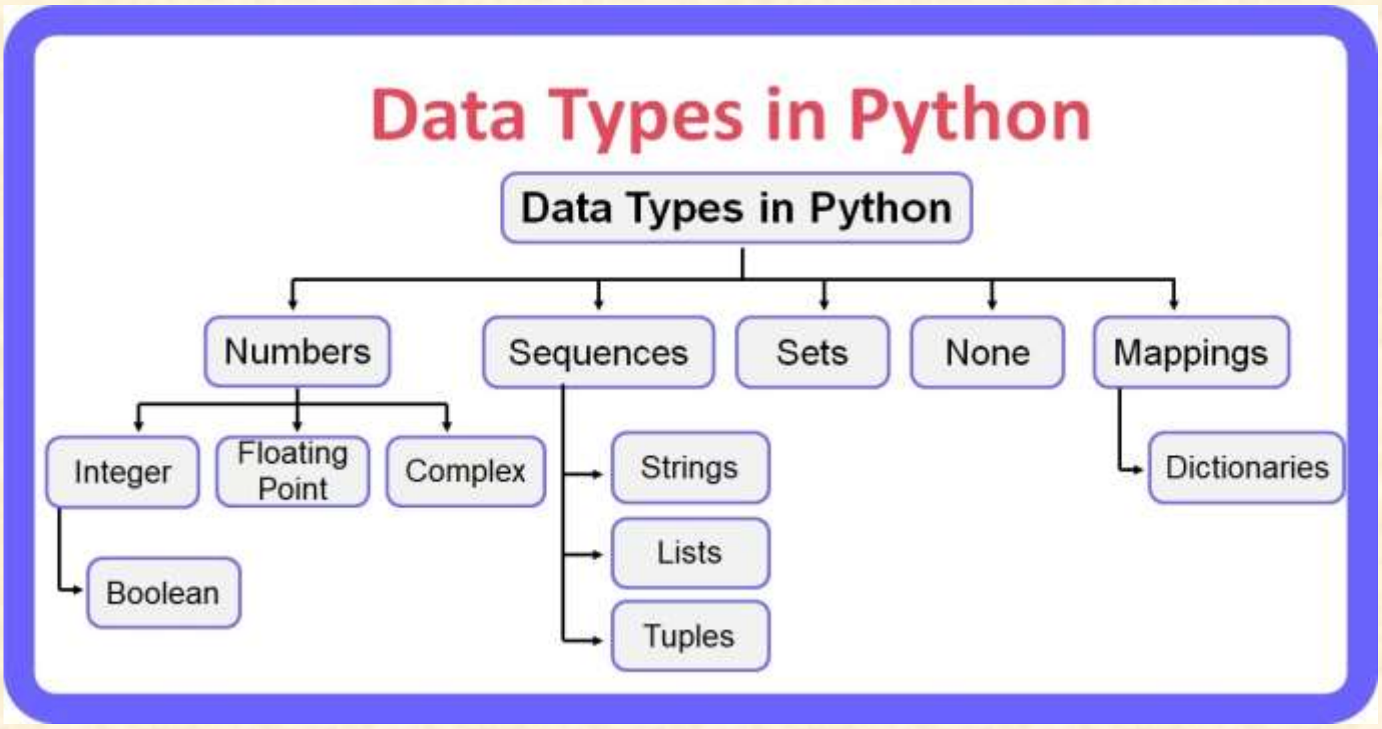
- **Python**: python is popular programming language. It is general-purpose high-level language. Developed by Guidown  Rassum made in 1989 and released in 1991. Python is an interpreted, object – oriented. Python is a dynamic semantics. It is used to web development and software development mathematics. Python is a server-side language.
- **Print ().**
- The python print () function helps in printing the specified message onto the screen, or any other standard output device.
- **Modules.**
- A module is a file containing code written by somebody else (usually) which can be import and used in our programs.
- **Type of modules.**

1. Buit in modules = Pre installed in python. Examples = os, apc etc.
2. External modules = Need to install using pip. Examples = Tensorflow, flask etc.

- **Pip.**
- Pip is the package manager for python you can use pip to install a module on your system.
- **Using python as a calculator.**
- We can use python as a calculator by typing “python + on the terminal.
- This is open REPL or Read evaluate Print loop.
- **Comments.**
- Comments are used to write something which the programmer does not went to execute.
- Can be used to mark author name date etc.
- Types of comments.
  1. Single line comments = written using # pound.
  2. Multi line comments = written using \\\ comment \\\.
- Playsound.
- from playsound import playsound
- playsound('D:\\Python\\to\\a\\sound\\file\\you\\want\\to\\play.mp3')

## Data Type.

- **Data Type** :- data types are the classification or categorization of data items. It represents the type of data present inside a variable.
- **Primary data types**.
  **1.** Integer.
  **2.** Float.
  **3.** Complex.
  **4.** Boolean.
  **5.** String.
  **6.** Double.
  **7.** Character.
- **Inbuild data type.**
  1. Start.
  2. Bytes.
  3. Range.
  4. List.
  5. Tuple.
  6. Set.
  7. Dictionary.
  8. None.



# Variable.

- **Variables:**- Variable is the name of a memory location which stores some data.

1. Variables are case sensitive.
   Upper case ‘A’ and lower case ‘a’ letter.
2. 1st character is alphabet or '_' .
   Only _age or a character.
3. no comma/blank space.
   Only use for underscore.
4. No other symbol other than '.

- **Python code.py**

```
a = "pawan"
b = 345
c = 45.32
print(a)
print(b)
print(c)
print(type(a))
```

**output.**

```
pawan
345
45.32
Class str
```

- **Python code.py**

```
a = input("enter the name")
h = int(input("enter the hindi marks"))
e = int(input("enter the english marks"))
m = int(input("enter the maths marks"))
s = int(input("enter the science marks"))
sst = int(input("enter the social science marks"))
sum = h+e+m+s+sst;
print("totoal sum of student",sum)
avg = 5/100;
print("total average of student",avg)
```

## Operators.

- **Operators:** Operators are the symbols that operators on a value a variable. Operators are used to perform operations on variables and value.
- **Arithmetic operators:**
- Arithmetic operators are used with numeric values to perform common mathematical operations:

| Operators | meaning. |
|---|---|
| + | Addition. |
| _ | Subtraction. |
| * | Multiplication. |
| / | Division. |
| % | Modules division. |

- **Assignments operators.**
- Assignment operators are used to assign values to variables:

| | |
|---|---|
| = | equal. |

- **Relational operators / Comparison Operators:**
- Comparison operators are used to compare two values:

| Operators | meaning. |
|---|---|
| < | is less than. |
| <= | is less than or equal. |
| > | is greater than. |
| >= | is greater than or equal. |
| == | is equal. |
| != | is not equal. |

- **Logical operators:**
- Logical operators are used to combine conditional statements:

| Operators | meaning. |
|---|---|
| && | AND |
| \|\| | OR |
| ! | NOT |

- **Bitwise Operators:**
- Bitwise operators are used to compare (binary) numbers:

| | |
|---|---|
| & | AND |
| \| | OR |
| ^ | XOR |
| ~ | NOT |
| << | Zero fill left shift |
| >> | Signed right shift |

- **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 |

- **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. **Example:** x in y
- **not in =** Returns True if a sequence with the specified value is not present in the object. **Example:** x not in y
- **Arithmetic operators:**
- **Python code.py**

```
a = 20
b = 5
print("The value of 9 + 5",a+b)
print("The value of 9 - 5",a-b)
print("The value of 9 * 5",a*b)
print("The value of 9 / 5",a/b)
print("The value of 9 % 5",a%b)
```

**output :**

```
The value of 9 + 5 25          The value of 9 - 5 15
The value of 9 * 5 100         The value of 9 / 5 4.0
```

- **Assignments operators:**
- **Python code.py**

```
a = 20;      a += 5;
b = 20;      b -= 5;
c = 20;      c *= 5;
d = 20;      d /= 5;
print(a)     print(b)
print(c)     print(d)
```

**output:**

```
25    15    100   4.0
```

- **Relational operators / Comparison Operators:**
- **Comparison operators are return Boolean.**
- **Python code.py**

```
a = (10 > 5)            b = (10 < 5)
c = (10 == 5)           d = (10 != 5)
print(a)                print(b)
print(c)                print(d)
```

**output:**

```
True                    False
False                   True
```

- **Logical operators:**
- **Python code.py (and).**

```
a =input("Enter the number a")
b = input("Enter the number b")
c = input("Enter the number c")
if(a>b and a>c):
   print("a is greater")
if(b>a and b>c):
   print("b is greater")
if(c>a and c>b):
   print("c is greater")
```

**output:**

```
Enter the number a 6                Enter the number b 8
Enter the number c 4                b is greater
```

- **Python code.py (or).**
- The logical OR operator ( || ) returns the boolean value true if either or both operands is true and returns false otherwise.

```
bool1  = True
bool2 = False
print("The value of bool1 or bool2 is",(bool1 or bool2))
```

**output:** The value of bool1 or bool2 is True.

- **Python code.py (not).**

```
d = int(input("Enter the total days"))
if(d!=7):
   print("it will not be a week")
else:
   print("it will be week")
```

**output:** Enter the total days 7.

```
it will be week.                Enter the total days 6.     it will not be a week.
```

- **Conditional statement.**
- **Types of conditional statement.**
  1. If.
  2. If-else.
  3. If-elseif-else.
  4. Switch.
- **Python code.py (if)**

```
a =input("Enter the number a")
b = input("Enter the number b")
c = input("Enter the number c")
if(a>b and a>c):
   print("a is greater")
if(b>a and b>c):
   print("b is greater")
if(c>a and c>b):
   print("c is greater")
```

**output:**

```
Enter the number a 6              Enter the number b 8
Enter the number c 4              b is greater
```

- **Python code.py (if else)**

```
d = int(input("Enter the total days"))
if(d!=7):
   print("it will not be a week")
else:
   print("it will be week")
```

**output:** Enter the total days 7.

```
it will be week.                Enter the total days 6.     it will not be a week.
```

- **Python code.py (If-elseif-else).**

```
number = 0
if number > 0:
   print("Positive number")
elif number == 0:
   print('Zero')
else:
   print('Negative number')
print('This statement is always executed')
```

**output:**

```
Zero
This statement is always executed
Positive number
This statement is always executed
Negative number
This statement is always executed
```

- **Python code.py (Switch).**
- **Switch case :** The switch case statement allow variable to be tested again in the list value each value is called case. The switch statement in checked in every value in case.

```
a = int(input("Enter the first number:"))
b = int(input("Enter the second number:"))
c = int(input("Enter the choice:"))
match c:
    case 1:
        print("Addition",a+b)
    case 2:
        print("Subtraction",a-b)
    case 3:
        print("Muntiplication",a*b)
    case 4:
        print("Division",a/b)
    case 5:
        print("modulus",a%b)
```

**output:**

```
 Enter the first number: 20        Enter the second number:30
 Enter the choice:1                Addition 50
Enter the first number: 10         Enter the second number: 5
Enter the choice: 2                Subtraction 5
Enter the first number: 10  Enter the second number: 5
Enter the choice:3                 Muntiplication 50
Enter the first number:10          Enter the second number: 2
Enter the choice:4                 Division 5.0
 Enter the first number:10         Enter the second number:4
Enter the choice:5                 modulus 2
```

- **Bitwise Operators:**
- **Python code.py (not)**

```
a = int (input("Enter the number a"))
b = int (input("Enter the number b"))
a = a^b
b = a^b
a = a^b
print("After the swapping")
print("a number",a)
print("b number",b)
```

**output:** Enter the number a: 4

```
Enter the number b: 2
After the swapping:
a number 2
b number 4.
```

## Keywords.

- **Keywords:** Python Keywords are some predefined and reserved words in Python that have special meanings. Keywords are used to define the syntax of the coding. The keyword cannot be used as an identifier, function, or variable name. All the keywords in Python are written in lowercase except True and False.
- **How many keywords are in python.**
- There are 33 keywords in python.

1. and

It is a Logical Operator.

2. False

Represents an expression that will result in not being true.

3. nonlocal

It is a non-local variable.

4. as

It is used to create an alias name.

5. finally

It is used with exceptions.

6. not

It is a Logical Operator.

7. assert

It is used for debugging.

8. for

It is used to create Loop.

9. or

It is a Logical Operator.

10.break

Break out a Loop.

11.from

To import specific parts of a module.

12.pass

pass is used when the user doesn’t .
want any code to execute.

13.class

It is used to define a class.

14.global

It is used to declare a global variable.

15.raise

raise is used to raise exceptions or errors.

```
name = "Shanelhai"
print('a' in name)(z)
output:
True.
False.
Boolean return karta hai
```

16.continue
Skip the next iteration of a loop.
17.if
To create a Conditional Statement.
18.return
return is used to end the execution.
19.def
It is used to define the Function.
20.import
It is used to import a module.
21.True
Represents an expression that will result in true.
22.del
It is used to delete an object.
23.is
It is used to test if two variables are equal.
24.try
Try is used to handle errors.
25.elif
Conditional statements, same as else-if.
26.in
To check if a value is present in a Tuple, List, etc.
27.while
While Loop is used to execute a block of statements.
28.else
It is used in a conditional statement.
29.lambda
Used to create an anonymous function.
30.with
with statement is used in exception handling.
31.except
try-except is used to handle these errors.
32.None
It represents a null value.
33.yield
yield keyword is used to create a generator function

# string.

- **String:** string is a data type in a python.
- String is a sequence of character enclosed in quotes.
- We can primary write a string in a three ways.
- A character array terminated by a '\0' (null character).
  null character denotes string termination.
  EXAMPLE
  char name[ ] = {'S', 'H', 'R', 'A', 'D', 'H', 'A','\0'};
  char class[ ] = {'A', 'P', 'N', 'A', ' ', 'C', 'O', 'L', 'L', 'E', 'G', 'E', '\0'};
- **Initialising Strings**
  char name[ ] = {'S', 'H', 'R', 'A', 'D', 'H', 'A','\0'};
  char name[ ] = "SHRADHA";
  char class[ ] = {'A', 'P', 'N', 'A', ' ', 'C', 'O', 'L', 'L', 'E', 'G', 'E', '\0'};
  char name[ ] = "SHRADHA";
- **String using Pointers.**
  Store string in memory & the assigned.
  address is stored in the char pointer 'str'.
- **Python.py.**

```
s = "shanelhai"
print(s.upper())
print(s.lower())
print(s.replace("shanelhai","Shaan"))
print(s.find("e"))
print(max(s))
print(min(s))
print(s.capitalize())
print(s.split())
print(s.swapcase())
print(s.strip())
print(s.startswith("s"))
print(s.partition("\\"))
print(s.endswith("i"))
print(s.title())
```

**output:**
```
SHANELHAI    shanelhai    Shaan   4     s
a       Shanelhai    ['shanelhai'] SHANELHAI
Shanelhai    True          ('shanelhai', '', '')          True          Shanelhai
```

- **String slicing.**
- A string in python can be sliced for getting a part of the string.
- **Python code.py.**

```
greeting = "Good Morning, "
name = "Harry"
#concatenate meaning is add on two string.
print(greeting + name)
```

**output.** **Good Morning, Harry**

- **String indexing.**
- Name = “Shaan” => length = 5; 0,1,2,3,4;
- The index in a string starts from 0 to (length-1) in python. In order to slice a string, we use the following string.
- **Python code.py.**

```
#string ko access karna.
name = "Shaan"
print(name[0])
print(name[1])
print(name[2])
print(name[3])
print(name[4])
print(name[0:4])
#that is string slicing.
# name[3] = "d" --> does not work.
#do not change string.
# character ko access karna.
```

**Output:** S h a a n

- **Negative indexing.**
- Negative indexing is a feature in Python that allows you to access elements from the end of an array, moving backwards. In NumPy, this feature is extended to multi-dimensional arrays, making it a versatile tool for data manipulation.

```
name = "Shaan"
print(name[-4])
print(name[-3])
print(name[-2])
print(name[-1])
print(name[-0])
```

**output:** h a a n S

- **Slicing with skip value.**
- We can provide a skip value as a part of our slice like this.
- Word = “python”
- Word = “[1:6:2] = yhn
- **Python code.py.**

```
name = "python"
print(name[1:6:2])
print(name[:7])
print(name[0:])
```

**output:**

```
yhn
python
python
```

- **Escape Sequence Characters.**
- Sequence of characters after back slash ‘\’ .
- Escape sequence characters comprises of more than one character but represents one characters when used within the strings.
- Examples = \n = newline, \t = tab , \’ = single quote , \\ = backshash etc.
- **Python code.py.**

```
story = "it is a good.\n\twe live in an era of an ever-changing society
guided by evolving technology."
print(story)
```

**output:**

```
it is a good.
        we live in an era of an ever-changing society guided by evolving
technology.
```

## Type casting.

- **Type casting:** Type casting function is used to find the data type of a given variable in python.
- **Python code.py**

```
a = "1234"
a = int(a)
print(type(a))
print(a + 5)
aa = 1234
b = "Shaan"
c = 1234.55
d = 123.44
print(type(aa))
print(type(b))
print(type(c))
print(type(d))
```

**output:**

```
<class 'int'>
1239
<class 'int'>                    <class 'str'>
<class 'float'>                  <class 'float'>
```

- **Conversion**
- A number can be converted into a string and vice versa  (if possible).
- There are many functions to convert one data type into another.
- Interger to string conversion.
- String to interger conversion.
- Interger to float conversion.

```
a = 31
a = str(a)
print(type(a))
b = "32"
b = int(b)
print(type(b))
c= 32
c = float(c)
print(type(c))
```

**output:**

```
<class 'str'>
<class 'int'>
<class 'float'>
```

# Loop.

- **Loop:** The purpose of the loop is the except some code number of time.
- **There are four types of loops.**
  1. For loop.
  2. While loop.
  3. Do while loop.
  4. Nested loop.
- **For loop.**
- For loop(initialisation; condition; updation) {
  // do something}
- For loop is a entry control loop.
- **Python code.py.**

```
for i in range(0, 3):
    n = input("Enter the name of employee:")
    p = int(input("Enter the principal:"))
    n = int(input("Enter the time period: "))
    r = int(input("Enter the rate:"))
    si = p*n*r/100
    print("The employee total simpal interst:",si)
```

**output:**

```
Enter the name of employee: Shaan
Enter the principal:100
Enter the time period: 200
Enter the rate:300
The employee total simpal interst : 60000.0
```

- **While loop.**
- While loop.
- While(condition) {
- // do something}
- While loop is a entry control loop.
- While loop is a pre- test loop.
- The condition is specified begin of the loop.

- **Python code.py.**

```
i = 1
table = int(input("Enter the table number:"))
while i<=10:
   r = table*i
   print("",r)
   i = i+1
```

**output:** 2 4 6 8 10 12 14 16 18 20

- **Nested loop.**
- **Nested loop:** A nested loop refers to a loop within a loop, an inner loop within the body of an outer one. Further, the first pass of the outer loop will trigger the inner loop, which will execute to completion. After that, the second pass of the outer loop will trigger the inner loop again.
- For loop(initialisation; condition; updation) {
  // do something}
- For loop(initialisation; condition; updation) {
  // do something}
- Statements.
- Rest of Code.
- **Python code.py.**

```
n= int(input("Please enter the number of rows : "))
for i in range(0,n):
   for j in range(0,i+1):
       print("* ",end="")
   print()
```

**output:**

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

## List

- **List:** List are used to store multiple items in a single variable list are created using square brackets:
- **List:** List are containers to store a set of values of any data type.
- **List items:** List items are ordered changeable and allow duplicate value definition.
- print(): Prints a string to the console.
- input(): Gets input from the user.
- len(): Returns the length of a string or list.
- range(): Returns a list of numbers.
- min(): Returns the minimum value in a list.
- max(): Returns the maximum value in a list.
- sum(): Returns the sum of the values in a list.
- sorted(): Sorts the items in a list.
- reversed(): Reverses the order of the items in a list.
- type(): Returns the type of an object.
- **Python code.py**

```
l = ["Shaan","BCA",8]
print(l)
l.append(9)
print(l)
l.insert(0,1)
print(l)
print("Shaan" in l)
print(11 in l)
print(len(l))
l.clear()
print(l)
```

**output:**

```
['Shaan', 'BCA', 8]
['Shaan', 'BCA', 8, 9]
[1, 'Shaan', 'BCA', 8, 9]
True
False
5
[]
```

- **We have a five students name store the name. and print the name one by one and the name has to be printed until Radha’s name appears.**

- **Python code.py**

```
Students = ["Ram","Shyam","Kishan","Radha","Radhika"]
for Students in Students:
    if Students =="Radha": # if statement in loop.
        break; // keyword used.
    print(Students)
```

**output:**

```
Ram                    Shyam                  Kishan
```

- **A student whose name was krishan left the school. Students list now its name should not be printed in the list. Apart from kishan everyone else's name should be printed.**
- **Python code.py**

```
Students = ["Ram","Shyam","Kishan","Radha","Radhika"]
for Students in Students:
    if Students =="Kishan": # if statement in loop.
        continue; # keyword list main name ni hona chayia.
    print(Students)
```

**output:**

```
Ram          Shyam
Radha        Radhika
```

## Tuples

- **Tuples:** Tuples are used to store multiple items in a single variable.
- Tuples are written with round bracket tuples item are ordered unchangeable and allow duplicate values.
- Tuples is an immutable tuple.
- **Python code.py**

```
A = ("Shaan",1,2,3,4,1,1)
print(A)
print(len(A))
print(type(A))
print(A[0])
print(A[0:2])
print(A.count(1))
print(A.index(3))
B = (1,4,2,5,3)
print(B)
for i in B:
    print(B)
print(min(B))
print(max(B))
print(sum(B))
print(sorted(B))
B2 = [("Shanelhai",21),("python",20)]
print(B2)
B3 = [("computer",5),(5,6)]
print(B3)
```

**output:**

```
('Shaan', 1, 2, 3, 4, 1, 1)                    7
<class 'tuple'>                                Shaan
('Shaan', 1)                          3
3
(1, 4, 2, 5, 3)                                (1, 4, 2, 5, 3)
(1, 4, 2, 5, 3)                                (1, 4, 2, 5, 3)
(1, 4, 2, 5, 3)                                (1, 4, 2, 5, 3)
1                                              5
15
[1, 2, 3, 4, 5]                       [('Shanelhai', 21), ('python', 20)]
[('computer', 5), (5, 6)]
```

## Set

- **Set:** set are used to state multiple items in a single variable.
- A set is a collection which is unordered unchangeable an unindexed and does allow duplicate value.
- Set is written with curly bracket.
- **Python code.py**

```
S = {2,4,6,8,10}
print(S)
print(len(S))
print(type(S))
print(S.pop())
S.remove(8)
print(S)
S.clear()
print(S)
S.add("Shanelhai")
print(S)
S.update(["Code","VS"])
print(S)
S.discard("Shanelhai")
print(S)
E1= {1,2,3}
E2 = {4,5,6}
E3 = {7,8,9}
E2.intersection(E3)
print(E2)
E1.intersection_update(E2)
print(E2)
E3.symmetric_difference(E3)
print(E3)
```

**output:**

```
{2, 4, 6, 8, 10}                5
<class 'set'>                   2
{4, 6, 10}                      set()
{'Shanelhai'}                   {'VS', 'Shanelhai', 'Code'}
{'VS', 'Code'}                  {4, 5, 6}
{4, 5, 6}                       {8, 9, 7}
```

# Dictionary

- **Dictionary :** Dictionary is a collection which is ordered changeable and do not allow duplicate dictionary are written with curly brackets and have keys and values.
- **Python code.py**

```
D = {"brand":"ford","moder":"mustang","year":2003,"year":2005}
print(D)
print(len(D))
print(type(D))
print(D["brand"])
x=D.get("year")
print(x)
print(D.keys())
print(D.values())
print(D.items())
D["Color"]="red"
print(D)
if "brand" in D:
    print("yes the brand keys is in dictionary")
else:
    print("no")
D["color"]="blue"
print(D)
D.update({"brand":"hero"})
print(D)
D.pop("model")
print(D)
D.popitem()
print(D)
del D["year"]
print(D)
D.clear()
print(D)
D1={"name":"shaan",
        "age":20}
for x in D1.keys():
          print(x)
for y in D1.items():
          print(y)
for z in D1.values():
          print(z)
D1.update({"year":1400})
print(D1)
```

## Function

- **Function: -**Function is a black of code which only rune when it is called.
- you can pass data known as parameters into a function and  a function can return data as a result.
- creating a function ....in python a function is defined using the def keyword.
- Code reusability.
- Function returns value as a result.
- We can pass value to the function called parameters.
- **Types of functions.**
- **Pre-defined function.**
- A predefined function is a function that has already been written in the programming language and can be used by the programmer.
  1. Len().
  2. Print().
  3. Int().
  4. Float().
- **User defined function.**
- User-defined function is a block of code that is given a name and can be used to perform a specific task.
- **Def  function name.**
- **types of user-defined function.**
  1. Function with no arguments and no return value.
  2. Function with no arguments and a return value.
  3. Function with arguments and no return value.
  4. Function with arguments and with return value.
- **Function with no arguments and no return value.**
- **Python code.py**

```
def sum ():
   a = int(input("Enter the number:"))
   b = int(input("Enter the number:"))
   s = a + b
   print("The sum", s)
sum()
```

- **Function with arguments and no return value.**
- **Python code.py**

```
def sum (a, b, c):
    if a > b and a > c:
        print("A is a largest number😎:")
    else:
        if b > a and b > c:
            print("B is a largest number😎:")
        else:
            if c > a and c > b:
                print("C is a largest number😎:")
a = int(input("Enter the number a:"))
b = int(input("Enter the number b:"))
c = int(input("Enter the number c:"))
sum(a ,b, c)
```

- **Function with arguments and with return value.**
- **Python code.py**

```
def table(num):
 for i in range(1, 11):
   print(num, 'x', i, '=', num*i)
 return
n = int(input("Enter number: "))
table(n)
```

- **Anonymous functions.**
- anonymous function is a function without a name. It can be defined using the lambda keyword and can be used to perform simple tasks.
- **Python code.py**

```
number = lambda a, b: a + b;
print(number(4,5))
```

## Modules

- **Modules**:-Modules are nothing but group of functions, variable and class that are saved to a file.
- Code reusability.
- If python file (first.py) called module then its module name should be first.
- **Types of modules.**
- **Pre-defined.**
  1. Radam.
  2. Celender.
  3. Keyword.
- **Pre-defined python code.py**
- **Celender.**

```
import calendar
print(calendar.month(2022,8))
```

- **Keyword.**

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

- **User-defined.**
  1. First etc.
- **Python code.py**

```
def sum ():
a = int(input("Enter the number:"))
b = int(input("Enter the number:"))
s = a+b
print("The sum",s)
```

- **Use of second module Python code.py**
- import Modules
  print(Modules.sum())
- **Use of multiple modules python code.py**

```
import Modules
print(Modules.sum())
print(Modules.number())
```

## Object oriented programming

- **OOP: Object Oriented Programming Language.**
- Object oriented programming language is a based-on class and object. Which is used to decomposed into big problem into small problem also help to make code re-useability.
- **Templates:-**
- **Class:** class are the group of member variable and member function. Class is the template to create an object.
- **Object=** An object is an instance of a class. It is a collection of attributes (variables) and methods. We use the object of a class to perform actions. Object is a real word entity such as pen and laptop.
- **Python code.py**

```
class sum():
        a=int(input("enter the first no"))
        b=int(input("enter the second no"))
        c = a + b
        print("sum of two number is",c)
obj = sum()
#python in every class one default constructor = def _ init _(Self).
```

- **Object oriented programming language features.**
- **Inheritance:** property of one class can be in inherited into the other class.
- **Types of inheritance.**
  1. Single Inheritance.
  2. Multiple Inheritance.
  3. Multi-level inheritance.
  4. Hierarchical Inheritance.
  5. Hybrid inheritance.
- **Python code.py**

```
class Father:
   def Lands(self):
      print("Having 50 Ekar Lands")
class Son(Father):
   def Money(self):
      print("Having 10 Lakhs Money")
S = Son()
S.Lands(),            S.Money()
```

**Output** = Having 50 Ekar Lands,          Having 10 Lakhs Money

- **Single Inheritance.**
- Single inheritance enables a derived class to inherit properties from a single parent class.
- **Syntax.**
  Class a // father class.
  Properties
  Class b // child class.
  Properties.
- **Python code.py**

```
class A:
    number1 = int(input("Enter the number1:"))
    number2 = int(input("Enter the number2:"))
    def Add(self):
        print("Addition of two number:",self.number1 + self.number2)
    def Sub(self):
        print("Substation of two number:",self.number1 - self.number2)
class B(A):
    def multi(self):
        print("Multiplication of two number:",self.number1 * self.number2)
    def Div(self):
        print("Division of two number:",self.number1 / self.number2)
    def Rem(self):
        print("Reminder of two number:",self.number1 % self.number2)
obj = B()
obj.Add()
obj.Sub()
obj.multi()
obj.Div()
obj.Rem()
```

- **Multiple-level Inheritance.**
- In this inheritance we have one parent class and multiple child class.
- Python code.py

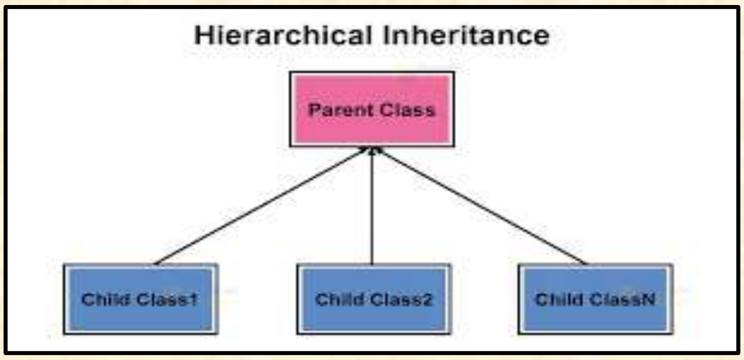
```
class calculation():                    // parent class
   a = int(input("Enter the number a:"))
   b = int(input("Enter the number b:"))
   def add(self):
         sum = self.a + self.b
         print("Addition of two number:",sum)
         sum = self.a - self.b
         print("Subtraction of two number:",sum)
         sum = self.a * self.b
         print("Multiplication of two number:",sum)
         sum = self.a / self.b
         print("Division of two number:",sum)
class logical(calculation):             // child class
      def day(self):
         self.day = int(input("Enter the total days in weeks:"))
         if self.day!=7:
            print("it is not a week:")
         else:
            print("it is a weeks:",)
class maths(logical): child class
   i = 1
   def table(self):
      self.table = int(input("Enter the table name:"))
      while self.i <= 10:
            self.r = self.table * self.i
            print(" ",self.r)
            self.i = self.i+1
obj = maths()
obj.add()
obj.day()
obj.table()
```

- **Multiple inheritance.**
- Inheritance which contains more parent classes and only one child class is called multiple inheritance.
- **Python code.py**

```
class square:          // parent class 1
        a = float(input("Enter the side of square a:"))
        b = float(input("Enter the side of square b:"))
        def s(self):
            area = self.a * self.b
            print("The are of square:",area)
class logical:          // parent class 2
        def day(self):
            self.day = int(input("Enter the total days in weeks:"))
            if self.day!=7:
                print("it is not a week:")
            else:
                print("it is a weeks:",)
class calculation():    // parent class 3
    def add(self):
        a = int(input("Enter the number a:"))
        b = int(input("Enter the number b:"))
        sum = self.a + self.b
        print("Addition of two number:",sum)
        sum = self.a - self.b
        print("Subtraction of two number:",sum)
        sum = self.a * self.b
        print("Multiplication of two number:",sum)
        sum = self.a / self.b
        print("Division of two number:",sum)
class maths(square,logical,calculation):         //child class
    i = 1
    def table(self):
        self.table = int(input("Enter the table name:"))
        while self.i <= 10:
                self.r = self.table * self.i
                print(" ",self.r)
                self.i = self.i+1
```

```
obj = maths()
obj.s()
obj.day()
obj.add()
obj.table()
```

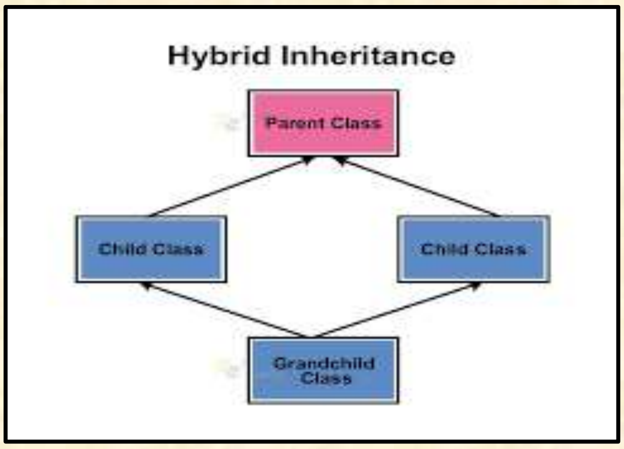
- **Hierarchical Inheritance.**
- Hierarchical inheritance which can contain only one parent class and multiple child classes but each child classes can parent class properties.



- **Python code.py**

```
class Father:
   surname =  "saifi";
   def show(self):
      print("My surname",self.surname)
class son1(Father):
   def disp(self):
      print("My name is Shanelhai",self.surname)
class son2(Father):
   def out(self):
      print("My name is Shahnawaz",self.surname)
s1 = son1()
s2 = son2()
s1.show()
s1.show()
s2.out()
s1.disp()
```

- **Hybrid inheritance.**
- Hybrid inheritance is a combination of multiple inheritance and multilevel inheritance. A class is derived from two classes as in multiple inheritance. However, one of the parent classes is not a base class. It is a derived class.



- **Python code.py**

```
class A:
  def method_a(self):
    print("Method A")

class B(A):
  def method_b(self):
    print("Method B")

class C(B, A):
  def method_c(self):
    print("Method C")

class D(C):
  def method_d(self):
    print("Method D")

d = D()
d.method_a()
d.method_b()
d.method_c()
d.method_d()
```

- **Encapsulation:** it is also known as data hiding. It means Users use the program but don’t know how data can be flow and how it works internally it also known as wrapping data.
- **Python code.py**

```
class Students:
  def __init__(self, name, rank, points):
    self.name = name
    self.rank = rank
    self.points = points
  # custom function
  def demofunc(self):
    print("I am "+self.name)
    print("I got Rank ",+self.rank)
# create 4 objects
st1 = Students("Steve", 1, 100)
st2 = Students("Chris", 2, 90)
st3 = Students("Mark", 3, 76)
st4 = Students("Kate", 4, 60)
# call the functions using the objects created above
st1.demofunc()
st2.demofunc()
st3.demofunc()
st4.demofunc()
```

- **Abstraction:** Abstraction is a process of hiding the implementation details from the user only the highlighted set of services provided to the user.
- **Python code.py**

```
import Afirst,Asecond
Afirst.d()
Asecond.a()
Asecond.b()
Asecond.c()
```

- **Abstract.**
- Abstract class is a class that contains one or more abstract method is a called abstract class.
- An object of an abstract class cannot be created.
- Python provides ABC module to work with abstraction.
- We use abstract method decorator to define abstract method.
- **Python code.py**

```
from abc import ABC, abstractmethod
class Car(ABC):
    def show(self):
        print("Every car has 4 wheels")
    @abstractmethod
    def speed(self):
        pass
class Maruti(Car):
    def speed(Self):
        print("Speed is 100Km/H")
class Suzuki(Car):
    def speed(self):
        print("Speed is 70Km/H")
obj = Maruti()
obj.show()
obj.speed()

obj2 = Suzuki()
obj2.show()
obj.speed()
```

- **Interface.**
- Interface is nothing but abstract class which contains only abstract method but not any normal method.
- **Python code.py**

```
from abc import ABC, abstractmethod
class Shape(ABC):
    @abstractmethod
    def show(self):
        pass
    @abstractmethod
    def disp(self):
        pass
class Square(Shape):
    def disp(self):
        pass
class Circle(Square):
    def show(self):
        print("circle has circle shape....")
    def disp(self):
        print("Square has 4 sides....")
obj=Circle()
obj.show()
obj.disp()
```

- **Polymorphism:** Polymorphism in Python is the ability of an object to take many forms. In simple words, polymorphism allows us to perform the same action in many different ways.
- **Python code.py**

```
print(5+5)
print("5"+"5")
```

1. Method overloading, also known as Compile time Polymorphism.
2. Method overriding, also referred to as Run time Polymorphism.
3. Overloading of operators.
4. Class Polymorphism in Python.

- **Method overloading.**
- Whenever class container more than one method with same name and different types parameters called overloading.
- **Python code.py**

```
class A:
    def show(self):
        print("Welcome")
    def show(self,firstname=" "):  // parameters
        print("Welcome",firstname)
    def show(self,firstname=" ",leastname=" "):
        print("Welcome",firstname,leastname)
obj = A()
obj.show()
obj.show("Shanelhai")
obj.show("Shanelhai","Saifi")
```

- **Method overriding.**
- Whenever we writing method name with same signature in parents & child class called method overriding.
- **Python code.py**

```
class A: # parent class
    def show(self):
        print("This is a parent class")
class B(A):
    def show(self):
        #use super key function.
        super().show()
        print("This is a child class")
obj = B()
obj.show()
```

- **How to document attribute.**
- **Python code.py**

```
class a():
    "Shanelhai"
    age = 20
    print(age)
obj = a()
print(a.age)
print(obj.age)
print(obj.__doc__) #use of document attribute.
```

- **How to pass value function.**

```
class A:
    def fun(self,age,name,address):
        print(age," ",name," ",address)
obj = A()
obj.fun(10,"Shaan","Haldwani")
```

**output** = 10   Shaan   Haldwani

- **Constructor.**

```
class A:
    def __init__(self,age,name,address):
        print(age," ",name," ",address)
obj = A(10,"Shaan","Haldwani")
```

**output** = 10   Shaan   Haldwani

- **Access modifiers.**
- Access modifiers are used to set the limit of member accessibility.
- There are three types.
  1. **Public:** Public methods are accessible outside the class and with the help of objects the public methods can be invoked inside the class.
  2. **Private:**
  3. **Protected:**
- **Python code.py**

```
class a:       #parent class.
   a = 10      #public.
   _b = 20     #proctected.
   __c = None  #private.
   print(a," ",_b," ",__c)
   #public and proctected member function use of outside.
   def add(self):
      self.__c = self.a + self._b
      print("addition to two value:",self.__c)
obj = a()
obj.add()
print(obj.a)
print(obj._b)
print(obj.__c)
#inheritance.
class b(a):       #child class.
   def show(self):
      print(obj.a)
      print(obj._b)
      print(obj.__c)
obj = b()
obj.show()
```

- **Multithreading in python.**
- Multithreading is defined as the ability of a processor to execute multiple threads concurrently. In a simple, single-core CPU, it is achieved using frequent switching between threads. This is termed context switching.
- **Why use multithreading.**
- The purpose of multithreading is to run multiple tasks and functions at the same time.
- **What is thread?**
- Thread is a pre-defined class which is available in threading module.
- Thread is a basic unit of CPU and it is well known for independent execution.
- Important by default main thread in every program.
- **Thread class method.**
  1. start(): : This method starts the thread's activity.
  2. run(): : This method represents the thread's activity.
  3. join(): : This method blocks the calling thread until the thread whose join() method is called terminates.
  4. name: : This method returns the name of the thread.
  5. getName(): : This method returns the name of the thread.
  6. setName(): : This method sets the name of the thread.
  7. is_alive(): : This method returns True if the thread is alive, False otherwise.

- **Python code.py**

```
from time import sleep
from threading import Thread
class A(Thread):
   def run(Self):
      for i in range(2):
         name = input("Enter the name of employee:")
         age  = int(input("Enter the age of employee:"))
         salary = int(input("Enter the salary of employee:"))
         phone  = int(input("Enter the phone number of employee:"))
         address = input("Enter the address of employee:")
         sleep(1)
class B(Thread):
   def run(Self):
      for i in range(2):
         principal = int(input("Enter the initial principal balance of employee:"))
         rate      = int(input("Enter the annual interest rate of employee:"))
         time      = int(input("Enter the time in years of employee:"))
         si      = principal * rate * time / 100
         print("Total simple interest of employeess:",si)
         sleep(1)
t1 = A()
t2 = B()
t1.start()
t2.start()
t1.join()
t2.join()
```

- **Operator overloading.**
- Operator overloading is a manner in which OO systems allow the same operator’s name or symbol to be used for multiple operations.
  Operator overloading in python docs.python
- **Python code.py**

```
class Vector:
    def __init__(self,i,j,k):
        self.i = i
        self.j = j
        self.k = k
    def __str__(self):
        return f"{self.i}i + {self.j}j + {self.k}k"
    def __add__(self,x):
        return  Vector(self.i+x.i,self.j+x.j,self.k+x.j)
V1 = Vector(3, 5, 6)
print(V1)
V2 = Vector(1,2,9)
print(V2)
print(V1+V2)
```

- **Python code.py**

```
class Point:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def __str__(self):
        return "({0},{1})".format(self.x, self.y)

    def __add__(self, other):
        x = self.x + other.x
        y = self.y + other.y
        return Point(x, y)
p1 = Point(1, 2)
p2 = Point(2, 3)

print(p1+p2)
```

- **Exception handling in python.**
- Exception is nothing but runtime errors and it occurs to incorrect implementation of logic.
- It is a mechanism through which we can handless the runtime errors.
  1. Try.
  2. Except.
  3. Else.
- **Python code.py**

```
a = int(input("Enter the number a:"))
b = int(input("Enter the number b:"))
try:
    c = a/b
    print("Result:",c)
except:
    print("Can't devide to zero:")
else:
    print("Shaan:")
```

- ***Args and ** Kwargs in Python.**
- **Args :-** The special syntax *args in function definitions in python is used to pass a variable number of arguments to a function. It is used to pass a non-key worded, variable-length argument list. The syntax is to use the symbol * to take in a variable number of arguments; by convention, it is often used with the word args.
- **Kwargs:-** Kwargs is a special syntax that allows us to pass a variable length of keyword arguments to the function. The **kwargs stands for keyword arguments which are passed along with the values into the function.
- ***Args and ** Kwargs in Python both is a optional.**
- **Python code.py**

```
def funArgs(Normal,*Args,** Kwargs ):
    print(Normal)
    for item in Args:
        print(item)
    print("\nNow I would like to introduce some of our heroes")
    for key, value in Kwargs.items():
        print(f"{key}is a{value}")
har = ["Harry","Rohan","Skilf","Hammad","Shivam","The Programmer"]
Normal = "I am a normal argument and the students are"
Kwargs = {"Rohan":"Monitor","Harry":"Fitness Instructor","The
programmer":"coordinator","Shivam":"Cook"}
funArgs(Normal,*har,** Kwargs)
```

- NumPy

  is a library for scientific computing with Python. It provides a high-performance multidimensional array object, and various tools for working with arrays.

- Pandas

  is a library for data analysis and manipulation. It provides data structures and operations for working with numerical tables and time series.

- Matplotlib

  is a library for data visualization. It provides a variety of functions for creating plots, charts, and histograms.

- Scikit-learn

  is a library for machine learning. It provides a variety of algorithms for classification, regression, clustering, and dimensionality reduction.

- TensorFlow

  is a library for machine learning and deep learning. It provides a variety of tools for building and training neural networks.