PYTHON

Contents

INTRODUCTION

What is Python?

Python is a popular programming language. It was created by **Guido van Rossum** and released in **1991.**

The PYTHON name is used in the **Famous TV show Monty Python Flying Circus.**

Why it is popular?

Today world famous website

- *- Netflix, Instagram, YouTube,spotify etc.,*
- *Google (Most of the website development using python)*
- *Microsoft .Net , Support by Python.*

It is used for:

- Web Development (server-side),
- Windows Development,

- Software Development,
- Data science,(Math's)
- System Scripting.
- Game Programming

Special Features

- Interpreter.
- Support object oriented programming.
- Other libraries attached to python.
- Python libraries attached to other languages.
- Extension library.
- Open source.
- Cpython – is a original python version 2.0,3.0 .

<u>OOPS:</u>

- Object programming language aims to implement real time entities like inheritance,polymorphism,hiding,etc. in programming.
- The main aim of oops to bind together the data and function

That operates on them so that no other part of the code can access this data except that function.

HOW TO DOWNLOAD PYTHON

Go to browser and search **python.org .**
Welcome to python.org -> downloads -> to display two version
Select 3.10 .

What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.

- Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

- The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.

Execute Python Syntax

As we learned in the previous page, Python syntax can be executed by writing directly in the Command Line:

```
>>> print("Hello, World!")
Hello, World!
```

Python Indentation

- Indentation refers to the spaces at the beginning of a code line.
- Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.
- Python uses indentation to indicate a block of code.

Example

```
if 5 > 2:
    print("Five is greater than two!")
```

Example

Syntax Error:

```
if 5 > 2:
print("Five is greater than two!")
```

Comments

- Python has commenting capability for the purpose of in-code documentation.
- Comments start with a **#,** and Python will render the rest of the line as a comment:

Example

Comments in Python:

```
#This is a comment.
print("Hello, World!")
```

Multi Line Comments

Python **does not** really **have** a syntax for **multi line comments.**

To add a multiline comment you could insert a # for each line:

Example

```
#This is a comment
#written in
#more than just one line
print("Hello, World!)
```

VARIABLES

Variables are containers for storing data values.

Creating Variables

- Python has no command for declaring a variable.
- A variable is created the moment you first assign a value to it.

Example

```
x = 5
y = "John"
print(x)
print(y)
```

Example

```
x = 4       # x is of type int
x = "Sally" # x is now of type str
print(x)
```

single or Double Quotes?

String variables can be declared either by using single or double quotes.

Example

```
x = "John"
# is the same as
x = 'John'
```

Case-Sensitive

Variable names are case-sensitive.

Example

This will create two variables:

```
a = 4
A = "Sally"
print(a)
print(A)
```

RULES

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Example

Legal variable names:

```
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"
```

Example

Illegal variable names:

```
2myvar = "John"
my-var = "John"
my var = "John"
```

Multi Words Variable Names

- Variable names with more than one word can be difficult to read.
- There are several techniques you can use to make them more readable:

Camel Case

- Each word, except the first, starts with a capital letter:

```
myVariableName = "John"
```

Pascal Case

Each word starts with a capital letter:

```
MyVariableName = "John"
```

Snake Case

Each word is separated by an underscore character:

```
my_variable_name = "John"
```

Many Values to Multiple Variables

- Python allows you to assign values to multiple variables in one line:

<u>Example</u>

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

- One Value to Multiple Variables
- And you can assign the same value to multiple variables in one line:

Example

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

DATA TYPES

- In programming, data type is an important concept.
- Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

Text Type:	str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	dict
Set Types:	set
Boolean Type:	bool

String

```
a='hai'
print(a)
a="hai"
print(a)
```

integer

```
a=1
b=2
c=3
print(a+b+c)
```

float

```
a=10.5
b=20.5
c=a+b
print( c)
```

PREDEFINED FUNCTION

- type conversion
- input function
- Isinstance function
- type function

type conversion

int(), float(),str() are the functions used to convert respective types

Eg:1

```
a='2'
b='3'
c=a+b
print(c)
print(int(a)+int(b))
```

Eg:2

```
a='2.25'
b='3.23'
print(float(a)+float(b))
```

Eg:3

```
a=2
b=3
 print(str(a)+str(b))
```

input function:

- to get values (input) from users
- all inputs are in string format

Eg:1

```
a=input("enter a number: ")
b=input ("Enter  a   number: ")
c=int(a)+   int(b)
print  ( c)
```

Eg:2

```
a=int(input("enter a number: "))
b=int(input("enter a number: "))
```

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

Eg 2.1:

```
print(int(input("enter a number: "))+ int(input("enter a
number: ")))
```

Eg:3

```
z="*"*20
print(z)
z="12"*2
print(z)
```

Eg:4

```
z="PYTHON"*int(input("enter a number"))
print(z)
```

Isinstance function:

- ❖ sy: isinstance(variable_name,datatype)

- to check whether the given variable belongs to given datatype , if yes return "True" else "False"

Eg: 1

```
c=5+3j
print(isinstance(c,complex))
a=34
print(isinstance(a,int))
z=34.5
print(isinstance(z,complex))
```

type function:

- syntax: type(variable_name or value)
- to return datatype of the variable or value

Eg: 1

```
z=12
print(type(z))
```

Eg:2

```
z=1.2
print(type(z))
```

Eg:3

```
z="12"
print(type(z))
```

Eg:4

```
z="Hello"
print(type(z))
```

OUTPUT FORMATTING:

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

print('I would like to eat {0} and {1}'.format ('pizza','Burger'))
print('Hello {name},{greeting}'.format (greeting='Good
morning',name='Ramesh'))

print(1,2,3,4)
print(1,2,3,4,sep='*')
print(1,3,5,7,sep='#',end='&')
```

OPERATORS

Operators are used to perform operations on variables and values.

Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

1.Arithmetic Operators

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

ADDITION

```
x = 5
y = 3
print(x + y)
```

SUBRACTION

```
x = 5
y = 3
print(x - y)
```

EXPONENTIATION

```
x = 2
y = 5
print(x ** y) #same as 2*2*2*2*2
```

FLOOR DIVISION

```
x = 15
y = 2
print(x // y)
#the floor division // rounds the result down to the nearest whole number
```

2.Assignment Operators

Assignment operators are used to assign values to variables:

Operator	**Example**	**Same As**
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x – 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3

3.Comparison Operators

Comparison operators are used to compare two values:

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

4.Logical Operators

Logical operators are used to combine conditional statements:

Operator	**Description**	**Example**
and	Returns True if both statements are true	x < 5 and x < 10
Or	Returns True if one of the statements is true	x < 5 or x < 4
Not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

5.Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	**Description**	**Example**
is	Returns True if both variables are the same object	x is y

is not	Returns True if both variables are not the same object	x is not y

Example (is operator)

```
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x
print(x is z)

# returns True because z is the same object as x

print(x is y)

# returns False because x is not the same object as y, even if they have the same content

print(x == y)
```

```
# to demonstrate the difference betweeen "is" and "==": this comparison returns True because x is equal to y
```

OUTPUT

```
TRUE
FALSE
TRUE
```

EXAMPLE(is not)

```
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x
print(x is not z)
# returns False because z is the same object as x
print(x is not y)
# returns True because x is not the same object as y, even if they have the same content
print(x != y)
# to demonstrate the difference betweeen "is not" and "!=": this comparison returns False because x is equal to y
```

OUTPUT

```
False
True
False
```

6.Membership Operators

Membership operators are used to test if a sequence is presented in an object:

Operator	**Description**	**Example**
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

EXAMPLE

IN OPERATOR

```
x = ["apple", "banana"]
print("banana" in x)
# returns True because a sequence with the value "banana"
is in the list
```

OUTPUT

```
TRUE
```

```
x = ["apple", "banana"]
print("orange" in x)
```

OUTPUT

```
False
```

NOT IN OPERATOR

```
x = ["apple", "banana"]
```

```
print("pineapple" not in x)
# returns True because a sequence with the value
"pineapple" is not in the list
```

Output

```
True
```

```
x = ["apple", "banana"]
print("apple" not in x)
```

output

```
false
```

7.Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits

		is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Exercise:

1. Write a program to add two number.

2. Write a python program to find square root.

3. Write a python program to calculate area of triangle.

   S=(a+b+c)/2

   Formula for Area:

$$\text{Area} = \sqrt{(s * (s - a) * (s - b) * (s - c))}$$

4. Write a python program to calculate simple interest and compound interest.

$$SI=PNR/100$$

$$CI=P(1+R/100)^{n}$$

5. Write a python program to convert Celsius to Fahrenheit.

$$F=C*9/5+32$$

6. Write a python program to convert Fahrenheit to Celsius.

$$C=(F-32)*5/9$$

7. Write a python program to swap two variable values using temp variable.

8. Write a python program to swap two variable values without using temp variable.

LIST

- Lists are used to store multiple items in a single variable.
- All item in list do not need to be of same type.
- enclosed with square bracket[ ].
- have different data types.

Eg1: Empty List

```
x=[]
```

Eg2:list of integers

```
x=[1,2,3,4]
print(x)
```

Eg:3

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

Eg4:mixed datatypes

```
x=[22,3.45,'a',"Siva KUMAR"]
print(x)
for i in x:
	print(i)
```

List Items

List items are **ordered, changeable, and allow duplicate values.**

List items are indexed, the first item has index [0], the second item has index [1] etc.

Eg5:accessing elements using index

```
x=[22,33,44]
```

```
print(x[2])
```

Note:

x=[11,22,33] x[0] → 11   x[1] → 22   x[2] → 33

seq.name index

index always starts with zero

Eg6:negative indexing

```
x=[22,33,44]
print(x[0])
print(x[1])
print(x[2])
print(x[-1])
print(x[-2])
print(x[-3])
```

Note:

- ✓ -1 index   denotes last element

- ✓ -2 index denotes 2nd element from last
- ✓ +ve index starts with 0
- ✓ -ve index starts with -1
- ✓ +ve index denotes elements from left to right
- ✓ -ve index denotes elements from right to left
- ✓ x=[22,33,44]

```
x=[22,33,44]
   ↓  ↓  ↓

    0   1   2
   -3  -2  -1
```

Eg7: Slicing

```
x=['a','b','c','d','e']
print(x[2:4])   #elements from 2nd index to 3rd     #index
print(x[:4])    #elements from 0th index to 3rd index
print(x[2:])    #elements from 2nd index to last #index
print(x[:])#all elements
print(x[1:-2])  #elements from 2nd index to 2nd element
from last
```

Eg8:change the elements

```
x=[22,33,44,55,66,77,88]
```

```
x[1]=11
print(l)
```

Allow Duplicates

Since lists are indexed, lists can have items with the same value:

Example

Lists allow duplicate values:

Eg:9

```
thislist = ["apple", "banana", "cherry", "apple", "cherry"]
print(thislist)
```

Append Items

To add an item to the end of the list, use the append() method:

Eg:10

```
x=[1,2,3]
x.append(33)
print(x)
```

Extend List

To append elements from another list to the current list, use the extend() method.

Eg:11

```
x.extend([11,22,44])
print(x)
```

Insert Items

To insert a list item at a specified index, use the insert() method.

The insert() method inserts an item at the specified index:

Eg:12

Insert an item as the second position:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

Eg13:inserting an element

```
x=[1,2,3]
x.insert(1,22)
x[2:2]=[4,5,6]
```

Eg14:deletion

```
x=[1,2,3,4,5,6,7,8,9]
del x[1]
```

```
print(x)


x.remove(9)    #removes the given element  return type →
none (doesn't return anything)
```

Eg:15

pop

```
print(x.pop(1))  # removes the element at given index  and
returns the removed element
print(x)
```

Eg:16

clear

```
x.clear()
print(x)
```

Eg17:index,count sorted,reverse methods

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

```
print(l.index(3))
print(l.count(2))
l.sort()#list is sorted
print(l)
l=[22,11,44,55,3]
print(sorted(l)) #return sorted list, doesnt sort the #list
l.reverse()
print(l)
```

Eg18:using membership operator

```
z=[1,2,3,4]
print(2 in z)
print(2 not in z)
```

Eg19:all,any,len,max,min,sum methods

```
l=[1,2,3,4]
print(all(l)) #if all elements are non zero returns #True
print(any(l)) #if any one element is non zero #returns True
print(len(l))
print(max(l))
print(min(l))

print(sum(l))
```

Matrix addition in list

Eg20:method1:

```
l1=[[1,2,3],[4,5,6],[7,8,9]]
l2=[[1,2,3],[4,5,6],[7,8,9]]
l3=[]
for i in range(3):
    for j in range(3):
        l3.append(l1[i][j] + l2[i][j])
print(l3)
```

Eg21:method2:

```
l1=[[1,2,3],[4,5,6],[7,8,9]]
l2=[[1,2,3],[4,5,6],[7,8,9]]
l3=[[0,0,0],[0,0,0],[0,0,0]]
for i in range(3):
    for j in range(3):
        l3[i][j]=l1[i][j] + l2[i][j]
print(l3)
```

Join Two Lists

There are several ways to join, or concatenate, two or more lists in Python.

One of the easiest ways are by using the + operator.

Example

Join two list:

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]

list3 = list1 + list2
print(list3)
```

```
>>nums=[25,12,36,95,14]
>>nums
[25,12,36,95,14]
>>nums[0]
25
>>nums[-5]
```

```
25
>>nums[4]
14
>>nums[2:]
[36,95,14]
>>nums[-1]
14
names=[“navin”,”kiran”,”john”]
>>names
[‘navin’,’kiran’,’john’]
>>values=[9.5,’Navin’,25]           # enter different type of
                                    data type & it accept
>>mil=[nums,names]
>>mil
[[25,12,36,95,14],[‘navin’,’kiran’,’john’]]
>>nums.append(45)                   # Add it last value
>>nums
[25,12,36,95,14,45]
>>nums.insert(2,77)
[25,12,77,38,95,14,45]
>>nums.remove(14)                   # delete value
>>nums
[25,12,77,36,95,45]
```

```
>>nums.pop(1)                # delete value using index
                             value
12
>>>nums
[25,77,36,95,45]
>>nums.pop()                 # Without specifying
                             delete last element
45
>>del num[2:]
>>nums
25,77
>>nums.extend([29,12,14,36])
>>nums
[25,77,29,12,14,36]
```

TUPLE

Tuple is an **ordered** sequence of items same as list.

Only difference is **immutable(can’t modified).**

Tuples are used to store multiple items in a single variable.

Tuple always mentioned within ().

A tuple is a collection which is ordered and **unchangeable**.
It contain duplicate value.

Eg1:empty tuple

```
t=()
print(t)
```

Eg2:

```
t=(1,2,3,4)
print(t)
t=(1,2,3,"deepak")
```

```
print(t)
```

Eg3:indexing ,slicing

```
t=(1,2,3,4)
print(t[2])
print(t[2:3])
```

Eg4:unpacking a tuple

```
t=(1,2,3)
print(t)
x,y,z=t
print(x)
print(y)
print(z)
```

Eg5:changing elements

```
t=(1,2,3)
t[1]=22     #error
print(t)
```

```
Eg6:deletion
t=(1,2,3)
del t[2]    #error


del t         #deleted
```

```
Eg7:count,index methods
t=(1,2,3,4,2)
print(t.count(2))
print(t.index(3))    #2
```

len

Tuple Length

To determine how many items a tuple has, use the len() function:

```
Eg8:len,max,min methods
t=(1,2,3,4,5)
print(len(t))
print(max(t))
print(min(t))
```

Join Two Tuples

To join two or more tuples you can use the + operator:

Example

Join two tuples:

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

Multiply Tuples

If you want to multiply the content of a tuple a given number of times, you can use the * operator:

Example

Multiply the fruits tuple by 2:

```
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2

print(mytuple)
```

SET

- unordered collection
- have unique elements
- immuatable elements
- elements enclosed with curly braces {}
- operations:
    - union,intersection,difference,...

Creation:

Method1:

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

Eg1:set of integers

```
s={22,33,76,2}
print(s)
```

Eg2:set of mixed data types

```
s={22,3.3,"Arun",'A'}
print(s)
```

Eg3:can't have dup.elements

```
s={22,3,3}
print(s)   # duplicate elements omitted
```

Eg4:empty set

```
s=set()  # s={} --> not an set its dictionary
print(type(s))
s={}
print(type(s))
```

Eg5:add method: to add an element

```
s={1,2,3}
print(s)
s.add(4)
print(s)
```

Eg6:update method: to add multiple elements

```
s={1,2,3}
print(s)
s.update([11,22,33])
print(s)
```

Eg7:discard

```
s={1,2,3}
print(s)
s.discard(3)
print(s)
```

Eg8:pop- to remove and return an element

```
s={1,2,3}
```

```
print(s)
print(s.pop())
print(s)
```

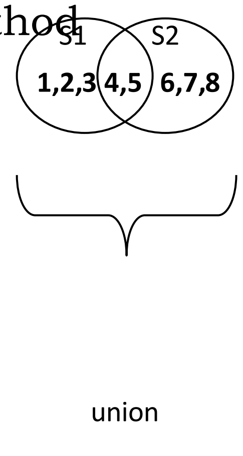
Eg9:clear - to clear set

```
s={1,2,3}
print(s)
s.clear()
print(s)
```

SET Operations:

1.Union operation:

- using '|'('OR') operator
- using union method



Eg1:

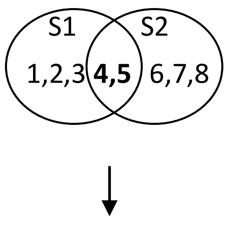
```
s1={1,2,3,4,5}
s2={4,5,6,7,8}
print(s1|s2)
print(s1.union(s2))
```

2.Intersection Operation:

- ❖ using '&'('AND') operator
- ❖ using intersection method

Eg2:

```
s1={1,2,3,4,5}
s2={4,5,6,7,8}
print(s1&s2)
print(s1.intersection(s2))
```

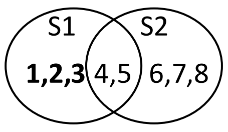


intersection

3.Difference Operation:

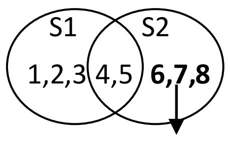
- ❖ using '-'('MINUS') operator
- ❖ using difference method

Eg3:



```
s1={1,2,3,4,5}
s2={4,5,6,7,8}
print(s1-s2)
print(s1.difference(s2))
print(s2-s1)
```

S1-S2



print(s2.difference(s1)

4.Symmetric Difference Operation: s2-s1

- ❖ using '^'('XOR') operator
- ❖ using symmetric_difference method

Eg4:

```
s1={1,2,3,4,5}
s2={4,5,6,7,8}
print(s1^s2)
print(s1.symmetric_difference(s2))
```

Eg5:using membersip operator

```
s={1,2,3,4}
print( 1 in s)
print( 1 not in s)
print( 11 not in s)
```

Eg6:max,min,sum method

```
s={1,2,7,3,4}
```

```
print("MAX:", max(s))
print("MIN:" ,min(s))
print("SUM:", sum(s))
print("Any true value: ", any(s))
print("All true values: ", all(s))
print("Sorted: ", sorted(s))
print("Length:" , len(s))
```

Eg7:vowels counting

```
s={'p','y','t','h','o','n','l','e','a','r','n'}
c=0

for a in s:
    if a in {'a','e','i','o','u'}:
        c=c+1
print("No of vowels:" ,c)
```

DICTIONARY

- unordered coll. of items
- a dictionary has a key:value pair
- all elements are combination of key and value
- elements enclosed with curly braces { }
- keys must be unique and immutable

Eg1:Empty dict

```
d={}
print(d)
print(type(d))
```

Eg2:integer keys

```
d={1:'one',2:'two'}
print(d)
```

Eg3:mixed keys

```
d={'num':201,'name':'kavin',1:'semester'}
print(d)
```

Eg4:using dict() method

```
d=dict({1:'one',2:'two'})
print(d)
```

eg5:access elements

```
d={'num':201,'name':'kavin',1:'semester'}
print(d['num'])
print(d['name'])
print(d[1])
```

Eg6:change value

```
d={'num':201,'name':'kavin',1:'semester'}
```

```
d['num']=303
print(d)
```

Eg7:**add elements**

```
d={'num':201,'name':'kavin',1:'semester'}
d['mark']=88
print(d)
```

Eg8:pop() method

```
d={'num':201,'name':'kavin',1:'semester'}
print(d.pop(1))
print(d)
```

Eg9:popitem() method

- to remove last item

```
d={'num':201,'name':'kavin',1:'semester'}
print(d.popitem())
print(d)
```

Eg10:del

```
d={'num':201,'name':'kavin',1:'semester'}
del d['num']
print(d)
```

Eg11:clear

```
d={'num':201,'name':'kavin',1:'semester'}
d.clear()
print(d)
d={'num':201,'name':'kavin',1:'semester'}
del d
print(d)
```

Eg12:items, keys,sorted method

```
d={'num':201,'name':'kavin','semester':2}
for i in d.items():
	print(i)
```

```
x=list(sorted(d.keys()))
print(x)
```

Eg13:using for loop

Method1:

```
        d={}
for x in {1,2,3}:
    d[x]=x
print(d)
```

method2:

```
        d={ }
        for x in range(10):
                d[x]=x
        print(d)
```

method3:

```
        d={x:x for x in range(10) }
        print(d)
```

```
d={x:x*x for x in range(10) }
print(d)

d={x:x*x for x in range(10) if x%2==0 }
print(d)
```

Eg15:membership operator(use keys only)

```
d={'num':201,'name':'kavin','semester',2}
print('num' in d)
```

Eg18:iteration

```
d={'num':201,'name':'kavin','semester':2}
for i in d:
        print(d[i])
```

Eg19:has_key,len,sorted,values,items methods

```
d={'num':201,'name':'kavin','semester':2}
print(len(d))
print(sorted(d))
print(d.keys())
print(d.values())
print(d.items())
```

Eg20:updation

```
d1={1:11,2:22}
d2={3:33,4:44}
d2.update(d1)
print(d2)
```

<u>More Examples:</u>

```
>>>data={1:'navin',2:'kiran',3:'harsh'}

>>>data
```

```
     {1:’navin’,2:’kiran’,3:’harsh’}

>>>data[3]
‘harsh’

>>>data.get(4)      #key error

>>>print(data.get(4))
       None

>>>data.get(4,’not found’)
       ‘not found’

>>>data.get(1,’not found’)
      ‘navin’
```

Here we creating dict with list

```
>>>keys=[‘navin’,’kiran’,’harsh’]

>>>value=[‘python’,’Java’,’Js’]

>>>data=dict(zip(keys,value))
```

```
>>>data
      {‘navin’:’python’,’kiran’:’Java’,’harsh’:’Js’}
```

Add value to dict

```
>>>data[‘monika’]=’cs’
>>>data
       {‘navin’:’python’,’kiran’:’Java’,’harsh’:’Js’,’monika’:’cs’}
```

Delete data

```
>>>del data[‘harsh’]

>>>data
      {‘navin’:’python’,’kiran’:’Java’,’monika’:’cs’}
```

Dict inside dict

```
>>>prog={‘Js’:’Atom’,’Cs’:’VS’,’python’:[‘pychram’,’sublime’],’j
ava’:{‘JSE’:’Netbeans’,’Jee’:’Eclipse’}}
```

```
>>>prog[‘js’]
      ‘Atom’

>>>prog[‘python’]
       [‘pychram’,sublime’]

>>>prog[‘python][1]
       ‘sublime’

>>>prog[‘Java’]
       {‘JSE’:’Netbeans’,’Jee’:’Eclipse’}
>>>prog[‘Java’][‘JEE’]
      ‘Eclipse’
```

CONDITIONAL STATEMENT

Simple if:

Syntax:

```
if test expression:
    Statement(S)
```

→if the text expression is false,the statement (S) is not executed.

Program to find greater of two numbers.

Example for if without else.

```
a=int(input("Enter First number.."))
b=int(input("Enter Second number.."))
if a>b:
   print("A is greater")
if b>a:
        print(“B is greater”)
```

<u>Exercise</u>: Write a program to check the given number is positive, negative or zero without else part.

<u>If else statement:</u>

```
Syntax:
        if test expression:
              Body of if statement
        else:
              Body of else statement
```

- **Indentation** is used to **separate the blocks**.

Program to input a number check whether it is divisible by 3 or not.

```
num=int(input(“Enter a number…”))
if num%3==0:
        Print(num,”is divisible by 3”)
```

```
else:
        Print(num,"is not divisible by 3")
```

EXERCISE:

Write a program to check if the number is positive or negative.

if....elif....else

```
syntax:
        if test expression:
            Body of if statement(S)
        elif test expression:
             Body of elif statement(S)
        else:
             Body of else statement(S)
```

elif→short of else if

Find the greatest of three number.

```
a=int(input("enter first number..."))
b=int(input("enter second number..."))
```

```
c=int(input(“enter third number… ”))
if a>b and a>c:
        print(“greatest is a”)
 elif b>c:
        print(“greatest is b”)
 else:
        print(“greatest is c”)
```

Exercise:

Program to input day of week in number (1…7) and print its character equivalent.

Nested if statement :

We can have a if…elif…else statement inside another if..elif..else statement. This is called nesting in computer programming.

Example:

```
num=float(input(“enter a no:”))
if num>=0:
        if num==0:
                print(“zero”)
        else:
                print(“positive number”)
else:
        print(“negative number”)
```

Example 2:

Program to find the greater of three number.

```
a=20
b=30
c=15
if a>b:
        if a>c:
                res=a
        else:
                res=c
elif b>c:
        res=b
else:
        res=c
```

```
print("greatest value=",res)
```

Note: There is no switch case statement in python.

Exercise:write a program-

(1) to find given year is leap year or not.

(2) to find second maximum of three number.

(3) to input three subject mark and check all are above 40 or using nested if.

If all are above 40 – its eligible.

If any one is below 40 – its not eligible.

(4) to input months value between 1…12 and find the month fall in which quarter.

Write a python to find second maximum of three number.

```
a=int(input("enter a value:"))
b=int(input("enter b value:"))
c=int(input("enter c value:"))
if a>b and a>c:
        if b>c:
                print("2nd largest is:",b)
        else:
                print(“2nd largest is:”,c)
elif b>c and b>a:
        if c>a:
                print(“2nd largest is:”,c)
        else:
                print(“2nd largest :”,a)
elif a>b:
        print(“2nd largest is:”,a)
else:
        print(“2nd largest is:”,b)
```

<u>LOOPING STATEMENTS</u>

for looping structure:

- *The for loop in python is used to iterate over a sequence (list,tuple,string) or other iterable obj.*
- *iterating over a sequence is called traversed.*

```
syntax:
    for var in sequence:
        body of for loop statement.
```

var→variable that takes the value of the item inside the **sequence** on each iteration.

Program1:

```
for i in “coderz”:
    print(i)
```

output:

```
    c
    o
    d
   e
    r
    z
```

the range function:

its generate sequence of number using range () function.

range (5)→generate 0 to 4(5 numbers)

we can also define start,stop and step size as

range (start,stop,step size)

step size → default to 1 if not provided.

<u>Advantage</u>:

This function does not solve all the value in memory, it would be inefficient. so it remember the start,stop,step size and generate the next number on the go.

```
for i in range (5,50,5):
            print i
```

```
o/p:
      5
      10
      :
      45
```

Exercise:

1. program to sum the even number 2…n.
2. program to calculate the average of number in a given list.
3. program to find given number is a perfect number or not.

for loop with else

```
digits=[1,2,3,4,5]
for i in digits:
      print(i)
```

```
else:
    print (“no items left”)
```

o/p:

```
      1
      2
      3
      4
      5
      no items left
```

while loop structure:

It is used to iterate over a block of code as long as the test expression(condition) is true.

usually,we use this loop we don’t know before hand,the no of time to iterate.

```
syntax:
    while <test_expression>:
            body of while loop
```

program to add natural number 1+2+3+.....+n

```
n=5
sum=0
i=1
while i<=n:
        sum=sum+i
        i=i+1
print ("the sum is",sum)
```

while loop with else:

```
i=1
while i<=3:
       print ("\n inside loop")
       i=i+1
```

```
 else:
        print("\n finally print while cond is false")
```

Exercise:write a program

(1) to check armstrong number.

(2) to find armstrong number in an interval.

(3) to count the number of digits in a number.

(4) program to reverse a given number.

(5) to check if a number is a palindrome.

(6) program to sum of digits (ex:343 → 3+4+3).

(7) program to count the no of odd & even digits ina no (ex:1234 → 2 even,2 odd).

(8) write a program to check prime number.

(9) write a program to print all prime number in a internal.

(10)write a program to find factorial of number.

(11)write a program to print fibonacci series upto n.

break statement:

```
for val in "string":
        if val=="i":
              break
        print(val)
print(“the end”)
```

o/p:

```
    s
    t
    r
    the end
```

<u>**continue statement**</u>:

```
for val in “string”
    if val ==”i”
        continue
   print(val)
print(“the end”)
```

o/p:

```
    s
    t
    r
    n
    g
    the end
```

pass statement:

```
syntax:pass
```

pass is null statement.

The different between comment & pass is

comment → interpreter ignores completely

pass → pass is not ignored but nothing will happen ,result into no operation(nop)

write a python program to find perfect number.

```
n=int(input("enter any no:"))
sum=0
for i in range(1,n):
    if(n%i==0):
        sum=sum+i
if(sum==n):
print("the number is a perfect number ")
else:
print("the number is not a perfect number")
```

FUNCTIONS

Function help break our program into smaller and modular chunks. As our program grow larger and larger , functions make it more organized and manageable.

SYNTAX:

```
def func_name(parameters):
            """doc string"""
       statements
```

def → Keyword

: → end of function header

Example:

```
def welcome_func(name):   #func definition
       print("Hello," + name + "Good morning!")


welcome_func('Ram')        #func calling
```

OUTPUT:

```
Hello,Ram Good morning!
```

The Return Statement:

SYNTAX:

```
return [expression_list]
```

1.program to find square value of a number.

```
def sample(a):
        c=a*a
        return(c)

n=int(input("Enter a number:"))
res= sample(n)      #function calling
print("The square of %d is %d"%(n,res))
```

2.write a program for factorial of number

```
def fact(n):
        x=1
        for i in range (1,n):
              x=x*i
       return x
 m= input("Enter a number…")
print("factorial value ={}".format(fact(int(m))))
```

FUNCTIONS ARGUMENTS:

```
def welcome_name(name,msg):
       print("Hello", name + ',' + msg)

welcome_func("priya","good morning")
```

OUTPUT:

```
hello priya,Good morning
```

Above function has two parameters

If we call it with one argument , shows error.

<u>VARIABLE FUNCTION ARGUMENTS:</u>

1.PYTHON DEFALUT ARGUMENTS:

```
def welcome_ func(name,msg=”Good Morning!”):
        print(“Hello”,name+’,’+msg)


welcome_func(“kumaresan”)
welcome_func(“Prakash”,”have a nice day”)
```

2.PYTHON KEYWORD ARGUMENTS:

```
def welcome_func(name,msg):
        print(“hai”,name,msg)



name=”karthick”
welcome_func(name=”karthick”,msg=”Have a nice day”)
welcome_func(msg=”have a nice day”, name=”karthick”)
welcome(name,msg= “have a nice day”)
```

name,msg → Keyword

3.PYTHON ARBITARY ARGUMENTS:

sometimes, we don’t know in advance the number of arguments that will be passed into a function.

we can use this by using arbitrary number of arguments.

That is, we going to pass n number of values into one argument.

Example1:

```
def arb_func(*names):
        for na in names:
                print(“hello”,na)

arb_func(“Priyadarshini”,”radika”,”Jayaraman”,”harsh”)
```

Example2:

```
def sample_func(*num):
        for i in num:
                print(i)
                print(type(i))

sampe_func(10,20)
sampe_func("Raj","Kumar')
sampe_func(70,"jayaraman")
```

4. **KWARGS IN PYTHON:

```
def test_variable(sstar,**dstar):
        print("formal arguments:",sstar)
        for v in dstar:
                print("another keyword arg: %s is
%s"%(v,dstar[v]))
#using keyword argument
```

```
test_variable(sstar=101,name=”Ramkumar”,sal=10000)
```

5.PYTHON RECURSION:

python program using recursive to find prime or not

```
def check(n,div=None):
        if div is None:
                div=n-1
        while div>=2:
                if n%div==0:
                        print(“not a prime number”)
                        return False
                else:
                        return check(n,div-1)
        else:
                print(“no is prime”)
                return true
```

```
n=int(input("enter number:"))
check(n)
```

6.PYTHON LAMBDA FUNCTION:

In Python , anonymous function is a function that is defined without name is called as lambda function.

It is also called as anonymous function.

syntax:

```
lambda arguments:expression
```

Lambda have any number of arguments but only one expression.

Example1:

```
f=lambda x,y:x+y
a=f(5,1)
print a
```

Example2:

same as above

```
def sumtwo(x,y):
	return(x+y)
print (“addition of two value is:”,sumtwo(5,1))
```

Exercise:

1.write a program for SI and CI using functions

2.write a program to reverse a given digits using functions

3.write a recursive program to find factorial of number.

4.write a recursive program to find sum of n number.

5.write a program power of 2 using anonymous function

BUILT IN STRING MANIPULATION

Capitalize

Program-1

'i love india'.capitalize()

isalnum

Returns true if the characters in the string <str> are alphanumeric (alphabets or numbers) and there is at least one characters ,otherwise false.

Program-2

```
s="abcd123"
s.isalnum()
true
```

isalpha

returns true if all characters in the string <str>are alphabetic and there is at least one character ,otherwise false.

program-3

```
s1='hello'
s1.isalpha()
True
S2='hello1'
s2.isalpha()
false
```

isdigit

returns true if all the characters in the string <str> are digits

program-4

```
 s3='123'
 s3.isdigit()
True
s3='123n'
s3.isdigit()
False
```

Isspace()

Returns true if there are only whitespace characters in the string <str> .there must be at least one character.

Program-5

```
s1="  "
 s1.isspace()
True
```

Isupper

Program-6

```
a='coderz'
 a.isupper()
False
```

Islower

Program-7

```
 a='CODERZ'
a.islower()
False
 a.isupper()
True
```

Istitle

Program-8

The istitle() method returns True if all words in a text start with a upper case letter, and the rest of the word are lower case letters, otherwise False.

Example:

"To Be Or Not To Be".istitle()

True

center() Method

The center() method will center align the string, using a specified character (space is default) as the fill character.

```
txt = "banana"
x = txt.center(20)
print(x)
```

find() Method

The find() method returns the index of first occurrence of the substring (if found). If not found, it returns -1.

```
txt = "Hello, welcome to my world."
x = txt.find("welcome")
print(x)
```

replace() Method

```
txt = "I like bananas"
x = txt.replace("bananas", "apples")
print(x)
```

title() Method

```
txt = "Welcome to my world"
x = txt.title()
print(x)
```

upper() Method

```
txt = "Hello my friends"
x = txt.upper()
print(x)
```

startswith() Method

Return True if the object value is start with the value which is mentioned within startwith().

```
txt = "Hello, welcome to my world."
x = txt.startswith("Hello")
```

```
print(x)
```

join() Method

Join in Python is an in-built method used to join an iterable's elements, separated by a string separator, which is specified by you.

```
myTuple = ("John", "Peter", "Vicky")
x = "#".join(myTuple)
print(x)
```

swapcase()

```
 "Hello World".swapcase()
'hELLO wORLD'
```

map(),filter(),reduce()

```
from functools import reduce

nums=[3,2,6,8,4,6,9,7]
evens=list(filter(lambda n: n%2==0,nums))
```

```
print(evens)
doubles=list(map(lambda n:n*2,evens))
print(doubles)
sum=reduce(lambda a,b:a+b,doubles)
print(sum)
```

DECORATOR

- Decorators allows programmers to modify the behavior of a function or class.
- It will add extra features into existing function at compile time itself.

Example:

```
def div(a,b):
    print(a/b)
def smart_div(func):
    def inner(a,b):
        if a<b:
            a,b=b,a
        return func(a,b)
    return inner
div=smart_div(div)
```

div(2,4)

Python Built-in Functions

```
abs() Function

 abs(-10)
10
```

all() Function

The all() function returns True if all items in an iterable are true, otherwise it returns False. It also returns True if the iterable object is empty.

```
all(s)
True
```

```
s=[0,1,2]
```

```
 all(s)
False
```

```
s=[]
all(s)
True
```

Python bin() Function

The bin() function returns the binary version of a specified integer. The result will always start with the prefix 0b .

```
>>> x=10
>>> bin(x)
'0b1010'
```

bool() Function

The bool() function returns the boolean value of a specified object. The object will always return True, unless: The object is empty, like [], (), {} The object is False. The object is 0.

```
x = bool(1)
print(x)
```

chr() Function

The chr() function returns the character that represents the specified unicode.

```
x = chr(97)
print(x)
```

filter() Function

Python's filter() is a built-in function that allows you to process an iterable and extract those items that satisfy a given condition. This process is commonly known as a filtering operation.

```
ages = [5, 12, 17, 18, 24, 32]

def myFunc(x):
    if x < 18:
        return False
    else:
        return True

adults = filter(myFunc, ages)

for x in adults:
     print(x)
```

id() Function

The id() function returns a unique id for the specified object. All objects in Python has its own unique id. The id is assigned to the object when it is created.

```
>>> x=45
>>> id(x)
1695601816
```

len() Function

```
mylist = ["apple", "orange", "cherry"]
x = len(mylist)
print(x)
```

max() Function

```
x = max(5, 10)
print(x)
```

min() Function

```
x = min(5, 10)
print(x)
```

ord() Function

The ord() function returns the number representing the unicode code of a specified character.

```
x = ord("h")
print(x)
```

pow() Function

```
x = pow(4, 3)
print(x)
```

round() Function

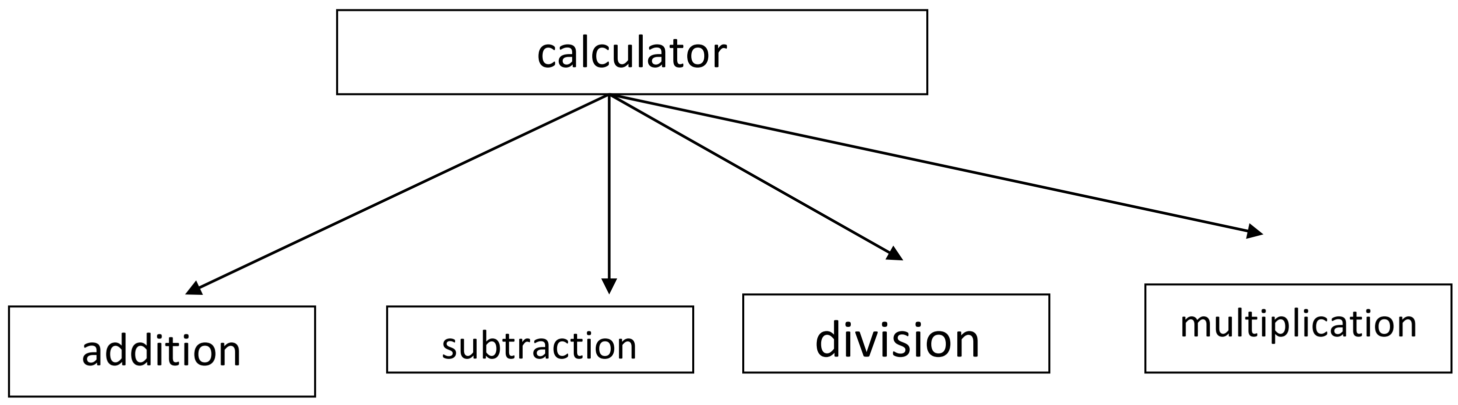
```
x = round(5.76543, 2)
print(x)
```

sum() Function

```
a = (1, 2, 3, 4, 5)
x = sum(a)
print(x)
```

MODULES:

- Module refer to a file containing python statements and definition
- We use modules to break down large program into small mangeable and organized files.
- Modules provide reusablility of code
- Module to store most used user defined functions and call it without rewrite



Module and packages

Eg:1

```
import platform
```

Eg:2

```
dir(platform)
```

Eg:3

```
platform.processor()
```

Eg:4

```
from platform import*
```

Eg:5

```
processor()
```

Eg:6

```
from platform import os,processor,machine
```

Eg:7

```
os
processor
machine
```

Eg:8

```
 import platform as p
p.processor()
```

Eg:9

<u>USER DEFINED MODULE:</u>

Eg1:

<u>module file: arith.py</u>

```
def add(x,y):
	return x+y
def sub(x,y):
	return x-y
def mul(x,y):
	return x*y
def div(x,y):
    return x/y
```

Another file:

```
import arith
print(arith.add(22,33))
print(arith.sub(22,33))
print(arith.mul(22,33))
print(arith.div(22,33))
```

PREDEFINED MODULE:

Decimal Module:

- *to handle decimal numbers*

Eg:

```
import decimal
>>> print(decimal.Decimal(0.5))
0.5
>>> print(decimal.Decimal(0.1))
0.1000000000000000055511151231257827021181583404
5410
```

Eg:2

```
>>> from decimal import Decimal as D
>>> print(D(1.2)+D(2.2))
```

```
3.4000000000000133226762955
```

Fraction Module:

Eg:1

```
import fractions
print(fractions.Fraction(1.5))
print(fractions.Fraction(5))
print(fractions.Fraction(1,3))
```

math module:

Eg:1

```
import math
print("PI:", math.pi)
```

Eg:2

```
import math as m    #import with renaming
print("PI:", m.pi)
```

Eg3:

We can import specific name from a module without importing the whole module.

```
From math import pi
Print("The value of pi is",pi)
```

Eg:4

```
import math
```

```
print(math.cos(math.pi))
print(math.exp(10))
print(math.log10(1000))
print(math.sinh(1))
print(math.factorial(6))
```

random module:

Eg:1

```
import random
print(random.randrange(10,20))
x=['a','b','c','d']
print(random.choice(x))
random.shuffle(x)
print(x)
print(random.random())
```

- Python is an object oriented programming language.
- collection of variables & methods(function)

Create a Class

To create a class, use the keyword class:

Eg:1 Empty class

```
class demo:
	pass
```

Eg:2

Accessing members

```
class sample:
    x,y=10,20



s=sample()
print("value of x",s.x)
print("value of y",s.y)
print("value of ",s.x+s.y)
```

Eg:3

```
class add:
        def get(self):
                self.a=int(input("enter a number"))
                self.b=int(input("enter a number"))
        def cals(self):
                self.c=self.a+self.b
        def show(self):
                print("Add:" , self.c)

a1=add()
a1.get()
a1.cals()
a1.show()
```

Eg:3

```
class std:
    m1,m2,m3=98,97,100
    def process(self):
        self.sum=self.m1+self.m2+self.m3
        self.avg=sum/3
        print(self.sum)
        print(self.avg)
```

```
s=std()
s.process()
```

Eg:4

```
class vehicle:
    name="BMW"
    kind="CAR"
    color="white"
    value=100.00
    def description(self):
        car="%s is a %s %s worth
$%.2f."%(self.name,self.color,self.kind,self.value)
        return car

car1=vehicle()
print(car1.description())
```

Constructor

Eg:1

```
class Person:
    def __init__(self, name):
        self.name = name
    def say_hi(self):
        print('Hello, my name is', self.name)

p = Person('santhosh')
p.say_hi()
```

Eg:2

```
class add:
        def __init__(self,x=10,y=10):
                self.a=x
                self.b=y
        def cals(self):
                self.c=self.a+self.b
        def show(self):
                print("Add:" , self.c)
a1=add()
a1.cals()
a1.show()

a2=add(22)
a2.cals()
a2.show()
```

```
a3=add(22,33)
a3.cals()
a3.show()
```

Eg:3

```
class employee:
    def __init__(self,name,id):
        self.id=id
        self.name=name
    def display(self):
        print("Id:%d \nname:%s"%(self.id,self.name))

emp1=employee("john",101)
emp2=employee("david",102)
emp1.display()
emp2.display()
```

Eg:4

Default constructor

```
class std:
    roll=101
    name="thilo"
    def __init__(self):
        print(self.roll,self.name)
st=std()
```

to delete object

Eg:1

```
class example:
    def __init__(self):
        print("object created")
    def __del__(self):
        print("object destroyed")

a=example()
del a
```

Eg:2

```
class add:
        def get(self):
                self.a=int(input("enter a number"))
                self.b=int(input("enter a number"))
```

```
    def cals(self):
        self.c=self.a+self.b
    def show(self):
        print("Add:" , self.c)

a1=add()
a1.get()
a1.cals()
a1.show()
del a1
a1.show() #error
```

Built In Class Function

getattr(obj,name,default)

It is used to access the attribute of the object.

setattr(obj,name,value)

It is used to set a particular value to the specific attribute of an object.

delattr(obj,name)

Delete the specific attribute.

hasattr(obj,name)

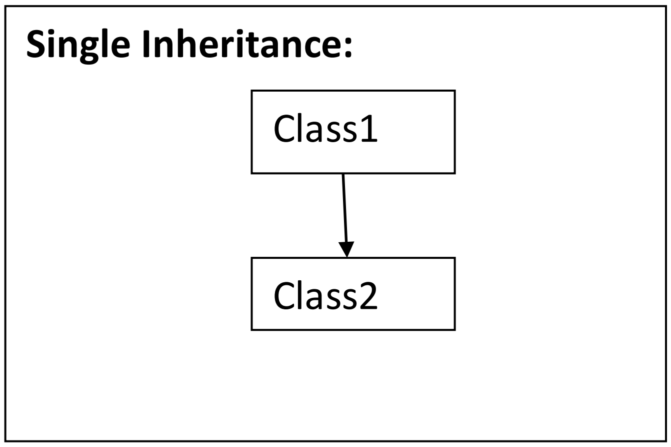
It returns true if the object contains some specific attribute.

Eg:1

```
class student:
    def __init__(self,name,id,age):
        self.name=name
        self.age=age
        self.id=id
s=student("raj",101,25)
print(getattr(s,"name"))
setattr(s,"age",27)
print(getattr(s,"age"))
print(hasattr(s,"id"))
delattr(s,"age")
print(s.age)
```

INHERITANCE

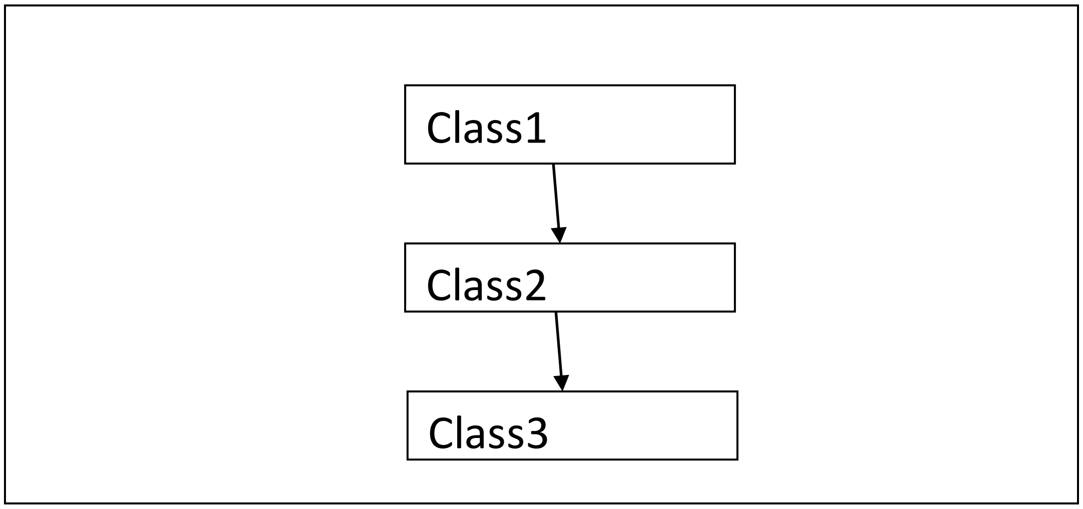
- Inheritance allows us to define a class that inherits all the methods and properties from another class.
- **Parent class** is the class being inherited from, also called base class.
- **Child class** is the class that inherits from another class, also called derived class.



Eg:14 Single Inheritance:

```
    class person:
        def __init__(self,no,name):
            self.Empno=no
            self.Ename=name
        def dispdata(self):
            print("Employee no:",self.Empno)
            print("Employee name:",self.Ename)
    class Employee(person):
        def callme(selfself):
            print("I can use the attribute of  person")

per=person(101,"Ramkumar")
emp=Employee(102,"Suresh")
print("Employee details")
print("_"*25)
per.dispdata()
emp.dispdata()
```

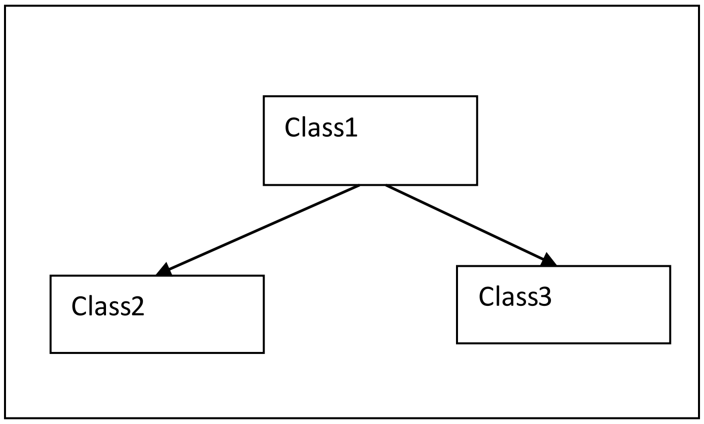
Multilevel Inheritance:



```
class addition:
    def gets(self):
            self.a=int(input("Enter a Number: "))
            self.b=int(input("Enter a Number: "))
    def calladd(self):
             self.c=self.a+self.b
    def disadd(self):
              print("Add: ",self.c)

class subtraction(addition):
    def callsub(self):
```

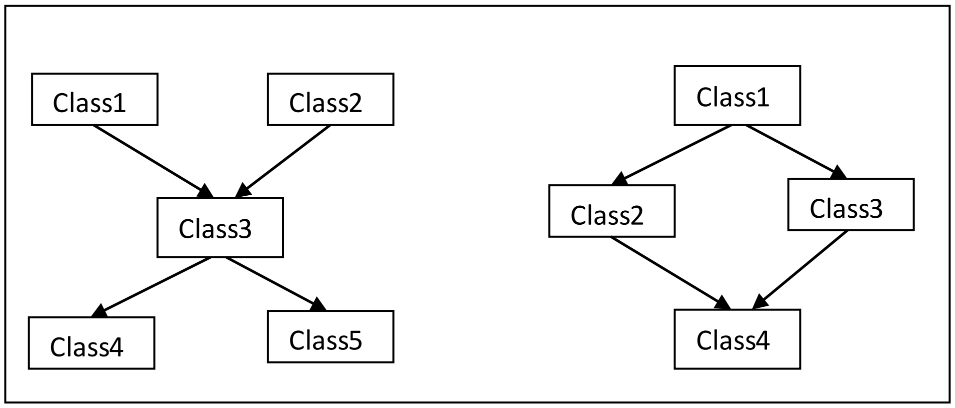
```
                self.d=self.a-self.b
    def dissub(self):
                print("Sub: ",self.d)

class multiplication(subtraction):
    def callmul(self):
                self.e=self.a*self.b
    def dismul(self):
                print("Mul: ",self.e)


a1=multiplication()
a1.gets()
a1.calladd()
a1.disadd()
a1.callsub()
a1.dissub()
a1.callmul()
a1.dismul()
```

Hierarchical Inheritance:



```
class addition:
    def gets(self):
            self.a=int(input("Enter a Number: "))
            self.b=int(input("Enter a Number: "))
    def calladd(self):
             self.c=self.a+self.b
    def disadd(self):
             print("Add: ",self.c)
class subtraction(addition):
    def callsub(self):
        self.d=self.a-self.b
    def dissub(self):
        print("Sub: ",self.d)
class multiplication(addition):
    def callmul(self):
        self.e=self.a*self.b
    def dismul(self):
        print("Mul: ",self.e)


a1=subtraction()
a1.gets()
a1.calladd()
a1.disadd()
a1.callsub()
a1.dissub()

a2=multiplication()
a2.gets()
a2.calladd()
a2.disadd()
a2.callmul()
a2.dismul()
```

Hybrid Inheritance:

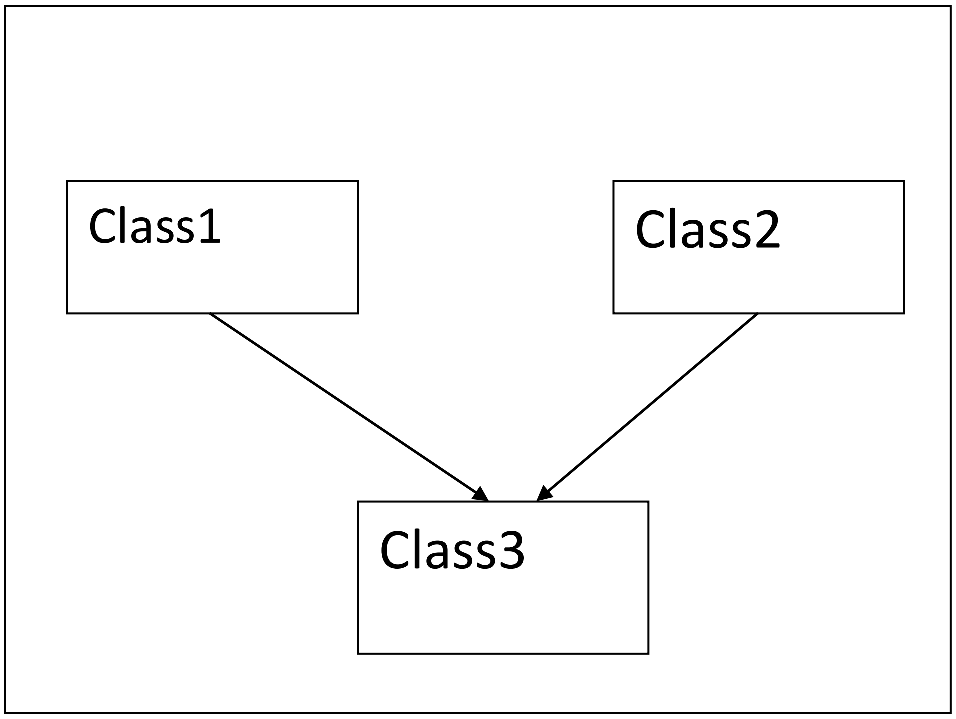


```
class addition:
    def gets(self):
            self.a=int(input("Enter a Number: "))
            self.b=int(input("Enter a Number: "))
    def calladd(self):
             self.c=self.a+self.b
    def disadd(self):
              print("Add: ",self.c)
```

```
class subtraction(addition):
    def callsub(self):
            self.d=self.a-self.b
    def dissub(self):
            print("Sub: ",self.d)


class multiplication(addition):
    def callmul(self):
            self.e=self.a*self.b
    def dismul(self):
            print("Mul: ",self.e)


class division(subtraction,multiplication):
    def calldiv(self):
            self.f=self.a//self.b
    def disdiv(self):
            print("Div: ",self.f)
```

```
a1=division()
a1.gets()
a1.calladd()
a1.disadd()
a1.callsub()
a1.dissub()
a1.callmul()
a1.dismul()
a1.calldiv()
a1.disdiv()
```

Multiple Inheritance:



```
class student:
        tot=0;avg=0.0;res=" "
        def __init__(self,no,na):
                self.stdno=no;self.stdname=na
        def stdfun(self):
                print(“student no:”,self.stdno,”\nstudent
                name:”,self.stdname)

class test1:
        def __init__(self,m1,m2):
                self.p1=m1;self.p2=m2
        def markdisp1(self):
                print(“c language:”,self.p1)
                print(“c++ language:”,self.p2)

class test2:
        def __init__(self,m3,m4)
                sef.p3=m3;self.p4=m4
                def markdisp2(self):
                        print(“java:”,self.p3)
                        print(“python:”,self.p4)

class result(student,test1,test2):
        def __init(self,no,na,m1,m2,m3,m4):
                student.__init__(self,no,na)
                test1.__init__(self,m1,m2)
                test2.__init__(self,m3,m4)
```

```
    def calfun(self):
        self.tot=self.p1+self.p2+self.p3+self.4
        self.avg=self.tot/4
        if self.p1>=40 and self.p2>=40 and self.p3>=40
        and self.p4>=40:
            self.res=”pass”
        else:
            self.res=”fail”

    def resdisplay(self):
        print(“total:”,self.tot)
        print(“average:”,self.avg)
        print(“result:”,self.res)

ob=result(101,”raj”,67,78,89,90)
ob.calfun()
print(“_”*25)
ob.stdfun()
print(“_”*25)
ob.markdisp1()
ob.markdisp2()
print(“_”*25)
ob.resdisplay()
```

```
print(“_”*25)
```

Exercise 1:

1.Create a Bus Child class that inherits from vehicle class. The default fare charge of any vehicle is seating capacity * 100.
If vehicle is Bus instance , we need to add an extra 10% on full fare as a maintence charge . So, final amt=total fare + 10% of total fare.

Output:

```
Total seating capacity =50
Final fare amount =5500
```

2.student (parent) --- > name,age ,gender
  Test(child 1) ---- > get five sub marks
  mark (child2) ----- > display Nam,e age ,gender   and total sub mark

Exercise: 2

Assume that a bank maintain two kinds of account for customers, one called as savings account and the other as current account. The savings account provides compound interest and withdrawal facilities but no cheque book facility. The current account provides cheque book facility but no interest. Current account holders should also maintain a minimum balance and if the balance falls below this level, a service charge is imposed.

Create a class **account** that stores customer name, account number and type of account. From this derive the classes **cur _acct** and **sav _acct** to make them more specific to their requirement. Include necessary member function in order to achieve the following tasks:

a) Accept deposit from a customer and update the balance
b) Display the balance
c) Permit withdrawal and update the balance

EXCEPTION HANDLING

Python Try Except

- The try block lets you test a block of code for Errors.
- The except block lets you handle the error.
- The finally block lets you execute code, regardless of the result of the try- and except blocks.

Program-1

```
print(x)
c=2+3
print(c)
```

Solution:

```
try:
    print(x)
except:
    print("error occur")
```

```
finally:
    c=2+3
    print(c)
```

Index Error

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

IndexError: list index out of range.

Name Error:

NameError: name 'b' is not defined.

```
try:
  print(x)
except NameError:
  print("Variable x is not defined")
except:
  print("Something else went wrong")
```

Finally

The finally block, if specified, will be executed regardless if the try block raises an error or not.

```
try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```

program-1

```
a=int(input("enter a number"))
b=int(input("enter a number"))
try:
    c=a//b
    print("c:",c)
except ZeroDivisionError:
    print("Cant divide a number with Zero")
print("END")
```

program-2

```
try:
    a,b=eval(input("Enter two numbers sep. by camma"))
    c=a+b
    print("C:",c)
except SyntaxError:
    print("Numbers must be seperated by comma")
print("This is the line")
```

program-3

```
try:
    a,b=eval(input("enter 2 nos seperated by comma: "))
    c=a//b
    print("c:",c)
except ZeroDivisionError:
    print("Cant divide a number with Zero")
except SyntaxError:
    print("Comma is missing in input values")
```

with except,else,finally block:

program-4

```
try:
    a,b=eval(input("enter 2 nos seperated by comma: "))
    c=a/b
    print("c:",c)
except ZeroDivisionError:
    print("Cant divide a number with Zero")
except SyntaxError:
    print("Comma is missing in input values")
except:
    print("Wrong Input")
else:
    print("No Error")
finally:
    print("Code Executed successfully")
```

User defined Exception:

Program-1

```
class VotingAgeError(Exception):
    pass


print("Voting age validation Started...")
age=int(input("Enter Your age:"))
try:
    if (age<18):
        raise VotingAgeError
    else:
        print("Your are eligible to vote")
except VotingAgeError:
    print("Your are not eligible to vote")


print("Voting age validation Finished")
```

program-2

```
class NegativeValueError(Exception):
    pass
class ZeroValueError(Exception):
    pass


while True:
```

```
try:
    n=int(input("Enter a number: "))
    if n==0:
        raise ZeroValueError
    if n<0:
        raise NegativeValueError
    else:
        print("Valid Input")
        print("The entered number is ", n)
        break;
except NegativeValueError:
    print("Number should not be negative")
except ZeroValueError:
    print("Number should not be Zero")
```

Exercise:

output of your program:

```
enter  number: 123
enter name: vijayan
enter age: (condition→  greater than18)
```

enter sal: (condition→ 2000 to 20000)

number:123

name:vijayan

age: (print “invalid age” if age is less than 18 , else print the age)

sal: (print “invalid salary” if salary is less than 2000 or greater than 20,000 , else print the salary)

Module Package:

```
import math
num=int(input("please enter the number to calculate factorial of:"))
try:
    result=math.factorial(num)
    print(result)
except:
    print("negative number")
```

Exception keyword

```
try:
    print(a)
```

```
except Exception as e:
    print(e)
```

FILE HANDLING

The key function for working with files in Python is the open() function.

The open() function takes two parameters; *filename*, and *mode*.

Syntax:

Open syntax:

```
Var_name=open("file name",mode)
```

Close syntax:

```
Var_name=close()
```

Program-1

```
f=open("wel.txt",'w')
f.close()
```

Example:

```
f=open("wel.txt",'w')
f.write("first line")
f.close()
```

Example:

Print one line statement

```
f=open("wel.txt",'w')
a=input("enter a string:")
f.write(a)
f.close()
```

Example:

Print multi line statement

```
f=open("wel.txt",'w')
for i in range(3):
    a=input("Enter a sring:")
    f.write(a)
f.close()
```

program2:

Read file

```
f = open("wel.txt", "r")
print(f.read())
```

<u>Read Only Parts of the File</u>

By default the read() method returns the whole text, but you can also specify how many characters you want to

Program-3

```
f = open("wel.txt", "r")
```

```
print(f.read(5))
```

Read Lines

You can return one line by using the readline() method:

Program-4

```
f = open("wel.txt", "r")
print(f.readline())
```

Read two lines of the file:

Program-5

```
f = open("wel.txt", "r")
print(f.readline())
print(f.readline())
```

for loop

By looping through the lines of the file, you can read the whole file, line by line:

Program-6

```
f = open("wel.txt", "r")
for x in f:
  print(x)
```

Delete a File

To delete a file, you must import the OS module, and run its os.remove() function:

```
import os
os.remove("wel.txt")
```

PICKLE MODULE

Pickle module used to convert object into binary (0’s and 1’s).

Write binary

```
import pickle
cars=["audi","benz","bmw"]
file="aaa.txt"
fileobject=open(file,"wb")
pickle.dump(cars,fileobject)
fileobject.close()
```

read binary

```
import pickle
file="aaa.txt"
fileobject=open(file,"rb")
print(pickle.load(fileobject))
fileobject.close()
```

METHOD OVERLOADING

- **Method Overloading** is the class having **methods** that are the same name with different arguments.
- Arguments different will be based on a number of arguments and types of arguments.
- It is used in a single class.
- It is also used to write the code clarity as well as reduce complexity.

Program-1

Error program

Passing one arguement

```
class sample:
        def second(self,a,b):
         c=a+b
         return c
a=sample()
print(a.second(10))
```

program-2

how to clear this error:

```
class demo:
    def sum(self,a=None,b=None,c=None):
        if a!=None and b!=None and c!=None:
            c=a+b+c
        elif a!=None and b!=None:
            c=a+b
        else:
            c=a
        print(c)
obj=demo()
obj.sum(10,20,30)
obj.sum(10,20)
obj.sum(10)
```

Exercise:

Area of rectangle and area of square using method overloading

rectangle=a*b

square=a*a

METHOD OVERRIDING

Declaring a method in sub class which is already present in parent class is known as method **overriding**. **Overriding** is done so that a child class can give its own implementation to a method which is already provided by the parent class.

Program-1

```
class father:
    def show(self):
        print("i am father")
class son(father):
    pass

a=son()
a.show()
```

program-2

```
class father:
    def show(self):
        print("i am father")
class son(father):
    def show(self):
        print("i am son")

a=son()
a.show()
```

OPERATOR OVERLOADING

Python operators work for built-in classes. But the same operator behaves differently with different types. For example, the + operator will perform arithmetic addition on two numbers, merge two lists, or concatenate two strings.

Program-1

```
a=20
b=10
```

```
print(a+b)
```

program-2

```
a=10
b=20
print(int.__add__(a,b))
```

(or)

```
a=10
b=20
print(int.__sub__(a,b))
```

(or)

```
a=10
b=20
print(int.__mul__(a,b))
```

(or)

```
a=10
```

```
b=20
print(int.__truediv__(a,b))
```

program-3

```
class student:
    def __init__(self,m1,m2):
        self.m1=m1
        self.m2=m2
        print(m1+m2)
    def __add__(self,others):
        m1=self.m1+others.m1
        m2=self.m2+others.m2
        s3=student(m1,m2)
s1=student(10,20)
s2=student(20,30)
s3=s1+s2
```

program-3

```
class book:
    def __init__(self,price):
        self.price=price
```

```
b1=book(10)
b2=book(20)
b3=b1.price+b2.price
print(b3)
```

program-4

```
class book:
    def __init__(self,price):
        self.price=price
    def __add__(self,other):
        return self.price+other.price

b1=book(10)
b2=book(20)
b3=b1+b2
print(b3)
```

<u>other operator overloading methods:</u>

```
__sub__()
__mul__()
```

__pow__()

__truediv__()

__floordiv__()

__mod__()

POLYMORPHISM

Polymorphism is a very important concept in programming. It refers to the use of a single type entity (method, operator or object) to represent different types in different scenarios.

Program-1

```
class version1:
    def button(self):
        print("colour red")
class version2(version1):
    def button(self):
        print("colour yellow")
```

```
a=version2()
a.button()
```

program-3

Polymorphism with class methods:

```
class India:
    def capital(self):
        print("New Delhi is the capital of India.")

    def language(self):
        print("Hindi is the most widely spoken language of India.")

    def type(self):
        print("India is a developing country.")

class USA:
    def capital(self):
        print("Washington, D.C. is the capital of USA.")
    def language(self):
        print("English is the primary language     of USA.")

    def type(self):
        print("USA is a developed country.")

obj_ind = India()
obj_usa = USA()
obj_ind.capital()
obj_ind.language()
obj_ind.type()
(or)
for country in (obj_ind, obj_usa):
    country.capital()
    country.language()
    country.type()
```

MULTITHREADING

Program-1

```
class hello:
    def run(self):
        for i in range(5):
            print("hello")
class hai:
    def run(self):
        for i in range(5):
            print("hai")
t1=hello()
t2=hai()
t1.run()
t2.run()
```

Program-2

```
from threading import*
class hello(Thread):
    def run(self):
        for i in range(5):
            print("hello")
class hai(Thread):
    def run(self):
        for i in range(5):
            print("hai")
t1=hello()
t2=hai()
t1.start()
t2.start()
```

program-3

time setting

```
from time import sleep
from threading import*
class hello(Thread):
    def run(self):
        for i in range(5):
            print("hello")
            sleep(1)
class hai(Thread):
    def run(self):
        for i in range(5):
            print("hai")
            sleep(1)
t1=hello()
t2=hai()

t1.start()  #after start() method thread #convert as three
thread main thread , #thread1 and thread2
sleep(0.2)
t2.start() # run under thread 2

t1.join()
t2.join()

print("bye")  # it will execute under main #thread
```

DATE TIME MODULE

Program-1

```
import datetime
x=datetime.datetime.now()
print(x)
```

program-2

```
import datetime

alarmHour=int(input("what hour do you want to wake up"))
alarmMinute=int(input("what minute do you want to wake
up"))

amPm=str(input("am or pm"))

if (amPm=="pm"):
    alarmHour=alarmHour+12

while(1==1):
    if(alarmHour==datetime.datetime.now().hour and
alarmMinute==datetime.datetime.now().minute):
```

```
print("wake up lazy")
        break
print("existed")
```

ABSTRACT CLASS IN PYTHON

- *An abstract class can be considered as a blueprint for other classes. It allows you to create a set of methods that must be created within any child classes built from the abstract class.*
- *A class which contains one or more abstract methods is called an abstract class.*
- *An abstract method is a method that has a declaration but does not have an implementation.*
- *While we are designing large functional units we use an abstract class. When we want to provide a common*

interface for different implementations of a component, we use an abstract class.

```
from abc import abc, abstractmethod

class polygon(abc):
    @abstractmethod  #decorator
    def noofsides(self):
         pass

class triangle(polygon):
    # overriding abstract method
    def noofsides(self):
        print("i have 3 sides")

class pentagon(polygon):


    # overriding abstract method
    def noofsides(self):
        print("i have 5 sides")

class hexagon(polygon):
    # overriding abstract method
    def noofsides(self):
        print("i have 6 sides")

class quadrilateral(polygon):
    # overriding abstract method
    def noofsides(self):
        print("i have 4 sides")



r = triangle()
r.noofsides()

k = quadrilateral()
k.noofsides()

r = pentagon()
r.noofsides()

k = hexagon()
k.noofsides()
```

DUCK TYPING

Duck typing is a trem commonly related to dynamically typed programming language and polymorphism.

“If it look like duck and quacks like duck ,it’s a duck”

```
class pycharm:
    def execute(self):
        print(“compling”)
        print(“running”)

class idle:
    def execute(self):
        print(“spell check”)
        print(“convention check”)

class laptop:
    def code (self,ide):
        ide.execute()

ide=idle()      #pycharm()
```

```
lap=laptop()
lap.code(ide)
```

SEARCHING ALGORITHM

Linear Search:

```
pos=1            #Global variable
def  Search(list,n):
    i=0
    while i<len(list):
        if list[i]==n:
            global()['pos']=i
            return true
        i=i+1
    return false

list=[5,8,4,6,9,2]
```

```
n=9
if search(list,n):
        print("Found at",pos+1)
else:
        print("Not Found")
```

Exercise:use for loop instead of while loop.

Binary Search:

- *If list contain about 1000 value its difficult takes more time.*
- *In Binary search we have to sort the list.*
- *If search value lower than mid change mid to upper.*
- *If search value is higher than mid change mid to lower bound.*

```
pos=-1
def search(list,n):
l=0
```

```
u=len(list)-1
while l<=u:
    mid=(l+u) //2
    if list [mid]==n:
        globals()[‘pos’]=mid
        return true
    else:
        if list[mid]<n:
            l=mid+1
        else:
            u=mid-1
return false

list=[4,7,8,12,45,99]
n=45
if search(list,n):
    print(“Found at”,pos+1)
else:
    print(“Not found”)
```

Bubble Sort:

```
def sort (nums):
    for i in range (len(nums)-1,0,-1):
        for j in range(i):
            if num[j]>num[j+1]:
                temp=nums[j]
                nums[j]=nums[j+1]
                nums[j+1]=temp

nums=[5,3,8,6,7,2]
sort(nums)
print(nums)
```

<u>Selection Sort:</u>

```
def sort (nums):
    for i in range (5):
        minpos=i
        for j in range (i,6):
            if nums[j]<nums[minpos]:
                minpos=j
        temp=nums[i]
        nums[i]=nums[minpos]
        nums[minpos]=temp
    print(nums)
```

```
nums=[5,3,8,6,7,2]
sort(nums)
```

MySql Connectivity

Insert data into MySql:

```
import mysql.connector

connection=mysql.connector.connect(host=”localhost”,user
=”root”,password=”1234”,database=”coderz”)
cursor=connection.cursor()
insert1=”insert into student(roll,name)values(101,’raj’);”
cursor.execute(insert1)
connection.commit()
connection.close()



# insert multiple data.

insert 2=”insert into student(roll,name)value(102,’ram’);”
insert 3=”insert into student(roll,name)values(103,’hari’);”
cursor.execute(insert 2)
cursor.execute(insert 3)
```

Retrive data from MySql:

```
import mysql.connector
connection=mysql.connector.connect(host=”local
host”,user=”root”,password=”1234”,database=”coderz”)
cursor=connection.cursor()
retrieve=”select*from coderz;”
cursor.execute(retrive)
rows=cursor.fetchall()
for row in rows:
        print(row)
connection.commit()
connection.close()
```

Update data in MySql:

```
import mysql.conncetor

connection=mysql.connector.connect(host=”local
host”,user=”root”,password=”1234”,database=”coderz”)
cursor=connection.cursor()
```

```
updatesql="update student set name='raj'where roll='104';"
cursor.execute(updatesql)
retrieve="select*from student;"
cursor.execute(retrive)
rows=cursor.fetchall()
for row in rows:
        print(row)
connection.commit()
connection.close()
```

<u>delete data in MySql</u>

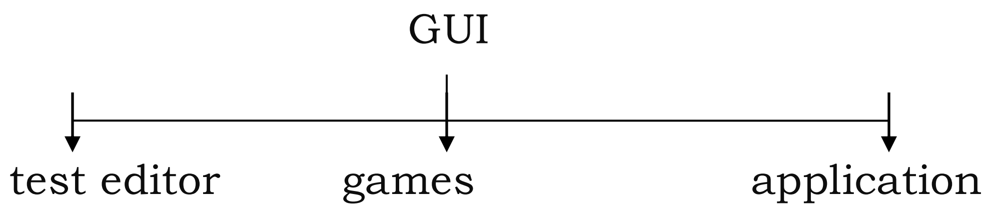
```
import  mysql.connector
connection=mysql.connector.connect(host="local
host",user="root",password="1234",database="coderz")
cursor=connection.cursor()
deletesql="delete freom student where roll='10';"
cursor.execute(deletesql)
retrieve="select*from student;"
cursor.execute(retrive)
rows=cursor.fetchall()
for row in rows:
        print(row)
```

```
connection..commit()
connection.close()
```

Tkinter

What is GUI?

GUI is a desktop app which helps you to interact with computers.



Tkinter(In built modules)

Tkinter in python GUI programming is standard python GUI library it give us an object oriented interface to TK GUI toolkil.

$$\begin{tikzcd}
\text{Import the tkinter module} \arrow[d] \\
\text{Great the GUI application main window} \arrow[d] \\
\text{Add widgets} \arrow[d] \\
\text{Enter the main event loop.(it will tell close window only by manually)}
\end{tikzcd}$$

```
import tkinter
from tkinter import *
window=tkinter.Tk()
#to rename the title of the window
window . title("GUI")
#pack is used to show the object in the window.
label=tkinter.Label(window,text="Hello World").pack()
window.mainloop()
```

Tkinter Widget

A widget is an element of a graphical user interface(GUI) that displays information or provide a specific way for a user to interact with the OS or an application.

- *Label*
- *Button*
- *Entry*
- *Radio*
- *Checkbox*
- *Combo box*
- *Scrolled text*
- *Spin box*
- *Menu Bar*
- *Note book*

1. Label:

Label:You can set the label font so you can make it bigger and may be bold.

```
from tkinter import *
```

```
window=Tk()
l1=Label(Window,text="Helloworld" ,font=("Arial bold",50))
l1.grid(column=0,row=0)
window.mainloop()
```

- *Using label we can set the default window six using geomentry function.*

```
l1=label(window,text="Helloworld"font=("Arial bold",50))
window.geometry('350*200')
l1.grid(column=0,row=0)
```

2. ***Button widgets:***

Lets start by adding the button to the

button to the window,the button is created and added to window the same as the label.

```
bt=Button(window,text="Enter")
bt.grid(column=1,row=0)
```

We can change foreground to background colour of button using fg & bg properly.

```
bt=button(window,text=”Enter”,bg=”Orange”,fg=”red”)
 bt.grid(column=1,row=0)
 def clicked()
        l1.configure(text=”Button was clicked!!”)
 bt=Button(Window,text=”Enter”,command=clicked)
```

Writing the button with the function.

- *Here try getting the user input using tkinter entry class (Tkinter textbox)*

```
 from tkinter import*
 window=TK()
 txt=Entry(window,width=10)
 txt.grid(column=1,row=0)
 l1=label(Window,text=” “)
 def clicked():
        res=”Welcome to”*txt.get()
        l1.configure(text=res)
 bt=button(window,text=”Enter”,command=clicked)
```

```
bt.grid(col=1,row=0)
l1.grid(column=3,row-0)
window.mainloop()
```

<u>Combo Box Widget</u>: (Dropdowm)

```
import tkinter as tk
from tkinter import.ttk
ttk.window=tk()
combo=combobox(Window)
combo['values']=(1,2,3,4,5,"Text")
            #Adding the combobox items using tuple
combo.current(3)
combo.grid(column=0,row=0)
window.mainloop()
```

<u>Check Button Widget:</u>

```
from tkinter import *
window=tk()
```

#creating a variable of type Boolean var which is not a standard python variable,it's a tkinter variable.

```
Chk_state=Boolean var()
Chk_state.set(true)
chk=checkbutton(window,text=‘select’,var=chk-state)
        #passing the chk state to check button class to set the
check state.
chk.grid(column=0,row=0)
chk.pack()
window.mainloop()
```

Radio Button Widgets:

Use Radio button class

```
from tkinter impor *
window=Tk()
rad1=Radiobutton(Window,text=’Python’,value=1)
rad2=radiobutton(window,text=’java’,value=2)
rad3=radiobutton(window,text=’scala’,value=3)
rad1.grid(column=0,row=0)
rad2.grid(column=1,row=0)
rad3.grid(column=2,row=0)
window.mainloop()
```

<u>Scrolled Text:</u>

```
imported tkinter as tk
from tkinter import ttk
from tkinter import scrolled text

window=tk.TK()
txt=scrolled text.Scrolled
text(Window,wrap=tk.WORD,width=40,height=10)
txt.grid(column=0,pady=10,padx=10)
txt.focus()   #blink cursor position
txt.insert(INSERT,'you text goes here')
window.mainloop()
```

<u>Message Box:</u>

Shows a message box when the user clicks a button.

```
from tkinter import message box
message box.showinfo('Message title','Message content')
```

- *show warning*
- *show error*

- *ask question*
- *ask ok cancel*
- *ask yes no*
- *ask retry cancel*

```
from tkinter import*
from tkinter import messagebox
 window=TK()
 def clicked():
       messagebox.showinfo(‘Message title’,’message content’)
btn=Button(Window,text=’enter’,command=clicked())
```

<u>Spin box widgets:</u>

```
from tkinter import *
window=TK()
          spin=spinbox(Window,from_=0,to=100,width=5)
spin.pack()
window.mainloop()
```

<u>Geometry Management:</u>

All tkinter widgets will have geometric measurement.

Geometry Manager Classes

1. *grid()-It organizes the widgets in table like structure.*
2. *pack()=It organize the widget in the block,which mean it occupies the entire available widget.*
3. *place()-Its used to place he widgets at a specific position you want.*

Organizing layouts & Widget

We use frame class to arrange layout in a window.

Frame→ Frame is used to create the division in the window. You can align the frames as you like with side parameter of pack() method.

Button→ Button is used to create a button in the window. It take several parameter like text(value of the button),fg(Color of the text),bg (color of back ground)

```
import tkinter
Window=tkinter.Tk()
window.title("title")
        # Creating 2 frame TOP & BOTTOM
Top_frame=tkinter.frame(window).pack()
bottomframe=tkinter.frame(window).pack(side="bottom")
btn1=tkinter.Buttom(top_frame,text="Button1",fg="red").pa
ck()
```

```
btn2=tkinter.Buttom(top_frame,text=”Button2”,fg=”green”).
pack()
btn3=tkinter.Buttom(bottom_frame,text=”Buttom3”,fg=”pur
ple”).pack(side=”left”)
btn4=tkinter.Buttom(bottom_frame,text=”Button4”,fg=”Ora
nge”).pack(side=”left”)
```

END

THANK YOU

BEST WISHES FOR YOUR FUTURE

CODERZ ACADEMY