /= a/=b 🡪 a=a/b
5. %= a%=b 🡪 a=a%b

Ex. a=10

b=6

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

#a+=b

#a\*=b

#a/=b

a//=b

print(a)

1. **Bitwise operators** – These operators can perform the operations on bits format of the operands internally.

They are:

Bitwise AND - &

Bitwise OR - |

Exclusive OR - ^

Complement - ~

Shift Left - <<

Shift Right - >>

Eg.

a=10

b=12

print('a = ',a,' b= ',b)

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

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

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

print('a<<1 = ',a<<1)

print('b>>1 = ',b>>1)

print('~b = ',~b)

**Special Operators**

**Python** also provide the following two special operators. They are:

1. Identity operators
2. Membership operators
3. **Identity operators** -  They are used to decide a value is similar to an object

They are : is

is not

Eg. a=b=10

print(a is b)

print(a is not b)

1. **Membership operators** - It is used to test whether a value is the member of a sequence or not.

They are : in

not in

Eg. 1) lst=[2, 5, 8, 13, 15]

print( 5 in lst) - True

print( 5 not in lst) - False

2)fruit=[“apple”,”banana”,”mango”,”orange”]

print(“mango” in fruit) - True

print(“mango” not in fruit) - False

Using control strings– the control strings are used in ‘c’ language. They can represent the data in a specified format.

%d – integer

%f – float

%s – string

Eg. a=10

b=12.5

c=”welcome”

print(“Integer: %d\nFloat : %f\nString :%s”%(a,b,c))

**math module**

**variables:**

**math module** can provide the following two variables:

1. **Pi** – it returns the pi value
2. **E**  - it returns the exponential value

**functions in math module:**

1. ***sqrt(n)*** – returns the square root of a number
2. ***pow(m,n)*** – returns the power value of m raised to n power
3. ***factorial(n)*** – returns the factorial of a number
4. ***trunc(n)*** – returns the integer value in a float value
5. ***floor(n)*** – returns the largest(nearest) integer <= a number
6. ***ceil(n)*** – returns the smallest(nearest) integer >= a number
7. ***sin(n***) – returns the sin value of a given no.of degrees
8. ***cos(n)*** – returns the cosine value of a given no.of degrees
9. ***radians(n***) – returns the radian value of a given no.of degrees
10. ***tan(n)***
11. ***cot(n)***

To work with the variables and functions of a module then that module can used in a python program. For this, import statement is used. It is applied in 2 ways:

**Syntax:1: import modulename**

Ex. import math

**Syntax:2: from modulename import variable or function(s)**

Ex. from math import pi, sqrt

To invoke a variable

Syntax: modulename.variable

Ex. math.pi

math.e

To invoke a function

Syntax: module.function(args)

Ex. math.sqrt(25)

Note: for syntax 2, module name will not be used.

***To define an alias name for a module***

Syn: import module as alias

Ex. 1) import math as m

print("Pi = " ,m.pi)

print('Factoria of 5 : ',m.factorial(5))

1. **import math**

print("E = ",math.e)

print("Sqrt(25) : ",math.sqrt(25))

print("pow(3,5) : ",math.pow(3,5))

print('Factorial(5) : ',math.factorial(5))

print("trunc(155/10) : ",math.trunc(155/10))

print("floor(23.45) : ",math.floor(23.45))

print("ceil(23.45) : ",math.ceil(23.45))

print("sin(45) : ",math.sin(45))

print("cos(45): ",math.cos(45))

print("Radians(180): ",math.radians(180))

ex.2) **from math import pi,sqrt**

print(pi)

n=int(input("enter a no : "))

print('square root is : ',sqrt(n))

**eval()**

This function is used to evaluate a string representation of an expression directly.

syn: eval(string)

ex. n=eval("10+20+30+50")

print(n)

x=eval("10\*\*3")

print(x)

a=input("enter an expression : ")

print(“Result : “,eval(a))

o/p - enter an expression : 4\*5+12/2

Result = 26.0

**del statement** – It is used to delete an object (variable)

syn: **del variable**

**Flow control statements:**

They are used to control the execution of a program. They are in 3 types.

* 1. Conditional statements
  2. Transfer statements
  3. Iteration/Loop statements
     + 1. Conditional statements – They can execute one more statements based on a condition. They are in 3 types.

1. If statement
2. If..else statement
3. If .. elif ... else statement
4. if statement:

It can execute a statement or set of statements when the condition is true only, otherwise they are skipped.

syn: if condition:

statement(s)

Note: In python, the { and } are not used. Here, a block of code can be represented by the indentation.

ex. a=int(input("enter no.1 : "))

b=int(input("enter no.2 : "))

max=a

if max<b:

max=b

print('Max : ',max)

1. if..else statement – It can execute only one block of statements at a time from 2 alternative blocks. If the condition is true then one block will be executed, otherwise another block will be executed.

syn: if condition:

statement(s)

else:

statement(s)

To accept present and prev reading. Find out no.of units and bill using

No.of units = present – prev.reading

* + 1. If no.of units >=100 then unitprice=3.50
       1. <100 = 2.50

Bill = no.of units\*unit price

***TO accept experience and salary of an employee. Find out hra, da, ta and total salary*** using:

hra da ta

1. If exp>=10 then 6 5 4%
2. <10 5 4 3% of salary

Total salary = salary +hra+da+ta

If .. elif.. statement – It is used to evaluate more than one conditions one after another. If any one condition is satisfied then that corresponding block will be executed and the remaining conditions will not be checked.

syn:

if cond1:

statements

elif cond2:

statements

elif cond3:

statements

----

---

elif cond-n:

Statements

Ex.

a=int(input("Enter maths marks : "))

b=int(input("Enter Physics marks : "))

c=int(input("Enter Chem marks : "))

tot=a+b+c

avg=tot//3

if a<40 or b<40 or c<40:

res="Fail"

elif avg>=80:

res="Grade-A"

elif avg>=70:

res="Grade-B"

elif avg>=60:

res="Grade-C"

elif avg>=50:

res="Grade-D"

else:

res="Grade-E"

print("Total : ",tot)

print("average : ",avg)

print("Result : ",res)

TO accept experience and salary of an employee. Find out bonus and total salary using:

1. If exp>=10 then bonus= 6%
2. If exp>=8 then =5%
3. If exp>=5 =4%
4. <5 =3% of salary

Total salary = salary +bonus

To accept present and prev reading. Find out no.of units and bill using

No.of units = present – prev.reading

* + 1. If no.of units >=400 then unitprice =5.50
    2. If no.of units >=300 then =4.25
    3. If no.of units >=200 =3.75
    4. If no.of units>=100 =2.80
    5. <100 = 1.50

Bill = no.of units\*unit price

***Nested if***:

It means one if statement may contain another if statement. Here, the if may be simple if, if .. else or elif …. It is applied when one condition is depending on another condition.

Syn:

If cond-1:

If cond-2:

Statements-1

else:

statements-2

else:

if cond-3:

statements-3

else:

statements-4

Biggest of 3 nos

a=int(input("Enter no.1 : "))

b=int(input("Enter no.2 : "))

c=int(input("Enter no.3 : "))

'''if a>b:

if a>c:

max=a

else:

max=c

else:

if b>c:

max=b

else:

max=c

'''

if a>b and a>c:

max=a

elif b>c:

max=b

else:

max=c

print("Big : ",max)

To accept gender, experience and salary. Find out bonus- using

1. If gender=male then
   * + 1. If exp>=10 then bonus=5%
       2. If exp>=5 =4%
       3. <5 =3%
2. If gender=female
   1. If exp>=8 then bonus = 4%
   2. <8 =3%of salary

**Iteration/Looping statements**

If a statement or set of statements are executed repeatedly is called a loop. The statements used to control a loop are called looping statements. In python, the looping statements are in two types.

1. While
2. for
3. ***while loop***

It is also known as a pre-test loop or entry-controlled loop. In this, the condition is evaluated at first, if it is true then only the body of loop will be executed.

Syntax: while condition:

-----

Statements

1. ***for loop:***

In Python, for loop is used to repeat the iterable objects such as lists, tuples, or strings etc. It is useful to repeat a section of code a certain number of times.

Syntax:

for variable in sequence:

Body of loop

Ex.

**Nested Loops**:

It means one loop statement may contain another loop statement within the body. It is mainly used for executing the repeated statements repeatedly. It contain at least 2 loops – an inner loop and an outer loop.

Syntax: outer loop

While condition:

Statements inner loop

For I in sequence:

Statements

Statements

**Transfer statements**

In python to transfer or move the program control from one place to another place at runtime transfer statements are used. They are:

* + - * 1. break statement
        2. continue statement
        3. pass statement

**break statement**:

It is used to terminate a loop without reaching an end of a loop. It performs an immediate exit from a loop. It is applied with the loop statements only.

Syntax: if condtion:

break

***Continue statement***

This statements continues the loop without executing the remaining statements in the body of loop. It is mainly used to suppress the execution of statements for some time.

Syntax: continue

**Pass statement**

It is used to represent an empty body for a function or if block.

Syntax: pass

Command Line Arguments

The command prompt selected from DOS is called command line. The arguments that are applied at the command prompt are called command line arguments. To represent these arguments python provides two properties called – argc and argv[].

argc – represent the no.of arguments applied at command line

argv[] – represent the arguments applied at command line

They are provided in **sys** module.

Note: argv[0] – represents the filename

**STRING DATA STRUCTURE**

A string is a collection of characters. In python, the strings are enclosed in either single quotes or double quotes.

Ex.

a="welcome"

b='bdps limited'

print(a)

print(b)

print(type(a))

print(type(b))

***String operators***

These operators can perform the operations on string data only.

They are:

* + - * 1. + ( concatenation) operator
        2. \* Repetition operator
        3. [:] slice operator

1. + (concatenation) operator – It is used to add one string to another.

Ex.

a=input("Enter string1 : ")

b=input("Enter String2 : ")

print(a+b)

1. \* (Repetition) operator – It is used to print a string no.of times repeatedly.

Ex. print(n\*”bdps”)

1. Slice operator [ : ] – It is used to retrieve portion of text from a string.

Syntax: string[start : end :step]

Ex.

s="welcome"

print(s[0:])

print(s[2:])

print(s[3:6])

print(s[1:6:2])

print(s[0:7:3])

print(s[0::3])

#to print last character

print(s[-1::])

#to print reverse

print(s[::-1])

#to print character from -5 to -1(end-1)

print(s[-5:-1])

**Member ship operators** – It is used to find a character is the member of a string or not. They are :

* 1. In operator
  2. Not in operator

Syntax: character in/not in string

s=input("Enter a string : ")

a=input("enter a substring : ")

if a in s:

print("Found")

else:

print("Not found")

***To display the characters in string***

Index value is used to retrieve the characters in a string.

Syn: s[index]

Ex. s="welcome"

print(s[0])

print(s[2])

print(s[-1])

To print the characters in string

Ex.

1. s="welcome"

a=len(s)

for i in range(a):

print(s[i])

1. s="welcome"

a=len(s)

for i in range(a):

print(s[0:i+1])

***String functions***

**len()** – It is used to find the length of a string

ex. s=”welcome”

print(len(s))

**Changing case of a string**

The following functions are used to change the case of characters in a string.

***upper()*** - used to convert a string into upper case

ex. s=input("Enter a string : ")

print(s.upper())

***lower()*** - used to convert a string into lower case

**ex.** print(s.lower())

***swapcase()*** - change the case of a character ie upper to lower and lower to upper

***title()*** - converts each word initial as capital

***capitalize()***  - converts a string initial only capital

Ex.

s=input("Enter a string : ")

print("Upper : ",s.upper())

print("Lower : ",s.lower())

print("Title case : ",s.title())

print("Capitalize : ",s.capitalize())

print("Swap case : ",s.swapcase())

**find() -** returns the index of a sub string in a string

**syn: find(sub,start,end)**

ex.

s=input("Enter a string : ")

sub=input("Enter a sub string : ")

n=s.find(sub)

print("Position : ",n)

print("Position after 12 : ",s.find(sub,12))

#to print all positions

pos=-1

flag=False

while True:

pos=s.find(sub,pos+1)

if pos==-1:

break

print("position : ",pos)

flag=True

if flag==False:

print("Not found")

***index()*** It returns the index value of sub string in a string. If not returns -1

Syn: index(substring, startindex, endindex)

s=input("Enter a string : ")

sub=input("Enter sub string : ")

a=s.index(sub)

print(sub, " is at : ",a)

a=s.index(sub,5)

print(sub, " after index 5 : ",a)

***count(sub,start,end) –***

It returns an integer representing the no.of occurences of a sub string within a string.

EX. s="Welcome To PYTHON"

print("String is : ",s)

sub=input("Enter a sub string : ")

c=s.count(sub)

print("No.of occurrences : ",c)

**replace()** – used to replace an old string with new string

syn: ***var=replace(oldstr,newstr)***

Ex.

s='welcome to python'

print(s)

r=s.replace('e','x')

print(r)

r=s.replace('o','bdps')

print(r)

**To check type of characters**

isalnum() – finds whether a string has alpha numeric characters or not

It returns true if the characters are alphabets or digits, otherwise returns false

isalpha() – finds whether a string has alphabets or not

isdigit() – finds whether a string has digits or not

isspace() - finds whether a string has spaces or not

istitle() - finds whether a string is in title case or not

isupper() - finds whether a string is in upper case or not

islower() - finds whether a string is in lower case or not

s=input("enter a string: ")

if s.isalnum():

print("--- is an alphanumreric")

if s.isalpha():

print("--- is an aphabet")

if s.islower():

print("--- is in lower case")

else:

print("--- is in upper case")

else:

print("--- is digit")

else:

print("--- is symbol")

***center(width[,fillchar])*** – It is used to align center by filling spaces or char with a string

ex.

s=input("Enter a string : ")

print(s)

print(s.center(15),"End")

print(s.center(20,'$'))

checking starting and ending part of the string

**startswith(substring) -**  finds whether a string starts with a substring or not

***endswith(substring)* –** finds whether a sting ends with a substring or not

ex. s=input("Enter a string : ")

s1=input("Enter a sub string : ")

print(s," starts with ",s1," : ",end=" ")

if s.startswith(s1):

print("yes")

else:

print( "No")

(or)

x=s.endswith(s1)

print(s,"ends with ",s1," : ",x)

**Splitting of Strings**

split() – It is used to split a sentence into words based on a substring. They are stored in a list object.

Syntax: ***object=s.split([char])***

Ex. s="welcome to bdps limited"

l=s.split()

print(l)

for i in l:

print(i)

**join()-**  It is used to join words in a sequence with a given separator

syntax: ***string=separator.join(group of strings)***

ex. tup=("welcome","bdps",”software” ,”training”, “Institute”)

s=" ".join(tup)

print(s)

**Removing spaces from a string**

***lstrip([char])*** - removes leading spaces or char

***rstrip([char])*** - removes trailing spaces or char

***strip([char])*** - removes both leading and trailing spaces or char

ex.

s1="Hello!"

s2=" welcome "

s3=" To BDPS"

print(s1+s2+s3)

print(s1+s2.lstrip()+s3)

print(s1+s2.rstrip()+s3)

print(s1+s2.strip()+s3)

s2="$@$$@@welcome@@$$$"

print(s2)

print(s2.strip('@$'))

***format()*** – It is used to set a specified format for the string to be displayed on screen. For this **replace operator {}** is used in print function.

Syntax: print(“{ } “.format(value))

s="{}, {} and {} are best friends".format('sachin','dhoni','kohli')

print(s)

s1="{0}, {1} and {1} are best friends".format('sachin','dhoni','kohli')

print(s1)

s2="{2}, {1} and {0} are best friends".format('sachin','dhoni','kohli')

print(s2)

s2="{a}, {b} and {c} are best friends".format(a='sachin',b='dhoni',c='kohli')

print(s2)

print("---Employee details----")

s3="Number: {}\nName : {} \nSalary : {} ".format(101,'dhoni',23500.00)

print(s3)

**To find the reverse of a string**

**method 1**:

s="welcome to bdps"

l="".join(reversed(s))

print(l)

**method(2)**

print(s[::-1])

**method(3):**

n=len(s)

i=n-1

res=""

while i>=0:

res+=s[i]

i=i-1

print(res)

Reverse order of words

s=”welcome to bdps limited”

l=s.split()

l1=[]

i=len(l)-1

while i>=0:

l1.append(l[i])

i=i-1

r=" ".join(l1)

print(r)

**LIST DATA STRUCTURE**

This data structure is used to store a group of items either similar or different types. In a list, the items are enclosed in [].

It is a mutable object ie the items in a list can be changed.

Syntax: object = [val1, val2, val3, …]

Ex. x=[1,2,3,4,5]

print(x)

(or)

for I in x:

print(i)

print(type(x))

To create an empty list

x=[]

1. l1=[‘mango’

Functions:

1. len() – It returns an integer representing the length of a list ie total no.of items.

syn: len(list)

ex. lst=[1,2,3,4,5,6,7,8,9,10]

n=len(lst)

print("List elements : ",lst)

print("Lenth = ",n)

print("---List items using index : ")

for i in range(len(a)):

print(a[i])

#To store user given items

l3=eval(input("enter list elements : "))

print(l3)

1. ***count()*** – it returns the number of occurrences of a given item in a list

syn: count(item)

ex. lst=[1,2,2,2,3,3,4,4,4,4,5,5,5]

print(lst.count(1)) 🡪 1

print(lst.count(2)) 🡪 3

1. ***index()*** - **returns the index of first occurrence of the specified item.**

Syntax: index(item,pos)

If pos is specified then it returns the index of item after given position

Ex.

s="This data structure is used to store a group of items."

print(s.index('is')) - 2

print(s.index('is',10)) - 20

print(s.index('to')) - 28

1. ***append(***) – It is used to add/append an item to a list.

Syntax: append(item)

e=[]

e.append(100)

e.append("kumar")

e.append("Manager")

e.append(25000)

print(e)

e.append(10)

e.append("Male")

print(e)

1. ***insert()*** – It is used to insert an item at a specified index

syntax: ***insert(index,item)***

ex. e=[100,"kumar","manager",25000]

print(e)

e.insert(2,"Male")

print(e)

1. ***remove()-*** It is used to remove an item from a list.

***Syn: remove(item)***

1. ***pop()*** – It removes the top item in a list (last).

***Syn***: pop()

Ex. e=[100,"kumar","manager",25000]

print(e)

e.pop()

print(e)

1. ***reverse()*** – It is used to reverse the list items.

***Syn***: reverse()

e=[100,"kumar","manager",25000]

print(e)

e.reverse()

print(e)

1. ***sort(reverse=true)***  - It is used to sort the elements in a list either in ascending or descending order.

***Syn: sort(reverse=true)***

*a=[3,5,1,2,8,4,9,6]*

*print("---List----")*

*print(a)*

*a.sort()*

*print("---After sorting....")*

*print(a)*

*print("---Descending order---")*

*a.sort(reverse=True)*

*print(a)*

1. ***copy()*** – It is used to copy one list to another.

Syn: list2 =list1.copy()

a=[3,5,1,2,8,4,9,6]

print("---List----")

print(a)

b=[]

b=a.copy()

print("After copy---")

print(b)

1. ***clear()*** – it Clears all items in a list.

Syn: clear()

a=[3,5,1,2,8,4,9,6]

print("---List----")

print(a)

a.clear()

print(a)

**Aliasing and Cloning of List objects:**

**Aliasing**

The process of giving another reference variable to an existing object is called aliasing.

Ex. x=[10,20,30,40]

y=x

print(id(x))

print(id(y)) - both objects refer the values at same address

note: If we change the contents in one object then the other object values are also affected because both objects can refer the values at same address.

X[2]=200

print(x)

print(y)

To overcome this problem cloning is used.

**Cloning:-**

It means to create a duplicate object for an existing object. The cloning is done in 2 ways – a) using slice operator

b)using copy() function.

Using slice operator

x=[10,20,30,40]

y=x[:]

y[2]=222

print(x)

print(y)

Using copy() function

x=[10,20,30,40]

y=x.copy()

y[2]=222

print(x)

print(y)

**Using mathematical operators**

Using + operator- to concatenate 2 list objects ie add one list to another.

x=[10,20,30,40]

y=[100,200,300]

print(x+y)

Using \* operator – used to repeat a list object.

x=[10,20,30,40]

y=x\*3

print(y)

**Membership operators**

In and Not in operator

x=[10,20,30,40]

print(20 in x)

print(30 not in x)

a=[3,4,5]

b=[5,6,7]

x=[]

print(a)

print(b)

for i in range(len(a)):

x.append((a[i]+b[i]))

print(x)

**Nested Lists:**

It means one list may contain another list or multiple lists.

Ex. s=[10,20,[30,40,50]]

print(s[0])

print(s[1])

print(s[2])

print(s[2][0])

note: The nested list elements are also accessed as matrix elements.

Ex.

s=[[1,2,3],[3,4,5],[5,6,7]]

print(s)

print("Row wise elements : ")

for i in range(len(s)):

print(s[i])

print("---Matrix format : ")

for i in range(len(s)):

for j in range(len(s[i])):

print(s[i][j],end=" ")

print()

2nd method:

s=[[10,20,30,40],[2,3,4,5],[11,12,13,14]]

for i in s:

for j in i:

print(j,end=" ")

print()

**List Comprehensions**

It is very easy and compact way of creating list objects from any iterable objects (like list, tuple, dictionary, range etc) based on some condition**.**

Syn: list = [expression for item in list if condition]

Ex. a=[x for x in range(1,11)]

print(a)

b=[x\*x for x in range(1,6)]

print(b)

c=[x\*\*2 for x in range(1,6) if x%2==0]

print(c)

TO print first letter in each word

s=[“welcome”,”Python”, “Ramesh”, “Akash”]

list = [w[0] for w in s]

print(list)

to count the no.of letters in each word

s = "translator is a program".split()

print(s)

s=[[w.upper(),len(w)] for w in s]

print(s)

**TUPLE DATA STRUCTURE**

1. Tuple is exactly same as List. But it is immutable. Ie changes cannot be allowed.
2. It is used when we need not change the values in a group.
3. Insertion order is preserved.
4. Duplicates are allowed.
5. Heterogeneous objects are allowed.
6. In this the elements are enclosed within Parenthesis

**Creation of a Tuple**

Ex.

t1=10,20,30

print(t1)

t2=(3,4,5,6)

print(t2)

t3=10,"kumar","clerk",7000

print(t3)

t4=(10,)

print(t4,type(t4))

t5=(10) # represent as int

print(t5,type(t5))

**Functions of Tuple object**

1. Len() – returns the length of a tuple object

Ex. t1=(10,20,50,30,40)

print("Tuple elements : ",t1)

print("Length : ",len(t1))

1. Index() – returns the index of an item

Ex. print(“Index of 30 : “,t1.index(30)) - 3

1. Count() – returns the no.of occurrences of an item

s=(10,20,50,30,20,50)

print(“No.of occurences of 20 : “,s.count(20)) - 2

1. Sorted() – returns the sorted elements into a tuple object

Ex.

t1=(10,20,50,30,40)

print("Tuple elements : ",t1)

t2=sorted(t1)

print(“Sorted elements : “,t2)

(or)

print("Sorted : ",sorted(t1))

1. Min() – returns the min. element
2. Max() – returns the max element

Ex.

print("Max : ",max(t1))

print("Min : ",min(t1))

***Tuple Packing and Unpacking***

**Packing -**  creating a tuple using group of variables is called packing.

Ex

print("---Packing----")

a=10

b=30

c=50

d=20

t=a,b,c,d

print(t)

print()

t1=(a,d,b,c)

print(t1)

print()

a,b,c,d=101,"kumar","clerk",6000

t=(a,b,c,d)

print(t)

**Unpacking –** the reverse process of packing ie store the tuple elements in variables

print("\n---Unpacking----")

a,b,c,d=t

print(a,b,c,d)

t=(101,"kumar","clerk",6000)

(a,b,c,d)=t

print(a,b,c,d)

***Storing elements through user input***

x=eval(input("Enter tuple elements : "))

print(x)

#to find sum and average

s=0

for i in x:

s+=i

print("Sum : ",s)

print("average : ",s/len(x))

**SET DATA STRUCTURE**

* It is used to represent a group of unique values as a single entity.
* Duplicates are not allowed.
* Indexing and slicing not allowed for the set.
* Heterogeneous elements are allowed.
* Set objects are mutable i.e once we creates set object we can perform any changes in that object.
* The set elements are enclosed in curly braces.
* We can also apply mathematical operations like union, intersection, difference etc on set objects.

***Creation of a set object***

s={1,2,3,4,5}

print(s)

print(type(s))

s1={1,2,3,4,5,3,5}

print(s1)

s2={10}

print(s2)

print(type(s2))

t={}

print(type(t))

# an empty set represents dictionary object

using ***set()*** – It is used to create a set object with another sequence ie list, tuple, range etc.

Syntax: object = set(sequence)

Ex.

l=[2,3,4,5]

s1=set(l)

print(s1)

s2=set(range(1,11))

print(s2)

s3=set(range(1,11,2))

print(s3)

***user input:***

s4=eval(input("Enter set elements : "))

print(s4)

***functions of set object***

* + - * 1. Add(x) – used to add an object

s= {10,20,30,40,50}

print(s)

s.add(100)

print(s)

* + - * 1. Update(x,y,z) – used to add a sequence of items to set object like list, range, tuple… (iterable objects).

Ex.

a=[11,12,13,14,15]

b=(22,33,44)

s.update(a,b)

print(s)

s=set()

a=[11,12,13,14,15]

s.update(range(1,9),a)

print(s)

* + - * 1. copy() – it returns the copy of an object. It is used to create a duplicate object ie clone object

Syn: Object=obj.copy()

s= {10,50,30,40,20}

s1=s.copy()

print(s)

print(s1)

* + - * 1. pop() – removes and returns the random element

print(s.pop())

print(s.pop())

print(s)

* + - * 1. remove(x) – removed the element x, if not then we will get keyError.

s.remove(20)

print(s)

s.remove(55)

* + - * 1. discard(x) – similar to remove, but we won’t get keyError

s={1,2,3,4,5}

print(s)

s.discard(4)

print(s)

s.discard(10)

* + - * 1. clear() – clears all items in the object
        2. del - used to delete an object

***Mathematical operations on sets***

The operators applied on sets in maths are also applicable on set objects.

They are: Union

intersection

difference

***union()*** - It combine the elements in both set objects without duplicates.

***Intersection()***  - It returns the common elements in both objects

***difference()*** – it returns the elements in first object and not in second object

s1= {10,50,30,40,20}

s2= {80,50,90,45,20}

print(s1)

print(s2)

print(s1.difference(s2))

print(s2.difference(s1))

print(s1.union(s2))

***membership operators***

in and not in

ex.

s=eval(input("enter set elements: "))

sub=input("Enter sub element : ")

if sub in s:

print("found")

else:

print("Not found")

**Dictionary Data Structure**

It is also similar to set Data Structure.

The elements are represented as key:value pair and enclosed in { }.

Syntax:

Object = {key1:value1, key2:value2, …}

***Creation of a dictionary:***

Ex

1. d={} – creates an empty dictionary

For this the elements are added as follows:

d[10]="Harish"

d[20]="Mahesh"

d[30]="Ramesh"

print(d)

print()

1. d1={1:"Mango",2:"Orange",3:"Banana"}

print(d1)

1. d2={

**using dict()**

d=dict()

print(d)

d[1]=100

d[2]=200

d[3]=300

print(d)

d1=dict({'a':101 , 'b':"kumar" , 'c':25000})

print(d1)

#elements as tuples

d2=dict({(1,100),(2,200),(3,300)})

print(d2)

**To accept data from user**

n=int(input("Enter how many employees : "))

d=dict()

i=1

while i<=n:

print("\nEnter employee ",i," details : ")

num=int(input("Enter number : "))

name=input("Enter name : ")

d[num]=name

i=i+1

print("\nEmployees details : ")

for x in d:

print(x,"\t",d[x])

***To update a dictionary:***

To change the value at a given key

Syntax: d[key]=value

If key does not exist then new key will be added.

Ex. d={1:100,2:200,3:300}

print(d)

d[2]=222

print(d)

d[4]=400

print(d)

***To delete a value at a key:***

Syntax: del d[key]

Ex. d={1:100,2:200,3:300,4:400}

print(d)

del d[3]

print(d)

del d[5]

***To delete a dictionary***

Syntax: del object

del d

print(d)

***Functions of dictionary***

d=dict({1:”anil”,2:”Mahesh”,3:”ravi”})

d2={(10,”mango”),(20,”Banana”),(30,”orange”)}

print(d2)

1. len() – returns the no.of items in a dictionary

Ex. d={1:"anil",2:"Mahesh",3:"ravi"}

print(d)

print("No.of items : ",len(d))

1. get(key) – returns the value associated with the key

Ex. print("value at 3 : ",d.get(3))

If key not found then it returns **None**

1. pop(key) – removes the entry associated with key and returns the value

Ex. d={1:"anil",2:"Mahesh",3:"ravi"}

print(d)

print("Pop(2) : ",d.pop(2))

print(d)

1. popitem() - removes both key:value of last item and returns it

ex. d={1:"anil",2:"Mahesh",3:"ravi"}

print(d)

print("Popitem : ",d.popitem())

print(d)

1. ***keys()*** – returns all keys associated in a dictionary
2. ***values()*** – returns all values in a dictionary
3. ***items()*** – returns all items in a dictionary

Ex. d={1:"anil",2:"Mahesh",3:"ravi"}

print(d)

print(d.keys())

print(d.values())

print(d.items())

1. ***copy() –*** used to copy one object values to another object.

Ex. d={1:"anil",2:"Mahesh",3:"ravi"}

print(d)

d1=d.copy()

print(d1)

1. ***setdefault()*** – used to set default value to a key

syntax: obj.setdefault(key,value)

if key not found then new key will be added. If found not changed.

Ex. for i in d:

print(i,d[i])

print("---Existing key----")

d.setdefault(2,"sachin")

for i in d:

print(i,d[i])

print("--- key not exist----")

d.setdefault(5,"sachin")

for i in d:

print(i,d[i])

1. ***update(x) –*** to update one object with another

d={1:"anil",2:"Mahesh",3:"ravi"}

d1={'a':100,'b':200,'c':300}

print("---d elements---")

for i in d:

print(i,d[i])

print("\n--d1 elements---")

for i in d1:

print(i,d1[i])

d.update(d1)

print("\n---after update----")

for i in d:

print(i,d[i])

1. ***clear()*** – clears all elements in an object

ex. d.clear()

**FUNCTIONS**

A function is a self-contained program segment, which can perform a specific task. Functions are used to divide a program into smaller parts called ‘modules’.

A function is developed once and can be executed many times ie re-usability of code.

***Types of functions***

The functions are of two types.

* + - 1. Pre-defined or library functions
      2. User-defined functions

1. ***Pre-defined Functions***

The functions that are already developed in python are called ‘pre-defined functions’.

Ex. type()

id()

input()

eval()

int()

float()

hex()

oct() ….

1. ***User-defined*** Functions

The functions that are developed by the user are called ‘User-defined functions’. They are developed at the time of writing a program in Python.

Like pre-defined functions, the user-defined functions are also developed with/without arguments and with/without return values.

In Python, a function is identified by the keyword ‘def’.

***To define a function (called)***

Syntax: ***formal arguments***

def function\_name([argslist]):

----

statements;

***To call a function (calling)***

To execute the code of either a user-defined or a pre-defined function that function must be calling.

Syntax: ***actual arguments***

[variable =] function\_name([args])

**Arguments (Parameters)**

In python, an argument is an object which passes a value from one function to another. The arguments are of two types. They are

* + 1. Actual arguments – applied in calling function, sends a value from calling to called function
    2. Formal arguments – applied in called function, received values send by actual arguments.

Ex.

Functions without arguments and without return value

def fact():

n=int(input("Enter a no: "))

f=1

for i in range(1,n+1):

f=f\*i

print("Factorial : ",f)

fact()

Functions with arguments

***To find factorial of a no***

def fact(n):

f=1

for i in range(1,n+1):

f=f\*i

print("Factorial of ",n," : ",f)

n=int(input("enter a no : "))

fact(n)

***To find mn***

def power(m,n):

p=1

for i in range(1,n+1):

p=p\*m

print("Power value of ",m," raised to ",n," : ",p)

m=int(input("entet m : "))

n=int(input("entet n : "))

power(m,n)

***Function returns a value***

If a function returns a value then return statement is used in function definition, to return a value.

Syntax:

def fun\_name(argslist):

---

Statements

Return statement

def add(a,b):

return a+b

def sub(a,b):

return a-b

def mul(a,b):

return a\*b

def div(a,b):

return a/b

def mod(a,b):

return a%b

a=15

b=8

print("a : ",a," b : ",b)

print("a+b : ",add(a,b))

print("a-b : ",sub(a,b))

print("a\*b : ",mul(a,b))

print("a/b : ",div(a,b))

print("a%b : ",mod(a,b))

def bonus(sal):

if sal>=20000:

bon=sal\*.05

elif sal>=10000:

bon=sal\*.04

else:

bon=sal\*.03

return bon

salary=float(input("Enter salary : "))

b=bonus(salary)

tot=salary+b

print("bonus : ",b)

print("Total : ",tot)

**Returning multiple values from a function:**

In c/C++/Java a functionreturns a value. But in Python it is possible to return multiple values from a function.

Syn: return var1,var2,…

def totavg(a,b,c):

t=a+b+c

avg=t/3

return t,avg

x,y=totavg(12,20,35)

print('Total : ',x)

print('Average : ',y)

def arithmetic(a,b):

i=a+b

j=a-b

k=a\*b

l=a/b

m=a%b

return i,j,k,l

x=arithmetic(12,7)

print(x)

for i in x:

print(i)

**Types of arguments**

**In python the arguments are classified into 4 types. They are**

1. **positional arguments**
2. **keyword arguments**
3. **default arguments**
4. **Variable length arguments**

**Positional arguments**

These are the arguments passed to a function in correct positional order. The number of arguments and position of arguments must be matched. If we change the order then result may be changed. If we change the number of arguments then we will get error.

def sum(a,b):

print(a+b)

sum(10,20)

sum(100,200)

def display(a,b,c):

print(a)

print(b)

print(c)

display(10,20,30)

display(101,"kumar",23000.00)

***Keyword arguments***

If We send arguments value with the name of the argument, is called keyword arguments.

Ex. def wish(name,msg):

print("Hello! ",name,msg)

wish("Sachin","Good Evening")

wish(name="Rakesh",msg="Good Night")

def show(eno,ename,salary):

print('Number : ',eno)

print("Name : ",ename)

print("Salary : ",salary)

print("Employee 1 : ")

#keyword arguments

show(eno=101,ename="kumar",salary=25000)

print("Employee 2 : ")

#positional arguments

show(102,"Mahesh",17500.00)

**Note:** positional arguments cannot be allowed after keyword arguments

ex. show(**eno=101,ename=’anil’**,25000)

**Default arguments**

if we omit the argument values in calling function then Python gives an error. To overcome this problem default arguments are used. With this, we can define the default values for the arguments in def function ie function definition.

Syntax:

def fun\_name(arg1=value, arg2=value , …):

Note: After default arguments we should not take non default arguments

def show(eno=999,ename='abcd',salary=2222.22):

print('Number : ',eno)

print("Name : ",ename)

print("Salary : ",salary)

print("\nEmployee 1 : ")

show(101,'anand',34000)

print("\nEmployee 2 : ")

show(102,"Mahesh")

print("\nEmployee 3 : ")

show(103)

print("\nEmployee 4 : ")

show()

**Variable length arguments**

Sometimes we can pass variable number of arguments to our function, such type of arguments are called variable length arguments. We can declare a variable length argument with \* symbol as follows:

Syntax: def fun\_name(\*arg):

def show(\*n):

for i in n:

print(i,end= " ")

print()

show()

show(10)

show(10,30)

show("apple","banana","orange")

show(101,"kumar","manager",23500.00)

***Recursive Functions***

If a function is calling itself is called recursive function. The recursive functions are also use to calculate the result.

For eg. In maths, factorial of a number will be calculated as follows:

5! = 5X4!

= 5X4X3!

= 5X4x3x2!

= 5x4x3x2x1!

= 5x4x3x2x1

= 120

In programming languages,

fact(5)= 5\*fact(4)

= 5\*4\*fact(3)

= 5\*4\*3\*fact(2)

= 5\*4\*3\*2\*fact(1)

= 5\*4\*3\*2\*1

= 120

***To find the factorial of a no using recursive functions***

def fact(n):

if n==1:

return 1

else:

return n\*fact(n-1)

n=int(input("enter n: "))

k=fact(n)

print("Factorial : ",k)

***To display the no.s from n to 1 using recursive functions***

***lambda function:***

syntax: lambda argslist : expression

ex. lambda a:a\*a

i=lambda a,b: a+b

j=lambda a,b: a-b

k=lambda x,y:x\*y

m=int(input("enter m : "))

n=int(input("enter n : "))

print("m+n : ",i(m,n))

print("m-n : ",j(m,n))

print("m\*n : ",k(m,n))

k=lambda a,b : a if a>b else b

m=int(input("enter m : "))

n=int(input("enter n : "))

print("Big : ",k(m,n))

**Note: Lambda Function internally returns expression value and we are not required to write return statement explicitly.**

k= lambda a:a\*a\*a

k1=lambda a: a\*\*3

n=int(input("Enter n : "))

print("cube : ",k(n))

print("cube : ",k1(n))

**Types of variables**

In general, the variables are classified into 2 types. They are

1. Global variables
2. Local variables

***Global variables:***

The variables that are declared in outside of a function are called ‘global variables’. They are accessed in one or more functions ie in the entire program.

Ex.

g=100

def fun1():

print(g)

def fun2():

print(g)

fun1()

fun2()

**Local variables:**

The variables that are declared inside a function are called ‘local variables’. They are accessed within the function in which they are declared. They are not accessed in another function.

Ex. g=100

def fun1():

a=10

print("In function 1 :")

print("Local variable : ",a)

print("Global variable : ",g)

def fun2():

print("\nFunction 2 : ")

# print("Local : ",a)

print("Global : ",g)

fun1()

fun2()

***global keyword***

This keyword is used to provide global accessibility for the local variable.

Syntax: global variablename

Ex. g=100

def f1():

global a

a=10

print(a)

print(g)

def f2():

print(g)

print(a)

f1()

f2()

**filter() function**

It is used to filter values from the given sequence based on some condition.

**Syntax: listobj=list(filter(function,sequence))**

def isEven(x):

if x%2==0:

return True

else:

return False

a=[2,5,6,23,8,12]

b=list(filter(isEven,a))

print(b)

using lambda function

b=list(filter(lambda x:x%2==0,a))

print(b)

***map() function***

It is used to apply some functionality and generate new element with the required modification for every element present in the given sequence.

Syntax: list= list(map(function,sequence))

def doublenum(x):

return x\*2

a=[2,5,6,23,8,12]

print(a)

b=list(map(doublenum,a))

print(b)

***function aliasing***

It means to define an alternative name to a function ie alias name.

syntax: aliasname = functionname

ex.

def wish(name,msg):

print("Hello! ",name,msg)

greet = wish

wish("sachin"," Good Morning")

greet("Rahul", " Good Night")

***Decorator functions***

Decorator is a function which can take a function as argument and extend its functionality and returns modified function with extended functionality.

The main objective of decorator function is we can extend the functionality of existing functions without modifications in that function.

Syn: def décor(func):

def inner(args):

----

return inner

To call décor function

Syn: @decor

def function(args)

statements

def decor(func):

def inner(name):

if name=='sunny':

print("Hello! Sunny Good Night")

else:

func(name)

return inner

@decor

def wish(name):

print("Hello! ",name," Good Morning")

wish('sachin')

wish('anil')

wish('sunny')

To call the same function with/without décor function

To call the same function without décor function we use the following:

Syntax: decorfunction=décor(wish)

def decor(func):

def inner(name):

if name=='sunny':

print("Hello! Sunny Good Night")

else:

func(name)

return inner

def wish(name):

print("Hello! ",name," Good Morining")

decorfunction=decor(wish)

wish('sachin')

wish('anil')

wish('sunny')

print("\n---with decor function----")

decorfunction('Rakesh')

decorfunction('Akash')

decorfunction('sunny')

note :

1. If a function call with the function name (wish) then function will be executed directly.
2. If a function calls with décor function then decorator function will be executed.

**Modules**

Module is a collection of variables, functions and classes. It is useful to group the elements. Every python program is a module by default, called main module.

The modules are 2 types.

Predefined modules

User defined modules

***Pre-defined modules***

The modules that are provided in python are called pre-defined modules.

Ex. **math , sys, calendar, random etc.**

***User-defined modules***

The modules that are defined by the user are called user-defined modules. A module is a collection of variables and functions.

Every python program is a user-defined module by default. ***The program name is the name of the module.***

***Import statement***

To work with either a pre-defined or user-defined module that module must be linked to the program. For this, import statement is used.

Syntax: import modulename

Example:

Ex.

a=100

b=200

def fun1():

print(a,b)

save as sample.py - now, sample is the module name

note: to work with sample module, use

import sample

**To acceass a module varible**

Syntax: modulename.variable

Ex. sample.a

sample.b

**To access a function**

Syntax: modulename.function(args)

Ex. sample.fun1()

Program:

import sample

print("--- using variables---")

print(sample.a)

print(sample.b)

print("\n---using function---")

sample.fun1()

**maths module**

def prime(n):

c=0

for i in range(1,n+1):

if n%i==0:

c=c+1

if c==2:

print("Prime")

else:

print("Not prime")

def oddeven(n):

if n%2==0:

print("even")

else:

print("Odd")

Import modulename

***To define an alias name for a module***

Syntax: import modulename as aliasname

import maths as m

n=int(input("Enter a no"))

m.prime(n)

m.oddeven(n)

***To import multiple modules***

Syntax: Import module1,module2,…

import maths,sample

sample.fun1()

n=int(input("Enter a no"))

maths.prime(n)

maths.oddeven(n)

2nd method

import maths as m,sample as s

s.fun1()

n=int(input("entet n : "))

print(n," is : ",end=" ")

m.prime(10)

print(n," is : ",end=" ")

m.oddeven(23)

***using from keyword***

it is used to access the variables or functions without modulename.

Syntax: from modulename import variable, function

from maths import prime,oddeven

n=int(input("entet n : "))

print(n," is : ",end=" ")

prime(n)

print(n," is : ",end=" ")

oddeven(n)

from sample import \*

print("x = ",a)

print('Y= ', b)

fun1()

note: \* represents both (all) variables and functions of the module

***to define alias name for variables and functions***

It is used to define an alias name for both variables and functions.

Syntax: from modulename import variable as name1, function as fun1

from sample import a as x,b as y, fun1 as bdps

print("x = ",x)

print('Y= ', y)

bdps()

#employ.py

def bonus(job,sal):

if job=='manager' or job=='MANAGER':

bon=sal\*.05

elif job=='clerk' or job=='CLERK':

bon=sal\*.03

else:

bon=sal\*.04

return bon

import employ as e

eno=int(input("Enter employee no"))

ename=input("Enter name ")

job=input("Enter job ")

sal=float(input("enter salary "))

bon=e.bonus(job,sal)

ts=sal+bon

print("Bonus : ",bon)

print("Total : ",ts)

***dir() function***

This function is used to display all members of current module or a specified module.

Syntax: dir([modulename])

Ex. print(dir())

print(dir(sample))

**Note**: For every module at the time of execution Python interpreter will add some special properties automatically for internal use.

x=10

y=20

print(dir())

print(\_\_builtins\_\_)

print(\_\_doc\_\_)

print(\_\_package\_\_)

print(\_\_name\_\_)

print(\_\_file\_\_)

print(\_\_spec\_\_)

For every Python program , a special variable name will be added internally. This variable stores information regarding whether the program is executed as an individual program or as a module.

If the program executed as an individual program then the value of this variable is

main

If the program executed as a module from some other program then the value of this variable is the name of module where it is defined.

***Random module***

This module is used to define random values. For this, it provide several functions to generate random numbers. We can use these functions while developing games, in cryptography and to generate random numbers for authentication.

Functions:

* + - * 1. random()

This function generates float values between 0 and 1 (not inclusive) ie 0<x<1

Eg.

from random import \*

for i in range(1,15):

print(random())

1. randInt(x,y) – It returns random integer values between 2 given numbersfrom random import \*

for I in range(1,11):

print(randint(1,20)

1. uniform(x,y) - It returns random float values between 2 given numbers(not inclusive)

from random import \*

for I in range(1,11):

print(uniform(5,20)

1. randrange(start,end,step)

Returns a random number from range start<= x < end

start - is optional and default value is 0

step - is optional and default value is 1

Ex. from random import \*

for I in range(1,11):

print(randrange(1,11))

from random import \*

for I in range(1,11):

print(uniform(1,11,2))

1. choice(list)

It doesn’t return random numbers ie used to return other than numbers (string,tuple,…). It will return a random object from the given list or tuple.

from random import \*

x=["akash","avinash","rakesh","mahesh"]

for i in range(1,8):

print(choice(x))

***Calendar Module***

Functions of calendar:

***month()*** – It is used to display the calendar of a specified month and year.

import calendar

yy=2018

mm=12

print(calendar.month(yy,mm))

***datetime Module***

It is used to get the current date and time in the system. This module contain various classes. They are:

1. date class
2. time class
3. datetime class
4. timedelta class
5. date class

It is used to represent a date value.

Ex. import datetime

d=datetime.date(2020,12,25)

print(d)

using constructor

syntax: date(yy,mm,dd)

from datetime import date

d=date(2020,12,25)

print(d)

#to get current date

d=date.today()

print('Current date : ',d)

print('Year : ',date.today().year)

print('Month : ',date.today().month)

print('Day : ',date.today().day)

# 2nd method

d=date.today()

print('Current date : ',d)

print('Year : ',d.year)

print('Month : ',d.month)

print('Day : ',d.day)

1. **time class**

It is used to represent a time value.

**Constructor:**

Syn: time(hour,minute,second)

from datetime import time

t=time()

print('Current Time : ',t)

t1=time(23,55,23)

print('Time is : ',t1)

#using default argument

t2=time(hour=11,minute=34,second=45)

print(t2)

to print hour, minute and second

a=time(10,45,35)

print('Time is : ',a)

print("Hour : ",a.hour)

print("Minute : ",a.minute)

print("Second : ",a.second)

1. datetime class

It is used to represent both date and time values.

Constructor:

Datetime(hour,minute,second)

Datetime(hour,minute,second,hour,minute,second)

from datetime import datetime

d1=datetime(2022,10,12)

print(d1)

a=datetime(2020,9,23,10,45,55)

print(a)

print("year : ",a.year)

print("month : ",a.month)

print("Dary : ",a.day)

print("Hour : ",a.hour)

print("Minute : ",a.minute)

print("Second : ",a.second)

***now()*** – It returns both date and time in a system.

from datetime import datetime

d=datetime.now()

print(d)

using import

import datetime

d=datetime.datetime.now()

print(d)

to display current date and time

from datetime import datetime

d=datetime.now()

print("Current date and time : ",d)

***date and time formatting***

**strftime()** – This function is used to display date and time values in a specific format ie datetime to string.

from datetime import datetime

d=datetime.now()

print("Current date and time : ",d)

a=d.strftime('%y-%m-%d')

print(a)

b=d.strftime('%y/%m/%d')

print(b)

c=d.strftime('%d/%m/%d , %H:%M:%S')

print(c)

strptime() – It is used to convert string into datetime object ie system format.

Syntax: strptime(datestring,format)

Ex.

d=datetime

a=d.strptime('26 june 2021','%d %B %Y')

print(a)

b=d.strptime('may,05,20','%B,%d,%y')

print(b)

**FILE HANDLING**

In general, the user entered data (input) and output of a program are primarily stored in RAM temporarily. To store that data permanently in secondary memory (hard disk) files are used.

**TYPES OF FILES**

In Python, the files are classified into 2 types.

* + - * 1. Text Files
        2. Binary Files

***Text Files***

They are used to store text data ie character data

***Binary Files***

They are used to store binary data ie images, audio, video etc.

**To open a file**

In general, to store data onto a file or retrieve data from a file then that file must be opened in memory. For this Python provides a built-in (pre-defined) function called open().

Syntax:

fileobject = open(filename, mode)

where filename – is a string expression

where mode represents

1. w – opens a new file to write data
2. r – opens an existing file to read data
3. a – opens a new/existing file to write data
4. r+ - opens a file to read and write data
5. w+ - opens a file to write and read data
6. a+ - opens a file to append and read data
7. x – opens a new file exclusively for write operation

Note: the above modes are used for text files only. For binary files they must suffixed with ‘b’.

ex. wb, rb, ab, r+b, w+b, a+b, xb

Ex.

f=open(“abc.txt”,”w”)

**To write data onto a file**

write(string) – write string data onto a file

writelines() – writes multiple lines of text

**To read data from a file**

read() – reads total data from a file

read(n) – reads n no.of characters from a file

readline() – reads a line

readlines() – reads all lines into a list

**To close a file**

After completion of our operations, finally an opened file must be closed. For this, close() function is used.

Syntax: object.close()

Ex. f.close()

**Properties of File object**

When we open a file then file object will automatically create in memory. For this object various properties are defined.

They are:

***name* –** name of the file

***mode*** – mode in which the file is opened

***readable*** – returns boolean value to represent whether the file is readable or not

***writable*** – returns boolean value to represent whether the file is writable or not

***closed*** – returns Boolean value to represent whether the file is closed or not

Ex. f=open("sample.txt","w")

print("Name of the file : ",f.name)

print("Mode : ",f.mode)

print("Readable : ",f.readable)

print("Writable : ",f.writable)

print("closed : ",f.closed)

f.close()

print("closed : ",f.closed)

***to write data on a file (creating a file)***

f=open("abc.txt","w")

s1="welcome to bdps"

s2="bdps software limited"

f.write(s1)

f.write(s2)

f.close()

print("File created.....")

***To read data from a file (display a file)***

***To write and display a file***

f=open("emp.txt","a")

s1="Welcome to BDPS"

s2="Software Training"

s3=" and Placement Institution"

f.write(s1)

f.write(s2)

f.write(s3)

f.close()

print("File contents----")

f=open("emp.txt","r")

x=f.read()

print(x)

***to copy a file***

f1=open("abc.txt","r")

f2=open("a.txt","w")

s=f1.read()

f2.write(s)

print(f1.name," is copied into ",f2.name)

f1.close()

f2.close()

***with statement***

It is used to open a file in Python. But the opened file will automatically closed by python internally ie no need to close a file explicitly.

It is used to group file operations statements as a block. After the block the file was automatically closed.

Syntax:

with open(filename,mode) as fileobject:

----

----

with open("a.txt","w") as f:

f.write("welcome to bdps")

print("Is closed : ",f.closed)

print("----File contents-----")

f=open("a.txt","r")

x=f.read()

print(x)

To write multiple lines

name=input("enter filename : ")

f=open(name,"w")

lst=["apple\n","Banana\n","Grapes\n","Orange"]

f.writelines(lst)

f.close()

f=open(name,'r')

x=f.read()

print(x)

To read multiple lines

f=open(name,'r')

lst=f.readlines()

for i in lst:

print(i,end="")

f.close()

f=open("e.txt","w")

eno='101'

ena='kumar'

sal=12500

'''

l=[eno,ena,str(sal)]

f.writelines(l)

f.close()

'''

f.write(eno)

f.write(ena)

f.write(str(sal))

f.close()

f=open("e.txt","r")

x=f.read()

print(x)

Using read(n)

f=open(name,'r')

print(f.read(8))

print(f.readline())

print(f.read(5))

print(f.read())

using readline()

f=open('x.txt','r')

print(f.readline())

print(f.readline())

print(f.readline())

f.close()

f=open('x.txt','r')

for i in f:

print(i,end="")

f.close()

***tell()*** – It is used to represent the cursor to the current position in a file ie returns the current cursor position

f=open('x.txt','r')

print("Current pos : ", f.tell())

print(f.read(10))

print("Pos : ",f.tell())

f.close()

***seek(offset)*** – It is used to move the cursor to a specified position in a file

f=open('x.txt','r')

print("Current pos : ", f.tell())

print(f.seek(10))

print("Pos : ",f.tell())

print(f.read())

f.close()

f=open('x.txt','r+')

print(f.read())

f.seek(11)

f.write("HCL ")

f.seek(0)

print(f.read())

f.close()

**How to check a file exist or not in system**

Python provides a library called os to get information about files in our computer. It has a sub module path to represent the files in our system. To check a file it provides the function called isFile().

Syntax: os.path.isfile(fname)

import os,sys

name=input("Enter filename : ")

if os.path.isfile(name):

print(".....File exists...")

f=open(name,'r')

else:

print("----File not found ----")

sys.exit(0)

print('--- File contents ----')

x=f.read()

print(x)

To count no.of lines, words and characters in a file

import os,sys

name=input("Enter filename : ")

if os.path.isfile(name):

print(".....File exists...")

f=open(name,'r')

else:

print("----File not found ----")

sys.exit(0)

lc=wc=nc=0

for i in f:

lc=lc+1

nc=nc+len(i)

w=i.split()

wc=wc+len(w)

print("No.of characters : ",nc)

print("No.of words : ",wc)

print("No.of lines : ",lc)

***To work with files and directories in a system***

For this os module is used. This module can provide all the files, directories and sub-directories in the system. To work with os module, **import os** will be used

**listdir()** – It is used to list or display the files in a directory

Syn: obj.listdir([path])

If path is omitted then it display the files in the current working directory

Ex. import os

d=os.listdir()

print("--Current Directory list: ---")

for i in d:

print(i)

To display in a specified directory

import os

p=input("Enter path : ")

d=os.listdir(p)

for i in d:

print(i)

***Delete Files and Directories in a path***

Sometimes, we need to delete files and directories that are no longer used. For this, Python provides various functions to delete files and folders.

***remove()*** – It is used to delete(remove) a file

syn: os.remove(filename)

ex. import os

file=input("Enter filename :")

if os.path.exists(file):

os.remove(file)

print(file," removed ....")

else:

print(file," does not exist")

***file.unlink()*** – It is also used to delete a file

This function is provided in **pathlib** module in python 3.4 onwards. To set file path use pathlib.Path.

ex.

import pathlib

f=input("Enter filename : ")

file=pathlib.Path(f)

file.unlink()

***os.rmdir()*** – used to remove an empty directory (folder)

import os,sys

name=input("enter directory name to remove :")

if os.path.isdir(name):

os.rmdir(name)

print("----Directory removed----")

else:

print(' ---- Directory does not exist ----')

***To remove a non-empty directory:***

For this python provides a function called **rmtree() in shutil** module.

Ex. import os,shutil

name=input("Enter directory name to remove :")

if os.path.isdir(name):

shutil.rmtree(name,ignore\_errors=True)

print(name,' removed')

else:

print(name, " doesnot exist"))

***os.rename() -***  It is used to change the name of a directory

Syn: os.rename(oldname,newname)

import os

os.rename("sub1","sachin")

print("Directory renamed...")

print(os.listdir())

***os.walk()*** - It is used to display all contents of current working directory

syntax: os.walk(separator)

import os

for dpath,dname,dfile in os.walk("."):

print("Current path : ",dpath)

print("Directories : ",dname)

print("Files : ",dfile)

print()

***os.stat() –*** It is used to display the properties (statistics) of a file like size, last accessed time, last modified time etc. For this the following variables are defined.

***st\_mode 🡪 protection bits***

***st\_info 🡪 inode number***

***st\_dev 🡪 device***

***st\_uid 🡪 userid of owner***

***st\_gid 🡪 group id of owner***

***st\_size 🡪 size of file in bytes***

***st\_atime 🡪 recent access time***

***st\_mtime 🡪 last modified time***

***st\_ctime 🡪 most recent meta data change***

***syntax: os.stat(filename)***

***To print all statistics of a file***

import os

st=os.stat("rmdir.py")

print(st)

print(“---Using for ----“)

print("---Statistics of a file ----")

for i in st:

print(i)

***To print specified statistics***

print("File size : ",st.st\_size)

print("File accessed time : ",st.st\_atime)

print("User id : ",st.st\_uid)

print(“File accessed time :”,datetime.fromtimestamp(st.st\_atime))

**Exception Handling**

When we are typing a program some errors have been occurred due to typing mistakes or unusual conditions. These errors may caused to system crash or abnormal program termination. The errors are classified into 2 types.

1. syntax errors
2. runtime errors (exceptions)

***Syntax errors***

The errors that are occurred due to typing mistakes or syntax are called ‘syntax errors’.

Eg. a=10

print a - print(a)

print(‘a= ,a) - print(‘a= ‘,a)

***Runtime Errors***

The errors that are occurred at the runtime of a program are called ‘runtime errors’. The runtime errors are known as ‘Exceptions’. These exceptions may be caused to system crash or abnormal program termination.

To handle those exceptions python uses **Exception Handling** mechanism. This mechanism uses **try-except** keywords.

***try:***

The statements caused to exceptions can be specified in try block.

Syn: try:

Statements

**except**

it is used to handle the exceptions raised in try block. It is similar to catch in C++/Java.

Syn: except:

Statements

**Types of Exceptions**

The exceptions are classified into 2 types. They are:

1. pre-defined exceptions
2. user-defined exception

***Pre-defined Exceptions***

The exceptions that are already defined in Python are called ‘pre-defined’ exceptions. Some of the important exceptions are:

**ZeroDivisionError –** raised when division by zero attempted

**TypeError** - raised when a function or operation applied on incorrect object

**ValueError –** raised when given value not matched

**ImportError** – raised when imported module not found

**IndexError** – raised when index is wrong

**KeyError** – raised when a key (variable) not defined

**IndentationError -**  raised when indentation is not correct

**FileNotFoundError -**  raised when a file not found

**EOFError -**  raised when the control reaches an end of file

***Except is used in various types as follows:***

1. except ZeroDivisionError:
2. except ZeroDivisionError as msg:
3. except (ZeroDivisionError,ValueError) :
4. except (ZeroDivisionError,ValueError) as msg:
5. except: - is called default exception

***To handle ZeroDivisionError***

try:

a=int(input("enter a :"))

b=int(input("enter b : "))

c=a/b

print(c)

except ZeroDivisionError:

print("divisible by 0 attempted")

except: #default exception

print("Exception raised")

try:

a=int(input("Enter a : "))

print(a)

except ValueError:

print("Invalid number")

try:

a=int(input("Enter a : "))

print(b)

except NameError:

print("Invalid name")

***To handle multiple exceptions***

try:

a=int(input("enter a :"))

b=int(input("enter b : "))

c=a/b

print(c)

print(d)

except ZeroDivisionError:

print("divisible by 0 attempted")

except ValueError:

print("Invalid number")

except NameError:

print("Invalid Name")

except:

print("Exception raised")

***using except (ZeroDivisionError,ValueError) as msg:***

try:

a=int(input("Enter a : "))

b=int(input("Enter b : "))

c=a/b

print(c)

print(d)

except ValueError as msg:

print(msg)

except (NameError,ZeroDivisionError) as msg:

print(msg)

***finally***

It is also be used as an exception handle block. But this block will be executed whether an except block may/may not executed. It will be executed at last by default.

Ex.

try:

a=int(input("enter a :"))

b=int(input("enter b : "))

c=a/b

print(c)

except ZeroDivisionError:

print("divisible by 0 attempted")

finally:

print(“End of the Program”)

***try-except-else***

else is also one of the block used in exception handling. This block will be executed when there is no exception ie if try block executed successfully then only else block will be executed.

try:

a=int(input("enter a :"))

b=int(input("enter b : "))

c=a/b

print(c)

except ZeroDivisionError:

print("divisible by 0 attempted")

except ValueError:

print("Invalid number")

else:

print("Hai")

finally:

print("End of Program")

***Nested try***

It means one try block may contains another try block.

try:

a=int(input("Enter a : "))

b=int(input("Enter b : "))

try:

c=a/b

print(c)

except ZeroDivisionError:

print('Division by zero attempted')

finally:

print("Inner finally...")

except ValueError:

print('Invalid Data given')

***exc\_info()*** – This function is used to display the information about the raised exception. This function is provided in sys module. With this, we can handle any type of exception.

Ex. import sys

try:

a=int(input("Enter a :"))

b=int(input('Enter b : '))

c=a/b

print('a/b = ',c)

except:

print('Oops! ', sys.exc\_info()[0],' occured.')

***\_\_class\_\_*** - In Python every exception can derived from Exception class only. Since, it is used to handle any exception raised in a program. It is also similar to exc\_info().

Ex. import sys

try:

a=int(input("Enter a :"))

b=int(input('Enter b : '))

c=a/b

print('a/b = ',c)

except Exception as e:

print(' OOPs! ', e.\_\_class\_\_,' occured.')

***User defined Exceptions***

The exceptions that are defined by the user are called ‘user defined exceptions’.

These exceptions are created, raised and handled by the user. But the predefined exceptions are automatically raised in a program and handled by the user.

To create an Exception

The user defined exceptions are created as follows:

Syntax: **class exceptionclassname(Exception):**

**def \_\_init\_\_(self,arg):**

**self.msg=arg**

Ex class NotEligibleException(Exception):

def \_\_init\_\_(self,arg):

self.msg=arg

**Raise statement**

This statement is used to raise a user defined exception at some situation. It is applied as follows:

Syntax: **raise exceptionclassname(message)**

without try-except

age=int(input("Enter age : "))

if age<18:

raise NotEligibleError("you are not eligible for vote")

else:

print("You are eligible for vote")

with try-except

try:

age=int(input("enter age: "))

if age<18:

raise NotEligibleException("you are not eligible for vote")

print("You are eligible for vote")

except NotEligibleException as msg:

print(msg)

***To handle marks out of range exception***

class MarksOutOfRangeException(Exception):

def \_\_init\_\_(self,arg):

self.msg=arg

a=int(input("Enter m1 : "))

b=int(input("Enter m2 : "))

c=int(input("Enter m3 : "))

try:

if a>100 or b>100 or c>100:

raise MarksOutOfRangeException("Marks >100 given")

elif a<35 or b<35 or c<35:

raise MarksOutOfRangeException('Fail')

else:

t=a+b+c

av=t//3

print("Total : ",t)

print("average : ",av)

except MarksOutOfRangeException as m:

print(m)

***Raising Exceptions***

try:

a=int(input("Enter a :"))

if a<=0:

raise ValueError('This is not a positive integer')

print(a)

except ValueError as ve:

print(ve)

**Object Oriented Programming**

The programming which was developed using classes and objects is called ‘Object-Oriented Programming (OOPs). The developers of OOPs can define the following features.

***Features of OOPs***

1. object
2. class
3. Data Encapsulation
4. Data Abstraction
5. Inheritance
6. Polymorphism
7. Dynamic Binding
8. Message Passing

***Object***

It is a real-world entity. Each entity can have some of the properties and behavior. Properties are also known as attributes (data) and behavior is nothing but operations (code) on properties.

***Class***

It is a model/blue print for an object ie representation of an object. A class is a collection of variables(data) and methods (code).

Syn: class classname:

Variables

def fun(self):

statements

***Data Encapsulation***

It means to combine both the data (attributes) and code (functions) together. A class can show this encapsulation because a class is a collection of variables and functions.

***Data Abstraction***

It means data hiding. With this, we can hide the data of a class from outside. In this, we can work with data without knowing the implementation details,

***Inheritance***

It is one of most powerful feature in oops. Inheritance is a mechanism which is used to create new classes from existing classes. The main advantage of inheritance is the **re-usability of code.**

Existing class - Base / Parent class

New class - Derived / Child class

In inheritance, a derived class has the ability to access the properties and behavior of its base class.

Features of Base class

Features of Derived class

***Polymorphism***

It a greek word. In greek, Poly means many and morphism means forms ie one function can be define in many forms. The polymorphism is classified into various types as follows:

Polymorphism

Compile-time Run-time

Overloading overriding

Static/Early Dynamic/Late

Binding Binding

Ex. Function overloading ex. Virtual functions

Operator overloading

***Dynamic Binding***

Binding refers the function (method) to be executed to a function(method) call.

*Compile binding - I*f the compiler identifies the function to be executed to a function at compile time is called ‘static/Early’ binding.

*Dynamic binding –* If the compiler identifies the function to be executed to a function at run time is called ‘Dynamic/ Late’ binding.

***Message Passing***

With this, we can establish the communication between the objects of a class. For this, the following steps are required:

1. Create the classes for objects
2. Create the objects for classes
3. Send message (information) from one object to another with functions using arguments.

**Class:-**

A class is a collection of variables and methods (functions). A class can create a new datatype with the classname.

Syntax:

class classname:

variables

def funname(self,arg,..):

statements

Ex. class sample:

a=100

def disp(self):

print(self.a)

**Object:-**

It is an instance of a class. For a class the memory will be allocated for that class objects only.

Syntax: obj= classname()

Ex. s = sample()

**.(dot) operator**

The dot (.) operator is used to access the members of a class ie both variables and methods.

To call a variable**:**

Syntax: object.variable

Ex. s.a=111

To call a method:

Syntax: obj.methodname()

Ex. s.disp()

***self-variable***:

self is the default variable for the constructor or instance methods of a class.

It will be the first argument for a method or constructor.

By using this we can access instance variables and instance methods of an object.

To accept and display a number

class sample:

def accept(self):

self.a=100

def disp(self):

print(self.a)

s=sample()

s.accept()

s.disp()

Ex. class employ:

def accept(self):

self.eno=100

self.ena='kumar'

self.sal=15600.00

def disp(self):

print(self.eno,self.ena,self.sal)

e=employ()

e.accept()

e.disp()

***Method with arguments:***

Like functions, a class methods are also defined with arguments. They are useful to send values to a class from outside.

Syntax: def methodname(self,args):

statements

Ex.

class employ:

def accept(self,eno,ena,sal):

self.eno=eno

self.ena=ena

self.sal=sal

def disp(self):

print(self.eno,self.ena,self.sal)

e1=employ()

a=input("Enter employee no :")

b=input("Enter name : ")

c=input("Enter salary : ")

e1.accept(a,b,c)

print("\n---Employee 1 ----")

e1.disp()

print("\n----Employee 2 ---")

e2=employ()

e2.accept(111,'Hamilton',23000)

e2.disp()

***Method returns a value:***

Syntax: def methodname(self,args):

statements

return statement

ex.

class emp:

def accept(self,a,b,c):

self.eno=a

self.ename=b

self.sal=c

def disp(self):

print("Number : ",self.eno)

print("Name : ",self.ename)

print("Salary : ",self.sal)

def bonus(self):

if self.sal>=10000:

self.bon=self.sal\*.02

else:

self.bon=self.sal\*.015

return self.bon

e=emp()

e.accept(101,'ramesh',7800)

k=e.bonus()

e.disp()

print("Bonus : ",k)

tot=e.sal+k

print("Total : ",tot)

**Constructors:**

* + - Used to construct or create an object in memory
    - Also used to initialize the variables of a class
    - Is automatically executed when an object is created.
    - Is also one of the method of a class
    - Defined with/without arguments
    - Name is init
    - Invoked once for each object

Syntax:

def \_\_init\_\_(self,[args]):

Statements

Types of Constructors:

* + - 1. Default constructors
      2. Non-Parameterized constructors
      3. Parameterized Constructors

***Default Constructors***

If any constructor is not defined in a class then Python interpreter invokes the default constructor internally.

Ex.

class sample:

a=100

def disp(self):

print("a : ",self.a)

s=sample()

s.disp()

***Non-parameterized constructor:***

If a constructor is defined without parameters (arguments) is called ‘Non-parameterized constructor’. It is used to initialize the variables of a class. It can be invoked when an object of that class is created.

Syntax: def \_\_init\_\_(self):

statements

Ex. class sample:

def \_\_init\_\_(self):

self.a=100

def disp(self):

print("a : ",self.a)

s=sample()

s.disp()

***Parameterized Constructor***

If a constructor is defined with parameters is called ‘Parameterized constructor’. For this parameterized constructor the values are send through the object declaration statement because constructors are invoked automatically when objects are created.

Syn: def \_\_init\_\_(self,args):

Statements

class emp:

def \_\_init\_\_(self,a,b,c):

self.eno=a

self.ena=b

self.sal=c

def disp(self):

print(self.eno,self.ena,self.sal)

x=input("enter number : ")

y=input("enter name : ")

z=input("enter salary : ")

e=emp(x,y,z)

e.disp()

***Constructor overloading***

If more than one constructors are defined in a class is called constructor overloading.

Note: not supported in python

***Types of variables***

The variables in oops are classified into 3 types.

They are **Instance variables (object level)**

**Static variables (class level)**

**Local variables (method level)**

***Instance variables:***

They are loaded into the memory along with class instance called object. For each object separate memory will be allocated.

The instance variables will be declared in:

1. Inside of a constructor
2. Inside of an instance method
3. Outside of a class using reference variable (object)

Note: they are defined using **self keyword**

* + - * 1. ***Inside of a constructor***

Ex. class sample:

def \_\_init\_\_(self):

self.a=10

def show(self):

print(self.a)

s=sample()

s.show()

* + - * 1. Inside of an instance method

class sample:

def \_\_init\_\_(self):

self.a=10

self.b=20

def accept(self):

self.c=30

def show(self):

print(self.a,self.b,self.c)

s=sample()

s.accept()

s.show()

* + - * 1. ***Outside of a class using reference variable***

The instance variables are also created in outside of a class with reference variable.

Ex.

class sample:

def \_\_init\_\_(self):

self.a=10

self.b=20

def accept(self):

self.c=30

def show(self):

print(self.a,self.b,self.c,self.d)

s=sample()

s.accept()

s.d=40

s.show()

***Static Variables***

* + A static variable value cannot be varied from object to object.
  + A static variable can be accessed by all objects of a class ie the no.of objects of a class can share the same memory.
  + A static variable is defined in outside of a class methods (constructors or instance methods).
  + The static variables can be accessed by static methods only. The static variables are invoked by the classname or cls variable.

Ex. class sample:

x=100

def \_\_init\_\_(self):

self.a=10

def show(self):

print("Instance var : ",self.a)

print("Static var : ",sample.x)

s=sample()

s.show()

print("In main x : ",sample.x)

print("In main x : ",s.x)

**How static variables are accessed:**

1. inside constructor: by using either self or classname
2. inside instance method: by using either self or classname
3. inside class method: by using either cls variable or classname
4. inside static method: by using classname
5. From outside of class: by using either object reference or classname

***Local variables***

The variables that are defined within a method definition are called ‘Local variables’.

class sample:

def \_\_init\_\_(self):

self.a=10

def read(self):

b=100

print("b : ",b)

def show(self):

print("a : ",self.a)

#print("b : ",self.b)

s=sample()

s.read()

s.show()

**Methods:**

Methods are used to perform the operations on variables of a class. The methods are also classified into 3 types. They are:

1. Instance methods (object level)
2. Static methods (class level)
3. Class methods

***Instance methods***

The methods that are loaded into the memory along with the class instances called objects, are called instance methods. They can use self variable as argument.

Syntax: **def methodname(self,args):**

**Statements**

class bdps:

def \_\_init\_\_(self):

self.a=10

self.b=20

def disp(self):

print(self.a,self.b)

b1=bdps()

b2=bdps()

b1.disp()

b2.disp() #print(b2.a,b2.b)

***Static methods***

* The static methods can declare with the decorator **@staticmethod** explicitly before a method definition.
* The static methods can’t allow instance and class variables
* It can’t allow self or cls variables as arguments
* Static methods can be invoked by the class name or reference variable (object)

Ex.

class sample:

def show(self):

print("---show method---")

@staticmethod

def disp():

print('---disp method---')

s=sample()

s.show()

sample.disp()

'''class sample:

a=10

b=20

@staticmethod

def disp():

print(sample.a,sample.b)

s1=sample()

sample.disp()

'''

class bdps:

@staticmethod

def add(a,b):

return a+b

def mul(x,y):

return x\*y

print(bdps.add(10,20))

print(bdps.mul(20,5))

without return value

class bdps:

@staticmethod

def add(a,b):

print("Sum : ", a+b)

def mul(x,y):

print("Product : ", x\*y)

a=int(input("Enter a : "))

b=int(input("enter b : "))

bdps.add(a,b)

bdps.mul(a,b)

to access a class variable

class sample:

a=10

@staticmethod

def show():

print(sample.a)

s=sample()

s.show()

***Class methods***

* Inside a method, if static variables (class variables) are used then is called class method.
* We can declare a class method by using @classmethod decorator
* For class method we should provide cls variable at the time of declaration
* Like static method, a class method is also invoked by the class name or reference variable (object).

Syn: @classmethod

def meth(cls,arg):

Ex.1) class bdps:

eno=11

ename='kumar'

salary=12500

@classmethod

def display(cls):

print("Number : ",cls.eno)

print("Name : ",cls.ename)

print("Salary : ",cls.salary)

bdps.display()

class sample:

a=10

b=20

@classmethod

def disp(cls,c):

cls.c=c

print("a = ",cls.a)

print("b = ",cls.b)

print("c = ",cls.c)

@classmethod

def total(cls):

t=cls.a+cls.b+cls.c

print("Total : ",t)

@classmethod

def product(cls):

p=cls.a\*cls.b\*cls.c

print("Product : ",p)

s=sample()

s.disp(30)

sample.total()

sample.product()

**To delete a static variable**

A static variable of a class can be deleted using del keyword. It will be applied inside of an instance or a static method.

Syntax: classname.variable

To delete from Inside of a class method cls variable is used.

Syntax: cls.variable

***To develop a bank transaction class***

import sys

class customer:

bankname='State Bank of India'

def \_\_init\_\_(self,name,bal=0.0):

self.name=name

self.bal=bal

def deposit(self,amt):

self.bal+=amt

print("Balance after deposit : ",self.bal)

def withdraw(self,amt):

if amt>self.bal:

print('Insufficient funds ....')

sys.exit(0)

else:

self.bal-=amt

print("Balance after withdrawal :",self.bal)

print("welcome to ",customer.bankname)

name=input("Enter name : ")

c=customer(name)

while True:

print('d - Deposit\nw - Withdraw\ne - Exit\n')

op=input("Enter your option : ")

if op=='d' or op=='D':

a=float(input("Enter amount : "))

c.deposit(a)

elif op=='w' or op=='W':

a=float(input("Enter amount : "))

c.withdraw(a)

elif op=='e' or op=='E':

print("Than Q!")

sys.exit(0)

else:

print("----Invalid option...Try again!")

To count no.of objects created for a class

class sample:

c=0

@staticmethod

def \_\_init\_\_():

sample.c=sample.c+1

@staticmethod

def show():

print("No. of objects : ",sample.c)

s1=sample()

sample.show()

s2=sample()

s3=sample()

sample.show()

s4=s2 #not affected because constructor will not be executed

sample.show()

***inner class***

If a class is defined inside of another class then is called inner class.

Syntax: class class1:

----

class class2:

----

Statements

Ex.

class outer:

def \_\_init\_\_(self):

print("----outer class execution----")

class inner:

def \_\_init\_\_(self):

print("---- inner class execution----")

def meth(self):

print("---method execution---")

o=outer()

i=o.inner()

i.meth()

class emp:

def \_\_init\_\_(self):

self.eno=101

self.ena='Kumar'

self.job='Manager'

self.sal=14500

self.db=self.dob()

def display(self):

print("Number : ",self.eno)

print("Name : ",self.ena)

print("Job : ",self.job)

print("Salary : ",self.sal)

class dob:

def \_\_init\_\_(self):

self.d=10

self.m=10

self.y=1995

def disp(self):

print("Date of Birth : %d - %d - %d"%(self.d,self.m,self.y))

e=emp()

d=e.dob()

e.display()

d.disp()

***Passing one class object to another class***

In python, it is also possible to send one class members to another class. With this, we can invoke the method of one class from another class.

class student:

def \_\_init\_\_(self,sno,sna):

self.sno=sno

self.sna=sna

def disp(self):

print('Number : ',self.sno)

print('Name : ',self.sna)

class marks:

def \_\_init\_\_(self,m,p,c):

self.m=m

self.p=p

self.c=c

def show(self,s):

s.disp()

print('Maths : ',self.m)

print('Physics : ',self.p)

print('Chemistry : ',self.c)

print('Total : ',self.tot)

print('Average : ',self.avg)

def result(self):

self.tot=self.m+self.p+self.c

self.avg=self.tot//3

s=student(101,'john')

m=marks(56,77,85)

m.result()

m.show(s)

***Destructors***

* Destructors are useful to destroy or delete objects in memory.
* Destructor is also a class method, is the lastly executed method of a class.
* But Python has an internal garbage collector, which clean up the resources used in a program.
* The garbage collector internally calls the destructor.
* A destructor name is \_\_del\_\_
* A destructor is defined as follows:

Syntax: def \_\_del\_\_():

statements

Example:

import time

class sample:

def \_\_init\_\_(self):

print("---Object initializing----")

def \_\_del\_\_(self):

print('---Destoying object---')

s=sample()

s=None

time.sleep(2)

print('---end---')

**Method overloading:**

If the same name can be used by no.of methods of a class then is called ‘method overloading’. The methods are defined with different number of arguments ie without argument, single argument, multiple arguments etc.

Note: at a time we may call one type of method only (one method only).

Ex.

class sample:

def show(self):

print("welcome to overloading")

def show(self,a):

print(a)

def show(self,a,b):

print(a,b)

s=sample()

#s.show()

#s.show(2345)

s.show(34.5,'kumar')

**INHERITANCE**

* + Inheritance is one of the most useful feature in oops.
  + It is a mechanism which is used to create new classes from existing classes.
  + With this, we can extend a class with the features of another class.
  + An existing class is called as ‘Base class/Parent class’ and newly created class is called as ‘Derived/Child class’.
  + In inheritance, a derived class can access the properties and behavior of its base class.
  + The main advantage of inheritance is the reusability of code ie the code written in one class (base) can be accessed from another class (derived).
  + It avoids the code redundancy ie re-written of code.

Syntax:

**class baseclassname**:

----

----

**class derivedclassname (baseclassname**):

----

----

**Types of Inheritances:**

1. Single Inheritance
2. Multiple Inheritance
3. Multi Level Inheritance
4. Hierarchical Inheritance
5. **Single Inheritance**

If a derived class can be inherited (derived) from only one base class then is called ‘single inheritance’.

Class A Base/Parent

Class B Derived/Child

Program for Single Inheritance

class first:

def accept(self,a):

self.a=a

def display(self):

print("a : ",self.a)

class second(first):

def read(self,b):

self.b=b

def disp(self):

print("b : ",self.b)

s=second();

x=int(input("enter no.1 : "))

y=int(input("Enter no.2 : "))

s.accept(x)

s.read(y)

s.display()

s.disp()

To accept and display student no,name and 3 subjects marks

class student:

def getdata(self):

self.sno=int(input("enter student no : "))

self.name=input("Enter name : ")

def putdata(self):

print("Number : ",self.sno)

print("Name : ",self.name)

class marks(student):

def getmarks(self):

self.m=int(input("Enter maths : "))

self.p=int(input("Enter physics : "))

self.c=int(input("Enter chemistry : "))

def putmarks(self):

print("Maths : ",self.m)

print("Physics : ",self.p)

print("Chemistry : ",self.c)

m=marks()

m.getdata()

m.getmarks()

m.putdata()

m.putmarks()

1. **Multiple Inheritance**

If a derived class is derived from more than one base classes then is called ‘multiple inheritance’.

base1 A B base2

C derived

Syntax:

**class base1:**

**----**

**class base2:**

**----**

**class derived (base1,base2):**

----

Ex.

class first:

def geta(self):

self.a=10

def puta(self):

print('a = ',self.a)

class second:

def getb(self):

self.b=20

def putb(self):

print('b= ',self.b)

class third(first,second):

def getc(self):

self.c=30

def putc(self):

print('c = ',self.c)

c=third()

c.geta()

c.getb()

c.getc()

c.puta()

c.putb()

c.putc()

1. **Multi Level Inheritance**

if a derived class is derived from another derived class which has been already derived from a base class is called ‘Multi Level Inheritance’.

Class A

Class B

Class C

Syntax:

**class base1:**

**-----**

**class derived1(base1):**

**----**

**class derived2(derived1):**

**----**

class first:

def geta(self):

self.a=10

def puta(self):

print('a = ',self.a)

class second(first):

def getb(self):

self.b=20

def putb(self):

print('b= ',self.b)

class third(second):

def getc(self):

self.c=30

def putc(self):

print('c = ',self.c)

c=third()

c.geta()

c.getb()

c.getc()

c.puta()

c.putb()

c.putc()

***Hierarchical Inheritance***

In inheritance, if no. of derived (child) classes are inherited from a base class then is called ‘Hierarchical Inheritance’. It will look like as inverted tree structure. In this, each derived class may also contain another no. of derived classes.

Ex. class A

class B class C

class A:

def geta(self):

self.a=10

def puta(self):

print('a = ',self.a)

class B(A):

def getb(self):

self.b=20

def putb(self):

print('b= ',self.b)

class C(A):

def getc(self):

self.c=30

def putc(self):

print('c = ',self.c)

a=B()

a.geta()

a.getb()

print("---From class B----")

a.puta()

a.putb();

b=C()

b.geta()

b.getc()

print("\n---From class C----")

b.puta()

b.putc()

**POLYMORPHISM**

* Polymorphism is a Greek word.
* Poly means many and morphism means forms
* One method is applied in different forms i.e. to perform different operations
* In polymorphism we have discuss about the following:

1. Duck Typing
2. Overloading
3. Overriding

**Duck Typing**

In Python we cannot specify the type explicitly. Based on provided value at runtime the type will be considered automatically. Hence Python is considered as Dynamically Typed Programming Language.

So, the same method can be defined in no.of classes for different purposes. It will be done by duck typing method.

Syntax:

def fun(obj):

obj.talk()

Ex.

class duck:

def talk(self):

print('---Quack Quack---')

class dog:

def talk(self):

print('--Bow Bow ---')

class cat:

def talk(self):

print('---meow meoww...')

def fun(obj):

obj.talk()

'''d=duck()

f1(d)

d=dog()

f1(d)

'''

d=[duck(),dog(),cat()]

for i in d:

f1(i)

**Overloading:**

Sometimes we can use the same method or operator for different purposes. For this overloading concept was used. Overloading means the same name can be used by no.of methods or operators.

There are 3 types of overloading methods are defined.

They are:

1. Method overloading
2. Constructor overloading
3. Operator overloading
4. ***Method overloading***

class Test:

def fun(self):

print('--No argument method---')

def fun(self,a):

print('--- single argument method---')

def fun(self,a,b):

print('---two arguments method---')

t=Test()

#t.fun()

#t.fun(10)

t.fun(10,20)

Note:

Python does not support method overloading. In this, method overloading the ***last method only be executed***. To overcome this problem in method overloading, default arguments or variable length arguments are used.

Using default arguments

class Test:

def sum(self,a=None,b=None,c=None):

if a!=None and b!=None and c!=None:

print("Sum of 3 numbers : ",a+b+c)

elif a!=None and b!=None:

print("Sum of 2 numbers : ",(a+b))

else:

print("Pass 2 or 3 arguments-----")

t=Test()

t.sum(10,20,30)

t.sum(10,20)

t.sum(10)

#using variable length arguments

class test:

def sum(self,\*a):

tot=0

for i in a:

tot=tot+i

print('Sum : ',tot)

t=test()

t.sum(10)

t.sum(10,20,30)

t.sum(40,60)

1. ***Constructor overloading***

If more than one constructors are defined in a class then is called ‘constructor overloading’. In this also the last constructor only be executed.

Ex.

class sample:

def \_\_init\_\_(self):

print('--constructor without arguments----')

def \_\_init\_\_(self,a):

print('---constructor with argument---')

#s=sample()

s1=sample(10)

1. ***Operator overloading***

In general, the operators like +, -, \*, > etc can perform the operations on numeric data only. They can’t perform the operations on objects. To make it possible operator overloading is used.

If the same operator will be used for multiple purposes is called operator overloading. In python, the following operators are used in operator overloading. For each operator one method is defined in python. They are:

Ex. **+ ---> object. \_add\_ (self,other)**

- ---> object.\_\_sub\_\_(self,other)

\* ---> object.\_\_mul\_\_(self,other)

/ ---> object. \_div\_ (self,other)

// ---> object. \_floordiv\_ (self,other)

% ---> object. \_mod \_(self,other)

\*\* ---> object. pow (self,other)

+= ---> object. iadd (self,other)

-= ---> object. \_isub \_(self,other)

\*= ---> object. \_imul \_(self,other)

/= ---> object. \_idiv \_(self,other)

//= ---> object. \_ifloordiv \_(self,other)

%= ---> object. imod (self,other)

\*\*= ---> object. ipow (self,other)

< ---> object. \_lt \_(self,other)

<= ---> object. \_le \_(self,other)

> ---> object.\_\_gt\_\_(self,other)

>= ---> object. \_ge \_(self,other)

Ex. class sample:

def \_\_init\_\_(self,a):

self.n=a

def disp(self,msg):

print(msg,self.n)

def \_\_add\_\_(self,other):

return self.n+other.n

def \_\_sub\_\_(self,b):

return self.n+b.n

s1=sample(100)

s2=sample(200)

s1.disp("s1 : ")

s2.disp("s2 : ")

print("s1+s2 : ",s1+s2)

print("s1-s2 : ",s1-s2)

*Method Overriding*

In inheritance, if both base and derived classes are defined the same method with same signature then is called ‘method overriding’. In this case, the derived class objects can execute the derived class methods only.

In method overriding, the base class methods are hidden by the derived class methods. To overcome this, super() keyword is used.

Syntax: super().methodname();

Without super()

Ex. class base:

def show(self):

print("---show in base class---")

class derived(base):

def show(self):

print("----show in derived class----")

d=derived()

d.show()

o/p - show in derived class

note: To invoke base class method super() is used in derived class method.

with super()

class base:

def show(self):

print("---show in base class---")

class derived(base):

def show(self):

super().show()

print("----show in derived class----")

d=derived()

d.show()

o/p - show in base class

show in derived class

Constructor overriding:

if both base and derived classes are defined with parameterized constructors then also super() is used to send values to the base class from the derived class.

Syntax:

\_\_init\_\_(self,args): - derived class constructor

super().\_\_init\_\_(args) - calling base class constructor

class base:

def \_\_init\_\_(self,eno,ename):

self.eno=eno

self.ename=ename

def show(self):

print("Number : ",self.eno)

print("Name : ",self.ename)

class derived(base):

def \_\_init\_\_(self,eno,ename,job,sal):

super().\_\_init\_\_(eno,ename)

self.job=job

self.sal=sal

def show(self):

super().show()

print("Job : ",self.job)

print("Salary : ",self.sal)

d=derived(101,'hari','clerk',6000)

print("----Employee details-----")

d.show()

REGULAR EXPRESSIONS

If we want to represent a group of Strings according to a particular format/pattern then we should go for Regular Expressions. i.e Regualr Expressions is a declarative mechanism to represent a group of Strings accroding to particular format/pattern.

Eg 1: We can write a regular expression to represent all mobile numbers

Eg 2: We can write a regular expression to represent all mail ids.

The main important application areas of Regular Expressions are

1. To develop validation frameworks/validation logic
2. To develop Pattern matching applications (ctrl-f in windows, grep in UNIX etc)
3. To develop Translators like compilers, interpreters etc
4. To develop digital circuits
5. To develop communication protocols like TCP/IP, UDP etc.

**re module**

We can develop Regular Expression Based applications by using python module called **re.** This module contains several inbuilt functions to use Regular Expressions very easily in our applications.

*compile() -* It is used to compile a pattern into RegexObject.

Ex. pattern=re.compile(“abc”)

*finditer() - It r*eturns an Iterator object which yields Match object for every Match.

Ex. matcher=pattern.finditer(“abababcabc”)

On Match object we can call the following methods.

1. start()  Returns start index of the match
2. end()  Returns end+1 index of the match
3. group()  Returns the matched string

import re

pat=re.compile("ab")

m=pat.finditer("welcome ababa to bdps ab")

for i in m:

print(i.start(),' ... ',i.end(),' ...',i.group())

#2nd method

print()

m=re.finditer("ab","welcome ababa to bdps ab")

for i in m:

print(i.start(),' ... ',i.end(),' ...',i.group())

Character Classes

They are used to search for a group of characters in a string ie pattern matching

[abc] - either a or b or c

[^abc] - except a or b or c

[a-z] - any lower case letter

[A-Z] - any upper case letter

[a-zA-Z] - any alphabet

[0-9] - any digit from 0 to 9

[a-zA-Z0-9] - any alphanumeric character

[^a-zA-Z0-9] - except alphanumeric character

import re

s="abk@bc2$5hb8"

'''m=re.finditer("[abc]",s)

for i in m:

print(i.start()," ... ", i.end() , " : ",i.group())

#other than a,b and c

print()

m=re.finditer("[^abc]",s)

for i in m:

print(i.start()," ... ", i.end() , " : ",i.group())

'''

#lower case alphabets

m=re.finditer("[a-z]",s)

for i in m:

print(i.start()," ... ", i.end() , " : ",i.group())

#other than alphabets

m=re.finditer("[^a-z]",s)

for i in m:

print(i.start()," ... ", i.end() , " : ",i.group())

#symbols only

m=re.finditer("[^a-zA-Z0-9]",s)

for i in m:

print(i.start()," ... ", i.end() , " : ",i.group())

pre defined character classes:

\s - space character

\S - Any character except space character

\d - any digit from 0 to 9

\D - any character except digit

\w - any word character [a-zA-Z0-9]

\W - any character except word character (special)

. - any character including special characters

import re

s="a7b kx@9z b7x"

m=re.finditer("\s",s)

for i in m:

print(i.start(),' ...', i.group())

print()

m=re.finditer("\d",s)

for i in m:

print(i.start(),' ...', i.group())

print()

m=re.finditer("\w",s)

for i in m:

print(i.start(),' ...', i.group())

print()

m=re.finditer("\W",s)

for i in m:

print(i.start(),' ...', i.group())

Quantifiers:

we can use quantifiers to specify the no.of occurrences to match.

a 🡪 exactly one ‘a’

a+ 🡪 atleast one ’a’

a\* 🡪 any no.of ‘a’s

a? 🡪 atmost one ‘a’ either o or 1

a{m} 🡪 exactly m no.of a’s

a{m,n} 🡪 min m and max n

import re

s="a7baam kxaa@9z aaab7ax"

mat=re.finditer(“a”,s)

for i in mat:

print(i.start(),".....",i.group())

print('\n--using a+---')

x='a+'

mat=re.finditer(x,"a7baa kxa@9z aaab7ax")

for i in mat:

print(i.start(),".....",i.group())

print('\n--using a{3}---')

x='a{3}'

mat=re.finditer(x,"a7baa kxa@9z aaab7ax")

for i in mat:

print(i.start(),".....",i.group())

print('\n--using a{2,3}---')

x='a{2,3}'

mat=re.finditer(x,"a7baa kxa@9z aaab7ax")

for i in mat:

print(i.start(),".....",i.group())

*functions of re module*

1. match()
2. fullmatch()
3. search()
4. findall()
5. finditer()
6. sub()
7. subn()
8. split()
9. compile()
10. match() – It is used to check the given pattern at beginning of target string. If the match available then we will get Match object, otherwise None.
11. Fullmatch() – It is used to match a pattern to all of target string ie complete string should be matched according given pattern.
12. Search() – It is used to search the given pattern in the target string or not.
13. Findall() – It is used to find all occurrences of the match
14. Finditer() – It returns the iterator yielding a match object fro each match.
15. Sub()- used for substitution or replacement
16. Subn() – used for no. of replacements
17. Split() – used to split the target string according to a particular pattern

import re

s=input("Enter string : ")

pat=input("Enter pattern : ")

m=re.match(pat,s)

if m!=None:

print('---Match is available at the beginning of the string---')

print('Start at : ',m.start(),' ... end at : ',m.end())

else:

print('--Not matched----')

fullmatch example

s=input("Enter string : ")

pat=input("Enter pattern : ")

m=re.fullmatch(pat,s)

if m!=None:

print('---Full string matched-----')

else:

print('--Not matched----')

search example

s=input("Enter string : ")

pat=input("Enter pattern : ")

m=re.search(pat,s)

if m!=None:

print('---Found at : ',m.start()," to ",m.end()," index")

else:

print('--Not Found----')

findall example

m=re.findall("[0-9]","ak5th6x7@12")

print(m)

m=re.findall("[a-z]","ak5th6x7@12")

print(m)

m=re.findall("[^a-z]","ak5th6x7@12")

print(m)

m=re.sub("[a-z]","#","ak5th6x7@12")

print(m)

m=re.subn("[a-z]","#","ak5th6x7@12")

print(m)

print("The Result string : ",m[0])

print("No.of replacements : ",m[1])

split example

m=re.split(",","apple,mango,banana,grapes,orange")

print(m)

for i in m:

print(i)

^ symbol – Checks whether a string starts with a given pattern or not

$ symbol – Checks whether a string ends with a given pattern or not

import re

s="Learning pyhon is very easy"

print("String is : ",s)

m=re.search('^Learn',s,re.IGNORECASE)

if m!=None:

print('---string starts with Learn')

else:

print('---not start with Learn')

to check mobile number is valid or not

import re

n=input("Enter mobile number: ")

m=re.fullmatch("[7-9]\d{9}",n)

if m!=None:

print("Valid Mobile Number")

else:

print("Invalid Mobile Number")

databases

**Database Handling**

Sometimes you need to save data entered by the user for future use. There are two ways to save data that is supplied by the user while running an application.

They are:

1. File handling.
2. Database management system.

To interface with a database using Python, you need the Python database API, which supports a wide range of database servers, such as MySQL, Microsoft SQL Server , Oracle , Etc. You need to download a separate database API module for each database you want to access. Here we want to access MySQL Database Server through Python.

The following steps are required to interact a database with Python language.

1. *Importing database module*

Ex. import cx\_Oracle or pymysql

1. *Establishing connection between the program and database, use connect method*.

Syntax: connect(host,user,passwd,database)

Ex.

con=pymysql.connect(host="localhost",user="root",passwd="root",database="bdps")

1. *To execute query statement to hold data in some object (cursor). It is the method of connection object.*

Syntax: obj=conobject.cursor()

Ex. cur=con.cursor()

1. *Executing sql queries using cursor object*

*execute(sqlquery) – to execute a single query*

*Executescript(sqlqueries) – to execute multiple queries separated by ;*

*Executemany() – to execute a parameterized query*

Ex. cur.execute("insert into emp1 values(105,'ramu',4500)")

1. *Commit or Rollback*

commit():

This method belongs to connection object.To apply the modifications to the database table, use the commit() method.

Once commit() is executed, then it’s not possible to undo the changes.

rollback():

This method belongs to connection object.The rollback() method cancels all the modifications applied to the database table. It keeps the database table when it was last saved.

con.commit()

1. *Fetch the result from a cursor*

fetchone():

This method belongs to cusor object and it fetches one row from resultset.

fetchall():

This method belongs to cusor object and it fetches all the rows from resultset.

1. Close the resources

cur.close()

con.close()

To Insert a row

import pymysql

con=pymysql.connect(host="localhost",user="root",passwd="root",database="bdps")

print('--- database connected----')

cur=con.cursor()

cur.execute("insert into emp1 values(105,'ramu',4500)")

cur.close()

con.commit()

con.close()

print("Row created...")

to insert a row by user input

import pymysql

con=pymysql.connect(host="localhost",user="root",passwd="root",database="bdps")

print('--- database connected----')

cur=con.cursor()

eno=int(input("enter no : "))

name=input("enter name : ")

sal=float(input("Enter salary : "))

cur.execute("insert into emp1 values(%d,'%s',%f)"%(eno,name,sal))

cur.close()

con.commit()

con.close()

print("Row created...")

To display the records in a table

import pymysql

con=pymysql.connect(host="localhost",user="root",passwd="root",database="bdps")

cur=con.cursor()

try:

cur.execute("select \*from emp1")

while True:

row=cur.fetchone()

if row==None:

break;

print(row[0],row[1],row[2])

except:

print('--Error in fetching rows---')

cur.close()

con.close()

***GUI Programming***

Python provides various options for developing graphical user interface applications. Most important are **Tkinter**, **JPython, PyQt,etc.**

**Tkinter Programming**

Tkinter is the standard GUI library for Python. Tkinter provides a fast and easy way to create GUI applications.

Steps for Creating a GUI application using Tkinter

* Import the '*tkinter'* module.
* Create the GUI application main window.
* Add one or more widgets to the GUI application.
* Enter the main event loop.

**Tk()** – This metnod belongs to tkinter module and creates a toplevel (main) window of an application.

Syntax: Obj=tkinter.Tk()

Ex. win=tkinter.Tk()

**geometry()** – It sets main window size

Syntax: geometry("Width x Height")

eg: win.geometry("300x200")

**title() –** It sets window title

Syntax: title(str)

Ex. win.title(“BDPS Limited”)

**mainloop()** – It Runs the main event loop

Eg: gui1.py

import tkinter from tkinter import \*

win = tkinter.Tk() win=Tk()

win.title("Python GUI");

win.geometry ("600x400")

win.mainloop()

**GUI widgets**

1. **Button**

It is used to create a button object to place on a window.

Syntax: Button(master,text,option(s))

where option represents

activebackground – background color when the button is under the cursor

activeforeground - foreground color when the button is under the cursor

bd – border width in pixels, default is 2

bg – normal background color

fg – normal foreground color

command – function/method to be called when the button is clicked

font – text font to be used for the button’s label

height – height of the button

width – width of the button in letters or pixels

padx – additional padding left and right of the text

pady – additional padding above and below the text

from tkinter import \*

from tkinter import messagebox

win=Tk()

win.geometry("400x200")

win.title('BDPS Limited')

def display():

messagebox.showinfo('BDPS','welcome to GUI')

b=Button(win,text='message',command=display)

b.pack()

win.mainloop()

to change background color of window

from tkinter import \*

from random import shuffle

import time

colors=['#e9c46a','#e76f51','#264653','#2a9d8f','#e85d04','#a2d2ff','#06d6a0','#4d908e']

win=Tk()

win.geometry("700x250")

win.title('BDPS Limited')

def change\_color():

while True:

shuffle(colors)

for i in range(0,len(colors)):

win.config(background=colors[i])

win.update()

time.sleep(1)

btn=Button(win,text="Color",command=change\_color,background='#06d6a0')

btn.pack(pady=50, padx=30)

win.mainloop()

1. **Checkbutton**

The Checkbutton widget is used to display a Checkbutton.

Syntax: Checkbutton ( master, option, ... )

options: width,height,onvalue,offvalue,etc.

gui3.py

from tkinter import \*

win = Tk()

C1 = Checkbutton(win, text = "Male", height=5, width = 20)

C2 = Checkbutton(win, text = "Female", height=5, width = 20)

C1.pack()

C2.pack()

win.mainloop()

**3.Label**

This widget implements a display box where you can place text or images.

Syntax: Label ( master, option, ... )

**4. Entry**

The Entry widget is used to accept single-line text string.

Syntax Entry( master, option, ... )

from tkinter import \*

win = Tk()

win.geometry("300x200")

L = Label(win, text="User Name")

L.pack( side = LEFT)

E = Entry(win)

E.pack(side = RIGHT)

win.mainloop()

from tkinter import \*

top=Tk()

top.title('--GUI in Python---')

B=Button(top,text="Save")

c1=Checkbutton(top,text="Music")

l=Label(top,text="Enter User")

e=Entry(top)

messagebox module

It is used to display a message dialog box in python. It is the sub module in tkinter module. Based on application requirements different relevant messages will be displayed by this module. For this, various functions are provided.

Syntax: messagebox.fun\_name(title,message,option)

The functions are:

*showinfo()*

It is used to display some relevant information to the user.

syntax: showinfo(title,message,option)

ex. messagebox.showinfo(“Information”,”Information”)

*showwarning()* – used to display warning message

*showerror()* – used to display error message

*askquestion()* – used to display question dialog box

*askokcancel()* – used to display dialog box with ok and cancel buttons

*askyesno()* – used to display dialog box with yes and no buttons

*askretrycancel()* – used to display dialog box with retry and cancel buttons