- **print(“answer is”, 5/2)   ans →answer is 2.5 (normal division)**
- **print(5//2)   ans→ 2 integer division**
- **2*3 – ans → 6**
- **2**3 is 6 i.e. 2 power of 3**
- **print(10* ‘xyz’) →it will print xyz 10 times**
- **print(r ‘c/hi:hello/navin’) → it will print the raw string i.e what we have written same as that, if we don’t give r before then it will not do the same.**
- **name = ‘wxyz’ <enter> print(name[0]) → ans= ‘w’ print(name[-1]) → ans = ‘z’ i.e. when we give –ve value it starts from right with first index as 1. name[0:2] → ‘wx’ name[1:] → ‘xyz’ name[:2] → ‘wx’ .**
- **To reverse a string in python just use <string name>[::-1]**
- **In python String is immutable.**
- **myname = “Aritra Basak” <enter> len(myname) → ans=12 i.e. returns length of the string.**
- **List can hold different types of data and list is mutable. nums=[23,24.0, ‘hi’ ,56] print(nums[1]) ans → 24.0 print(nums[1:]) ans → 24.0, ‘hi’ ,56 .i.ei it prints all numbers from index 1. print(nums[-1]) ans → 56 , print(nums[-4]) ans → 23. We can create nested list also   list1 = [<list 1 >,<list2>]**
- **<list name>.append (number or anything)→ adds a element at the end of the list.**

  <list name>.insert(index, the number or anything u want to add) → adds anything at the given index.

  <list name>.remove(the item) → simply removes the item from the list.

  <list name>.pop(index) → removes the item from the given index. If we don’t specify the index value in case of pop(), it will remove the last element from the list.

  del <list name>[starting index : ending index] → deletes all the elements in that specified index

  sorted(list(<list name>) → use to sort elements in a list

  <list name>.extend([write all the numbers u want to add]) → adds the given numbers at the end of the list.

  min(<list name>) → gives the minimum value in the list , max(<list name>) → gives the maximum value in the list , sum(<list name>) → gives the sum of all the elements in the list, <list name>.sort() → sort the list in ascending order.
- **Tuple in python is immutable and it uses round brackets e.g. tup=(1,3,5,7,9). tup[1] →3.**

- **Sets in python is mutable and it prints the given elements in random sequences and it will not support duplicate values, rest are same as list.**
- **x=5 print(id(x)) → gives the address of the variable x. Address is not based on the variable but on the data it stores , like x=5 and y=5 then x and y have the same address.**
- **num= 5 type(num) ans→ int. num=5+6j (here j is imaginary part i.e. -1) type(num) ans→ complex.**
- **Type casting → a= 5.66 (a float value) b=int(a) // type casted to int**
- **Dictionary in python, it contains different values and must be represented with keys e.g. d={‘A’: ‘Apple’ , ‘B’: ‘Ball’ , ‘C’: ‘Cat’}. Here A, B and C are the keys for their respective values. If we use d.keys() it will return all the names of the keys , if we use d.values() it will return all the values. d[‘<key name>’] to get the particular value like d[‘A’] → ‘Apple’.**
- **x+=2 → x=x+2 , x*=3 → x=x*3**
- **Here && is and . || is or. Not operation like x= True x=not x ans → False**
- **bin() converts decimal to binary e.g. bin(25) → 0b11001. The first 0b represents that the result is in binary format. oct() converts decimal to octal and in the result the first 2 digits 0o represents that it is in octal format and the hex() converts decimal to hexadecimal and the first 2 digits 0x represents hexadecimal format. And to just convert all of them in decimal again just print them.**
- **Bitwise and → & Bitwise or → |. These checks with binary conversion. Like 12 & 13 → 12, as we are performing and gate conversion with 12 and 13 binary form and the result after conversion is 12.**
- **Left shift 10<<2 means shifting 10’s binary numbers in left by 2 position and for right shift 10>>2 we are shifting the 10’s binary number in right by 2 position. e.g right shift 10>>2 →1010 >> 0010. Left shift 10<<2 → 1010<< 101000 .**
- **We need to import math function to use different mathematics function like math.sqrt(). math.floor(2.5) → 2 math.ceil(2.5) →3**
- **We can also do importing like this, import math as m , so we will use m instead of math to invoke a function everywhere.**
- **In python we don’t use relational operators (== and !=) with float values, cause while comparing the values gets rounded up which differentiates them. E.g. a=1.001 b = 2.001 c =3.303 d=3.302 <enter> if((b-a)==(c-d)): <enter> print(“TRUE”) <enter> else: <enter> print(“FALSE”) → output FALSE.**

- **USER Input → <variable>= input("Enter a number") → will return the value as string and will not perform any mathematical operations. So we use <variable>=<data type>(input("Enter a number"). But for character ort string we don't need to mention the data type.**
- **BUT ITS BEST TO USE eval function for fetching input x= eval(input("Enter a number")) , here with eval we can also take complex number as input even take equations also and print answers directly (don't use eval to take a string input) →**

```
x =eval(input("Enter a number"))
y= eval(input("enter another number"))
print(x+y)
result = eval(input("Enter the expression "))
print(result)
```

```
OUTPUT
Enter a number10
enter another number12
22
Enter the expression 10+4+2-9
7
```

- **Command line arguments in python → import sys <enter> x =int(sys.argv[1]) <enter> print(x). And in command prompt write python <file name>.py <space> value.**
- **In case of conditional statements if we use multiple if statements then if, first condition is satisfied in if then also the prog will move to the rest of if statements and check it Its better to use elif there. Example of conditional →**

```
x =eval(input("Enter a number"))
if x%2==0:
 print("Even")
elif x%2!=0:
 print("Odd")
else:
 print("Error")
```

- **While loop in python**

```
i=5
while i>0:
 print(i)
 i=i-1
```

- **print("hello",end= " ") <enter> print("World") <enter> → hello World // prints the both statements in same line in spite of two different print statements.**

- **For loop in python :**

```
l=[1,2,3,4,5,6]
for i in l:
 print(i)
```

- **For loop with a range :**

```
for i in range(0,11):
  print(i)
```

we can also specify the gap of each iteration for i in range(0,11,2) → the 2 specify the gap of each iteration and to print reverse order just reverse the limits and in gap of each iteration give the –ve form like → for i in range(11,0,-1)

- **We can create class and objects here to use here: →**

class <name>: // use class keyword to define a class

def <method name>(self): // use def keyword to define a method and 'self' is the parameter through which object is passed inside the method. We should use self to refer object inside the method.

print("anything")

<variable name> = <class name>() // creates an object of <variable name> of class <class name>

<object variable name>.<method name>() // to call the method defined inside the user defined class.

```
class Computer:
  def config(self):
    print("Ryzen 3, 8gb , 1TB")
com1 =Computer()
print(type(com1))
com1.config()
OUTPUT
<class '__main__.Computer'>
Ryzen 3, 8gb , 1TB
```

- **If we define a variable inside the __init__ method it is called the instance variable and if we define any variables outside __init__ but inside class then it is called the class variable and it is same for every object.**

- **We can also pass arguments in methods and perform tasks, for that we need to take arguments in a constructor called __init__.**

```
class Computer2:

  def __init__(self,cpu,ram):
    self.cpu=cpu
    self.ram=ram
  def inside(self):
    print("config is",self.cpu,self.ram)
com2=Computer2('Ryzen 5',8)
com2.inside()
OUTPUT
config is Ryzen 5 8
```

- **In python there are at least 3 kinds of methods in Python having different first arguments:**

  Instance method - instance, i.e. self

  Class method - class, i.e. cls

  Static method – nothing

```
class Test():

  def __init__(self):
     pass // as __init__ is a constructor which will take self as an argument but we are not
             performing any task so we are just passing it

  def instance_mthd(self):
     print("Instance method.")

 @classmethod      // this is a decorator we need to write this before a class method else
                    we need to pass cls while calling the method
  def class_mthd(cls):
     print("Class method.")

 @classmethod    // this is a decorator we need to write this before a static method
   def static_mthd():
     print("Static method.")
```

```
class Students:
  school = "JDS"
  def __init__(self,m1,m2,m3,m4):
    self.m1=m1
    self.m2=m2
    self.m3=m3
    self.m4=m4
  def avg(self):
    return (self.m1+self.m2+self.m3 + self.m3)/4
  @classmethod
  def getschool(cls):
    print("The school name is ",cls.school)
  @staticmethod
  def info():
    print("This is a Student class")
s1=Students(65,60,57,59)
s2=Students(46,66,56,69)
s3=Students(67,65,63,67)
Students.getschool()
Students.info()
print("Average of a student in an exam is",s1.avg())
print("Average of a student in an exam is",s2.avg())
print("Average of a student in an exam is",s3.avg())
OUTPUT
The school name is  JDS
This is a Student class
Average of a student in an exam is 59.75
Average of a student in an exam is 56.0
Average of a student in an exam is 64.5
```

- **In python we can create nested class and the object of that class must be created inside the outer class inside the __init__ method like self.<variable>=self.<inner class name>(), or we can call it outside of any class like we are calling the outer class <object>.<variable name>=<object>.<inner class name>, to call it we will use <object>.<variable for inner class>.<that method which we want to call> like→**

```
class Student:
  def __init__(self,name, roll):
    self.name=name
    self.roll=roll
  def show(self):
    print(self.name,self.roll)
  class laptop:
    def __init__(self,brand,cpu):
      self.brand=brand
      self.cpu=cpu
    def show(self):
      print(self.brand,self.cpu)
s1=Student('Aritra',80)
s1.show()
s1.lap=s1.laptop('Asus','Ryzen5')
s1.lap.show()
     OUTPUT →Aritra 80
             HP ryzen5
```

- **In python we can perform inheritance just to inherit ant properties from any parent class write the class name with the child name like class B(A): (inheriting A's features in B). Inheritance here are of 3 types → Single-level (where one class is inhering a single class), Multi-level(where one class is inheriting a class which have already inherited another class), Multiple(where one class is inheriting two different classes and those 2 classes which are being inherited have no relation) like→**

```
class A():
  def feature1(self):
    print("From class A ")
class B(A): //single-level
  def feature2(self):
    print("from class B")
class C():
  def feature3(self):
    print("from class C")
class D(C,A): //Multiple
  def feature4(self):
    print("from class D")
class E(B): //multi-level
  def feature5(self):
    print("from class E")
a=A()
a.feature1()
b=B()
b.feature1()
b.feature2()
c=C()
c.feature3()
d=D()
d.feature1()
d.feature3()
d.feature4()
e=E()
e.feature1()
e.feature2()
e.feature5()
Output
```

```
From class A
From class A
from class B
from class C
From class A
from class C
from class D
From class A
from class B
from class E
```

- **With super keyword we can call any methods from super class while we created object of sub class only, →super.<method from A>().**
- **In python one example of polymorphism is duck typing**
- **In python we use <variable>= ‘’.join(reversed(<string input>)) to reverse a string**