

- 1.What is Python
- 2.Features of Pyhton
- 3.Python History
4. what type of application we can develop through python
5. Python Install
6. First Python Program
7. Python DataType
8. Python Literals
9. Python Operators
10. Python Condition
11. Python Iterative Control
12. Python String
13. Python List
- 14.Python Tuple
- 15.Python Set
16. Python Dictionary
17. Python Function
18. OOPS Concept

# Python Basic

## 1.What is Python ?

**Python** is a general purpose, dynamic, high level, and **interpreted** (line by line execute )programming language. It supports Object Oriented programming approach to develop applications. It is simple and easy to learn and provides lots of high-level data structures.

## 2. Features of Python?

Python provides lots of features that are listed below.

### i) Easy to Learn and Use

Python is easy to learn and use. It is developer-friendly and high level programming language.

### ii) Interpreted Language

Python is an interpreted language i.e. interpreter executes the code line by line at a time. This makes debugging easy and thus suitable for beginners.

### iii) Cross-platform Language

Python can run equally on different platforms such as Windows, Linux, Unix and Macintosh etc. So, we can say that Python is a portable language.

### iv) Free and Open Source

Python language is freely available at [official web address](#).The source-code is also available. Therefore it is open source.

### v) Extensible

It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our python code.

### vi) Integrated

It can be easily integrated with languages like C, C++, JAVA etc.

### 3. History of Python

Python laid its foundation in the late 1980s.

The implementation of Python was started in the December 1989 by **Guido Van Rossum** at CWI in Netherland.

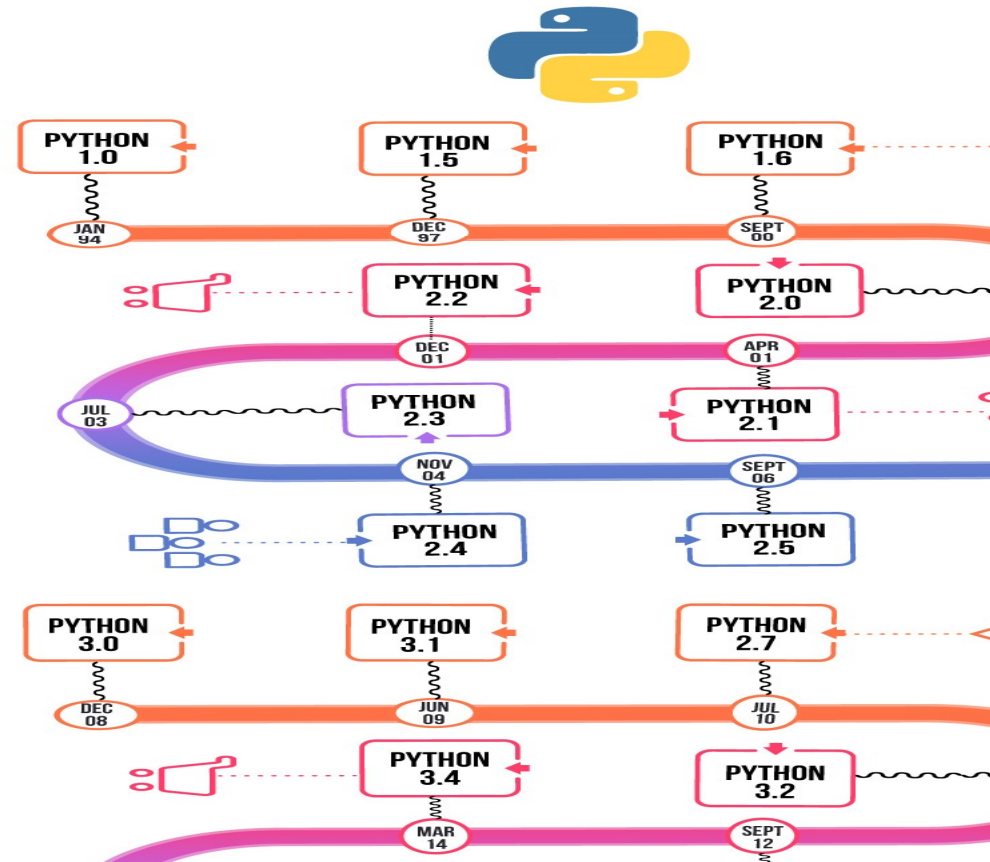
In 1994, Python 1.0 was released with new features like: lambda, map, filter, and reduce.

- **Python is influenced by following programming languages:**

ABC language.

Modula-3

-> Python Version :-



## 4. what type of application we can develop through python(why learn)

### 1) Web Applications

We can use Python to develop web applications. It provides libraries to handle internet protocols such as HTML and XML, JSON, Email processing, request, BeautifulSoup, Feedparser etc. It also provides Frameworks such as Django, Pyramid, Flask etc to design and develop web based applications. Some important developments are: PythonWikiEngines, Pycoco, PythonBlogSoftware etc.

### 2. Desktop GUI Applications

Python provides Tk GUI library to develop user interface in python based application. Some other useful toolkits wxWidgets, Kivy, pyqt that are useable on several platforms. The Kivy is popular for writing multitouch (a touch is just a mouse single touch , multi-touch operations like "zoom" and "spin" ) applications.

### 3) Scientific and Numeric

Python is popular and widely used in scientific and numeric computing. Some useful library and package are SciPy, Pandas, IPython etc. SciPy is group of packages of engineering, science and mathematics.

### 5) Business Applications

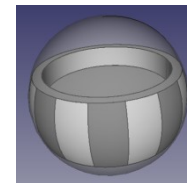
Python is used to build Business applications like ERP and e-commerce systems. Tryton is a high level application platform.

### 6) Console Based Application

We can use Python to develop console based applications. For example: **IPython**.

### 7.) 3D CAD Applications

To create CAD application Fandango is a real application which provides full features of CAD.



### 8. Applications for Images

Using Python several application can be developed for image. Applications developed are: VPython, Gogh, imgSeek etc.

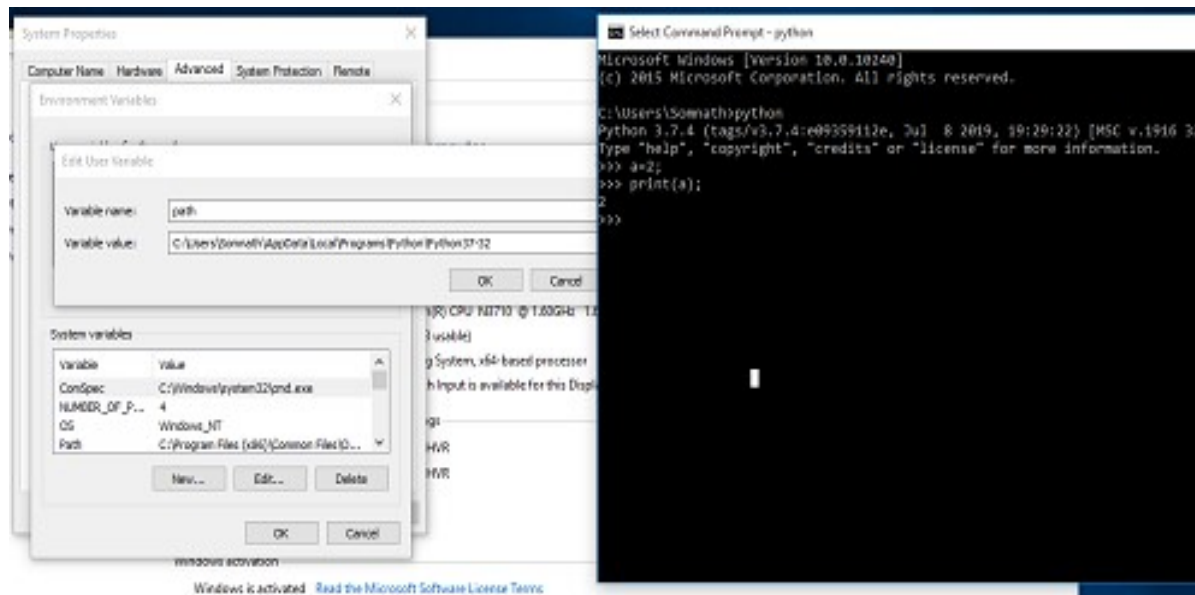
## 5. Python Install for windows

Visit the link <https://www.python.org/downloads/> to download the latest release of Python. In this process, we will install Python 3.8.1 on our Windows operating system.



### -> set the path of python

"my computer" and go to Properties → Advanced → Environment Variables.

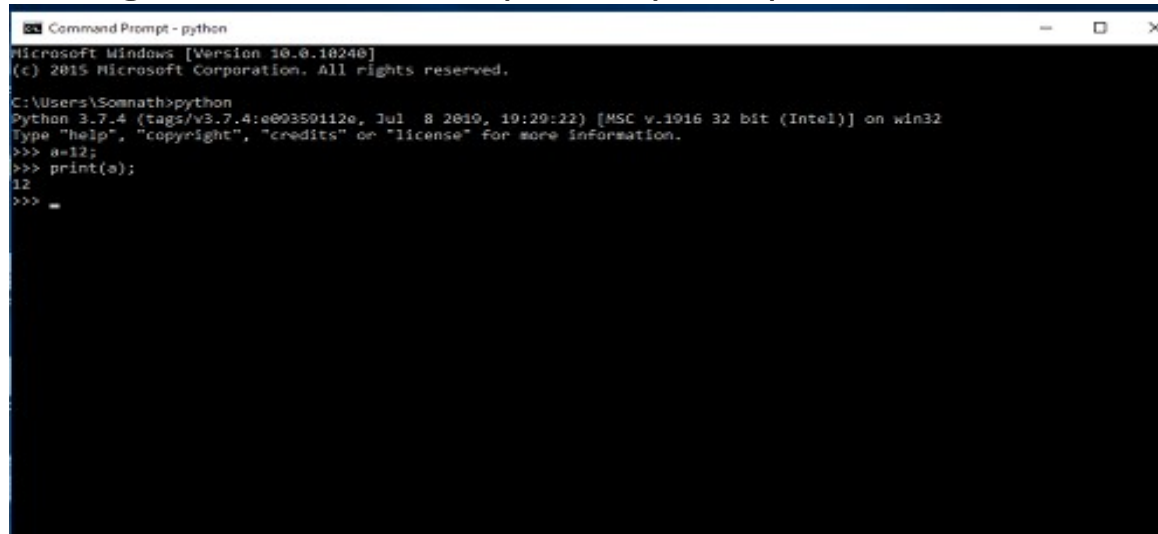


## 6. First Python Program

Python provides us the two ways to run a program:

- Using Interactive interpreter prompt
- Using a script file

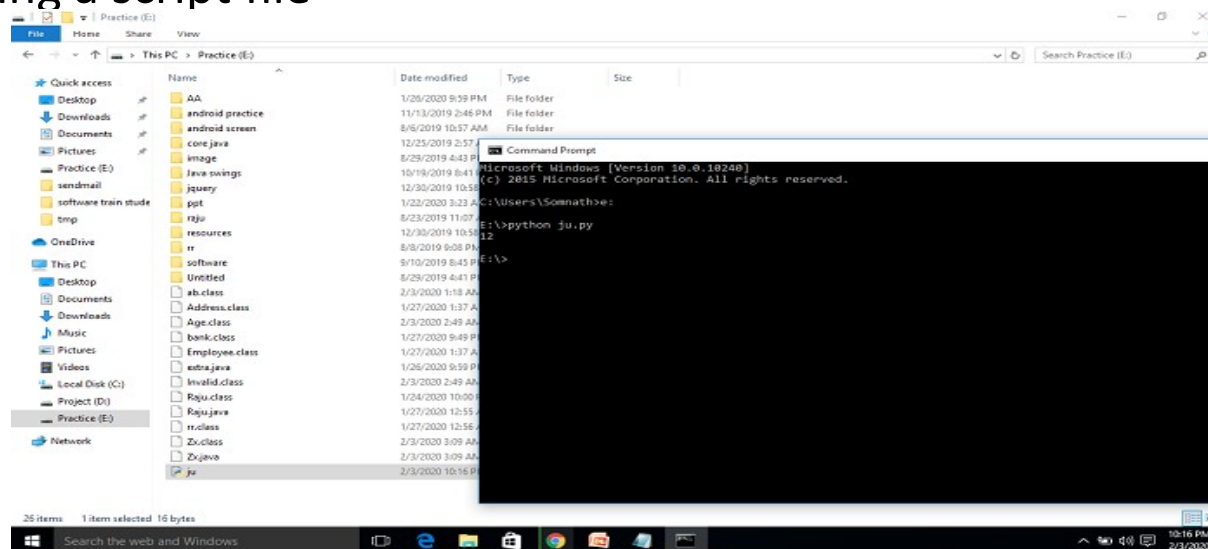
->Using Interactive interpreter prompt



```
Microsoft Windows [Version 10.0.18240]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\Somnath>python
Python 3.7.4 (tags/v3.7.4:0000000, Jul 8 2019, 10:29:22) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> a=12;
>>> print(a);
12
>>>
```

-> Using a script file



Install PyCharm

<https://www.jetbrains.com/pycharm/download/download-thanks.html?platform=windows>

## 7. Python Data Types & Variable

```
Int a=12; // int is a DataType and a is Variable
```

**But Python point of view no need to declare any DataType .**

**Example->**

```
a=12; // Python automatically understand it is Int type Data  
Print(type(a));
```

### -> Standard data types

- Numbers
- String
- List
- Tuple
- Dictionary

## 8. Python Literals:-

```
# multi Line String  
a='hello' \  
  'somnath';  
print(a);  
#triple quotation  
s="welcome  
to  
G.tech "  
print(s);  
# Special literals.  
s1=None; # its not a String it is Literals  
print(s1);
```

## 9. Python Operators:-

Operator -> Operator is a Symbol like .. +, - etc

Operand -> Operand is value or variable like 5+5 , 5 is a operand

1. Arithmetic operators(+, -, \*, /, %, \*\*, //)

**\*\* (exponent)** -> power like  $2^3=8$

**// (floor)** ->  $5//2$  ; Output:- 2

2. Assignment operators(+=, -=, \*=, /=, %=, \*\*=, //=)

3. Comparison operators(==, !=, >, >=, <, <=)

4. Logical operators(and, or, not)

5. Bitwise operator(&, |, ^, <<, >>)

**^(xor)** -> The resulting bit will be 1 if both the bits are different otherwise the resulting bit will be 0.

6. Membership Operators(in, not in)

Its use List, tuple & Dictionary

7. Identity Operators(is, is not)

a=5;

if type(a) is int:

print("ok");

### -> Simple Programme

```
a=5+3*2**3;
```

```
print(a);
```

Find output=???

### -> Python Comments

# single line comments

'''multiline

comments'''



## 10. Python Decision Control:

If /if-else /nested if

```
#if block
if(1):
    print("ok");
# if else block
if(0):
    print("ok");
else:
    print("sry");
#nested if
if(1):
    if(1):
        print("ok");
```

## 11. Python Iterative Control For/while

```
#for loop
for i in range(0,10):
    print(i,end="");
#for loop with increment 2
for j in range(0,10,2):
    print(j);
#inner for loop
for k in range(0,5):
    for l in range(0,5):
        print(l,end="");
```

-> break & continue & pass Statement

```
i=1;
while True:
    print(i);
    i=i+1;
    if(i>5):
        break;
```

```
-> Continue
i=1;
while True:
    print(i);
    i=i+1;
    if(i<5):
        continue;
    else:
        break;
```

-> Pass

(pass keyword is used to execute nothing; it means, when we don't want to execute code, the pass can be used to execute empty)

```
for i in range(0,5):
    if i==3:
        pass;
    else:
        print(i);
```

```
#while loop
i=0;
while(i<5):
    print(i);
    i=i+1;
#multiple of number
j=0;
while j<=10:
    print("%d X %d = %d \n"%(5,j,5*j));
    j = j+1;
```

## 12. Python String

The sequence of characters in the quotes. Python allows us to use single quotes, double quotes, or triple quotes to create the string.

```
s="hello";  
s1='hello';  
s2="hello";  
print(s,s1,s2);  
Print(type(s));
```

### -> Strings indexing and splitting

Like other languages, the indexing of the python strings starts from 0.

For example, The string "HELLO" is indexed.

str = "HELLO"

H	E	L	L	O
0	1	2	3	4

str[0] = 'H'

str[1] = 'E'

str[2] = 'L'

str[3] = 'L'

str[4] = 'O'

However, we can use the : (colon) operator in python to access the substring. Consider the following example.

str = "HELLO"

H	E	L	L	O
0	1	2	3	4

str[0] = 'H'      str[:] = 'HELLO'

str[1] = 'E'      str[0:] = 'HELLO'

str[2] = 'L'      str[:5] = 'HELLO'

str[3] = 'L'      str[:3] = 'HEL'

str[4] = 'O'      str[0:2] = 'HE'

str[1:4] = 'ELL'

## 12.1 Reassigning strings

Strings are immutable (not changable ) in python. A string can only be replaced with a new string since its content can not be partially replaced

```
s="hello";  
s[0]='j';  
print(s);  
Error:-  
Traceback (most recent call last):  
  File "C:/Users/Somnath/Desktop/pac.py", line 2, in <module>  
    s[0]='j';  
TypeError: 'str' object does not support item assignment
```

```
s="hello";  
s="hello2";  
print(s);
```

Output-  
hello2

### ->String Operators

```
str = "Hello"  
str1 = " world"  
print(str*3) # prints HelloHelloHello  
print(str+str1)# prints Hello world  
print(str[4]) # prints o  
print(str[2:4]); # prints ll  
print('w' in str) # prints false as w is not present in str  
print('wo' not in str1) # prints false as wo is present in str1.  
print("The string is : %s"%(str)) # prints The string str : Hello
```

### -> Python Formatting Operator

```
a=12;  
b=12.5;  
c="somnath";  
print("my name is %s,age is %d,markes is %f"%(c,a,b));
```

## 12.2 Built-in String functions

```
s="somnath";
s1 = s.capitalize(); # proper case
print(s1);
#lower string
str = "SOMNATH"
str2 = str.casefold()
print("New value:", str2) # convert lower case
# center function use
str3 = "Hello somnath"
# Calling function
str4 = str3.center(20)
print("New value:", str4)
# center function use replace #
str = "Hello somnath "
str2 = str.center(20,'#')
print(str2) # replace with #
#count word
str = "Hello somnatmh"
str2 = str.count('m') #str.count('m',2,3) 2 start index and 3 is end index
print(str2)
# endswith
str = "Hello this is somnath."
abc = str.endswith("is",0,13)
# Displaying result
print(abc)
# expands tabs
str = "Welcome \t to \t the \t Python."
# Calling function
str2 = str.expandtabs(1) #tab bydefault 4 space extra 1 add
print(str2);

# find(sub[, start[, end]])
str = "Welcome to the Python."
str3 = str.find("t",0,25)
print(str3)
# format
val = 10
# Calling function
print("decimal: {0:d}".format(val)); # display decimal result
print("hex: {0:x}".format(val)); # display hexadecimal result
print("octal: {0:o}".format(val)); # display octal result
print("binary: {0:b}".format(val)); # display binary result
```

## 13 Python List

A list can be defined as a collection of values or items of different types. The items in the list are separated with the comma (,) and enclosed with the square brackets [].

```
emp = ["som", 102, "india"];  
print("Name : %s, ID: %d, Country: %s"%(emp[0],emp[1],emp[2]))
```

**List indexing and splitting->**

List = [ 0, 1, 2, 3, 4, 5]

0	1	2	3	4	5
---	---	---	---	---	---

List[0] = 0

List[0:] = [0,1,2,3,4,5]

List[1] = 1

List[:] = [0,1,2,3,4,5]

List[2] = 2

List[2:4] = [2, 3]

List[3] = 3

List[1:3] = [1, 2]

List[4] = 4

List[:4] = [0, 1, 2, 3]

List[5] = 5

List = [ 0, 1, 2, 3, 4, 5]

Forward Direction → 0 1 2 3 4 5

0	1	2	3	4	5
---	---	---	---	---	---

```
//Backwordd Direction List display  
emp = ["som", 102, "india"];  
print(emp[-1]);
```

## ->Updating List values

Lists are the most versatile data structures in python since they are mutable and their values can be updated by using the slice and assignment operator.

Python also provide us the append() method which can be used to add values to the string.

```
List = [1, 2, 3, 4, 5, 6,2]
print(List)
List[2] = 10; #update 2 index value
print(List)
List[1:3] = [89, 78] # 1 and 2 index are del
print(List)
#del List[0] # del perticular index
#List.remove(5); # remove 5 value
#print(List);
#append list
my_list = ['gtech', 'for']
another_list = [6, 0, 4, 1]
my_list.append(another_list)
print (my_list)
#estend List
my_list1 = ['gtech', 'for']
my_list1.extend('abc'); # split value and
add "a","b","c"
#my_list1.extends([25,89,83])
print (my_list1)

# use min() , max(), sum()
```

### Python List Operations

```
List = [1, 2, 3, 4, 5, 6]
List1=[5,8,9,6];
print(List*2); # Repetition, two times print value
print(List+List1); #Concatenation, add list and List1,
print(2 in List); # Membership, print True if 2 include in List
for l in List: #Iteration print value
    print(l);
print(len(List)); # return Length
```

### List Built-in functions

```
List = [1, 2, 3, 4, 5, 6]
List1=[5,8,9];
List1.clear(); # empty List
print(List1);
List1=List.copy(); #copy List
print(List1);
print(List.count(4)); # haw many 4 include in your List
print(List.index(4)) # index value return that contain 4
List.insert(2,12); # 2 index ,12 value put not Replace ( List.append(20) its mean add 20 end of list)
print(List);
print(List.pop()); # List.pop(-2) that means del reverse second position value
List.reverse(); # reverse List not sorting
print(List);
List.sort(reverse=False); # List sort Asc order
print(List);
```

## 14. Python Tuple

Python Tuple is used to store the sequence of immutable python objects. Tuple is similar to lists since the value of the items stored in the list can be changed whereas the tuple is immutable and the value of the items stored in the tuple can not be changed.

```
tuple1 = (10, "abc", 30, 40.5, 50, 60)
print(tuple1);
```

### Tuple indexing and splitting

    Tuple = ( 0, 1, 2, 3, 4, 5 )

0	1	2	3	4	5
---	---	---	---	---	---

Tuple[0] = 0            Tuple[0:] = (0, 1, 2, 3, 4, 5)

Tuple[1] = 1            Tuple[:] = (0, 1, 2, 3, 4, 5)

Tuple[2] = 2            Tuple[2:4] = (2, 3)

Tuple[3] = 3            Tuple[1:3] = (1, 2)

Tuple[4] = 4            Tuple[:4] = (0, 1, 2, 3)

Tuple[5] = 5

### List and Tuple Operation Same

#### List VS Tuple:-

##### List->

The list is used in the scenario in which we need to store the simple collections with no constraints where the value of the items can be changed.

##### Tuple->

The tuple is used in the cases where we need to store the read-only collections i.e., the value of the items can not be changed. It can be used as the key inside the dictionary.

### Nesting List and tuple

```
a = [(101, "Ayush", 22), (102, "john", 29), (103, "james", 45), (104, "Ben", 34)]
print(a);
print(a[0])
```

### Possible to add another value in Tuple ??

No, #using merge of tuples with the + operator you can add an element and it will create a new tuple

```
t = (1,2,3)
```

```
t = t + (1,8) # this t is a different object
```

#adding items in a specific index

```
t = t[:2] + (15, 20, 25) # index from 2 (value replace 1 and 8)
```

```
print(t);
```

### # Sort and Reverse Tuple

```
t=(25,89,10,8);t1=[5,89,10,8]g=reversed(t);
```

# tuple is immutable so g store reverse value and return object reference

```
print(tuple(g));
```

# show tuple value

```
g1=sorted(t1,reverse=False); # sorted tuple
```

```
print(tuple(g1));
```

## 15. Python Set

The set in python can be defined as the unordered collection of various items enclosed within the curly braces. The elements of the set **can not be duplicate. The elements of the python set must be immutable.**

Unlike other collections in python, there is no index attached to the elements of the set, i.e., we cannot directly access any element of the set by the index. However, we can print them all together or we can get the list of elements by looping through the set.

### Creating a set:-

```
Days = {"Monday", "Tuesday", "Wednesday", "Thursday",  
        "Friday", "Saturday", "Sunday"}  
print(Days)  
print(type(Days))
```

Another Way ...

```
Days = set(["Monday", "Tuesday", "Wednesday", "Thursday",  
           "Friday", "Saturday", "Sunday"])  
print(Days)  
print(type(Days))
```

### Python Method ->

```
Days = set(["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"])  
print(Days)  
Days.add('abc');  
Days.add('xyz');  
print(Days);  
Days.update(["Friday", "July", "August", "September", "October"]); # update set not Element  
print(Days);  
Days.discard("july"); # del july ##if not find "july" do not show any error  
print(Days)  
# Days.remove("January"); # if not find "july" show any error.  
#Days.pop();  
#Days.clear(); # clear all set  
# union | operator  
Days1 = {"Monday", "Tuesday", "Wednesday", "Thursday"}  
Days2 = {"Friday", "Saturday", "Sunday"}  
print(Days1 | Days2) #printing the union of the sets  
#Intersection of two sets  
set1 = {"Ayush", "John", "David", "Martin"}  
set2 = {"Steve", "Milan", "David", "Martin"}  
print(set1 & set2) #prints the intersection of the two sets  
#Intersection_update() method  
a = {"ayush", "bob", "castle"}  
b = {"castle", "dude", "emyway"}  
c = {"fuson", "gaurav", "castle"}  
a.intersection_update(b, c)  
Print(a) # common data show (" castle ")  
#Difference of two sets  
Days1 = {"Monday", "Tuesday", "Wednesday", "Thursday"}  
Days2 = {"Monday", "Tuesday", "Sunday"}  
print(Days1 - Days2) #{"Wednesday", "Thursday" will be printed} #
```

### FrozenSets:-

The frozen sets are the immutable form of the normal sets, i.e., the items of the frozen set can not be changed and therefore it can be used as a key in dictionary. The elements of the frozen set can not be changed after the creation. We cannot change or append the content of the frozen sets by using the methods like add() or remove().

```
Frozenset = frozenset([1,2,3,4,5])  
print(type(Frozenset))  
print("\nprinting the content of frozen set...")  
for i in Frozenset:  
    print(i);  
Frozenset.add(6) #it is immutable that means not possible to change .
```



## 16. Python Dictionary

Dictionary is used to implement the key-value pair in python. The dictionary is the data type in python which can simulate the real-life data arrangement where some specific value exists for some particular key.

In other words, we can say that a dictionary is the collection of key-value pairs where the value can be any python object whereas the keys are the immutable python object,

### Creating the dictionary:->

```
Dict = {"Name": "Somnath", "Age": 30}
print("my name is " + Dict["Name"]);
```

```
Dict = {"Name": "Somnath", "Age": 30}
Dict["Name"] = "bhadra"; # update Dictionary
print("my name is " + Dict["Name"]);
```

```
Dict = {"Name": "Somnath", "Age": 30}
Dict1 = {};
Dict1 = Dict; #copy
print(Dict1);
del Dict1["Name"];
print(Dict1);
Dict.clear(); #clear function
print(Dict);
#Iterative
Dict2 = {"Name": "Somnath", "Age": 30}
for x in Dict2.values(): #Dict2.items() show key and value
    print(x);
#duplicate key nor support
Dict3 = {"Name": "Somnath", "Age": 30, "Name": "abc"}
for x,y in Dict3.items():
    print(x,y);
#formkey function
keys = {'a', 'e', 'i', 'o', 'u'}
value = 'vowel'
vowels = dict.fromkeys(keys, value)
print(vowels);
#get function use
dictionary = {"message": "Hello"}
data = dictionary.get("message1") # not include key return none
print(data) # Hello, World!
#update value
dictionary["message"] = "somnath";
print(dictionary);
```

## .7. Python Function :

A function can be defined as the organized block of reusable code which can be called whenever required.

### Creating a function:-

```
def ab():  
    print("ok");  
ab();
```

### Argument Function

```
def ab(a):  
    print(a);  
ab(12);
```

### Types of arguments:-

- Required arguments
- Keyword arguments
- Default arguments
- Variable-length arguments

```
def ab(a): # required argument  
    print(a);  
ab(12);  
def ab1(a,b): # key argument  
    print("a value is ",a,"and ",b)  
ab1(b=50,a=75);  
def ab2(a,b=90): # default argument  
    print("value a",a,"and",b);  
ab2(20);  
def ab3(*a): # variable length argumnt  
    print("passed value",type(a));  
    print("print value ");  
    for x in a:  
        print(x);  
ab3("ab","cd","ef","gh");  
def ab(a,b=None): #optional argument  
    print(a,b);  
ab(10);
```

### ->Call by reference in Python

In python, all the functions are called by reference, i.e., all the changes made to the reference inside the function revert back to the original value referred by the reference.

```
def ch(list1):  
    list1.append(10);  
    list1.append(50);  
    print(list1);  
list1=[58,90,40];  
ch(list1);  
print(list1);
```

### String Example

```
def ch(str):  
    str+="somnath";  
    print(str);  
str="hello";  
ch(str);  
print(str); # not possible to change original value
```

## 7.1 Scope of variables:-

The scopes of the variables depend upon the location where the variable is being declared. The variable declared in one part of the program may not be accessible to the other parts.

- In python, the variables are defined with the two types of scopes.
- Global variables
- Local variables

```
def ch():  
    m="hello"; # local variable  
    print(m);  
print(m); # not print outSide variable
```

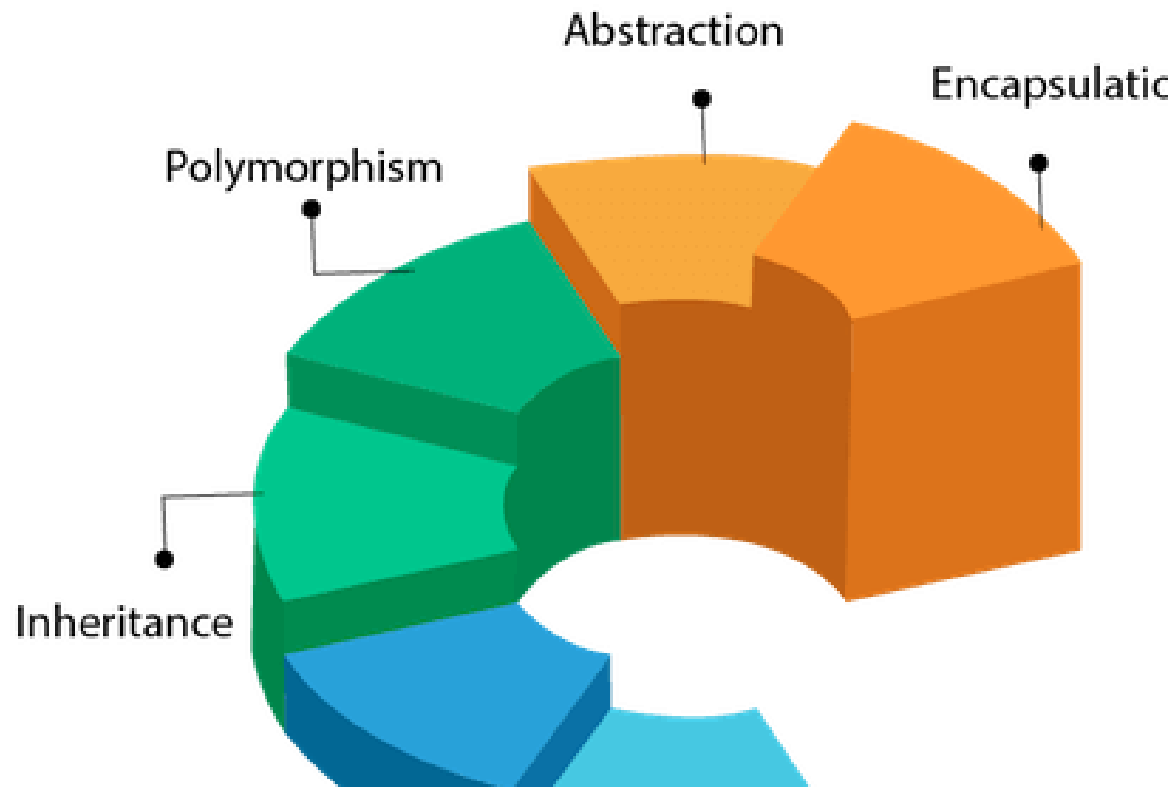
```
def ch(m):  
    m+=2;  
    print(m);  
m=12; #global varibale  
ch(m)  
print(m); # global variable print
```

**->Python Built-in Functions**

**Open ->python bultIn Function .doc file**

Like other general purpose languages, python is also an object-oriented language since its beginning. Python is an object-oriented programming language. It allows us to develop applications using an Object Oriented approach. In Python, we can easily create and use classes and objects.

# OOPs (Object-Oriented Programming)



## Difference Between Object Oriented Programming Language and Procedural Programming Language

Object-oriented programming is the problem-solving approach and used where computation is done by using objects.

Procedural programming uses a list of instructions to do computation step by step.

## 1&2. Object and Class and Constructor and ststic and self keyword

Object:-> object is Real world entity, It can be physical or logical. Like A bulldog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

Class:-> *Collection of objects* is called class. It is a logical entity. Like Dog, under Dog is bulldog,,pug etc... Class doesn't consume any space.

### How to create class and object in Python :->

class Dog:

*# State/attribute*

name= "abc"

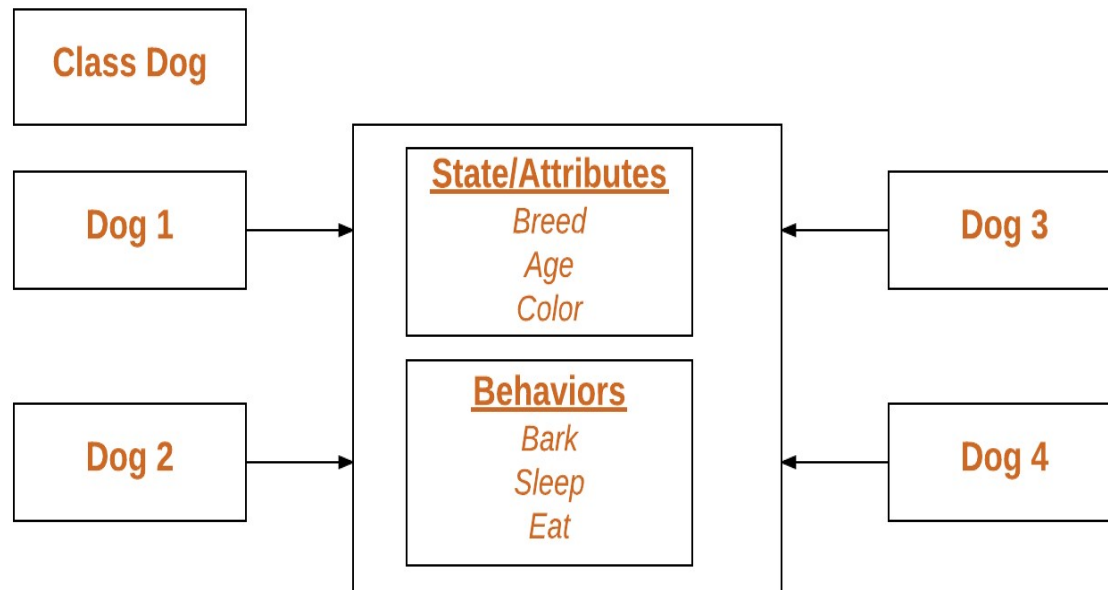
*#Behavior*

**def** eat(self):

print("chkn");

BulDog=Dog(); *#object create*

BulDog.eat(); *#Method call*



**Constructor** -> A constructor is a special type of method (function) which is used to initialize the instance members of the class.

Constructors can be of two types.

- Parameterized Constructor
- Non-parameterized Constructor

## Creating the constructor in python

```
class Dog:
    def __init__(self):
        print("ok");
BulDog=Dog();
```

```
class Dog:
    def __init__(self,name):
        self.name=name;
        print(name);
BulDog=Dog("som");
```

## -> BuiltIn Class Attribute

1. `__dict__` It provides the dictionary containing the information about the class namespace.
2. `__doc__` It contains a string which has the class documentation
3. `__name__` It is used to access the class name. it is special variable like that c or Java main function type
4. `__module__` It is used to access the module in which, this class is defined.
5. `__bases__` It contains a tuple including all base classes.

**Example :-**

```
class Dog:
    'i am dog'
    def __init__(self):
        print("ok");
    def dis():
        print("ok2");
print(Dog.__dict__);
print(Dog.__doc__);
print(Dog.__name__);
print(Dog.__module__);
```

```
class Dog:
    'i am dog'
    def __init__(self):
        print("ok");
    def dis():
        print("ok2");
BulDog=Dog();
BulDog1=Dog();
BulDog2=Dog();
print(id(BulDog))
print(id(BulDog1))
print(id(BulDog2))
del BulDog;
#print(id(BulDog)) # delete object
```

## Static method

A static method can be invoked without the need for creating an instance of a class.

```
class Example:
    name = "Example" #static variable or class variable
    @staticmethod
    def static():
        print(Example.name);
Example.static();
```

```
class aa:
    def det(self,name):
        self.name=name;

    def show(self):
        print(self.name);
s=aa();
s.det("som");
s.show();
```

```
class aa:
    def det(self,name):
        self.name=name; #name is global variable automatically
    def call(self):
        print("i am self call")

    def show(self):
        self.call(); #current class method
        print(self.name);
s=aa();
s.det("som");
s.show();
```

## Self keyword

Self is a **reference variable** that refers to the current object.

## 3.Inheritance

**What-**→Reusable code that means Parent –Child Relationship (IS-A )

**Features:->1. Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.

**2. Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

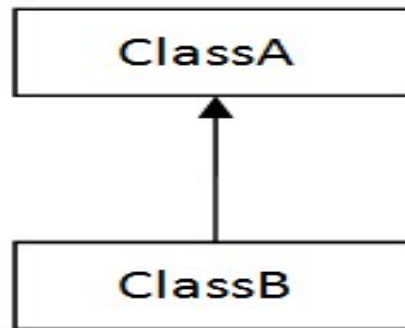
**How to crete Inheritance:-**

```
class aa:
    def __init__(self):
        print("simple constructor");
class aa2(aa): # inherit aa to aa2
    def get(self):
        print("i am get");

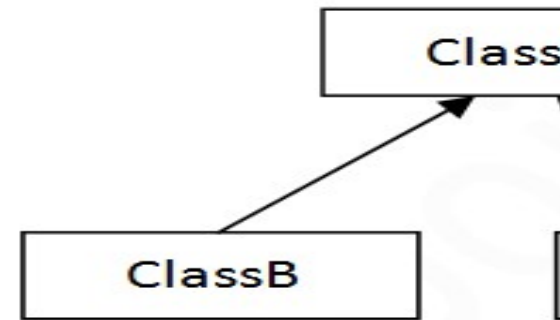
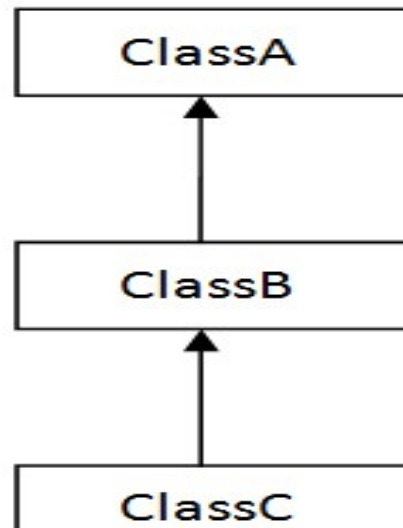
s=aa2();
```



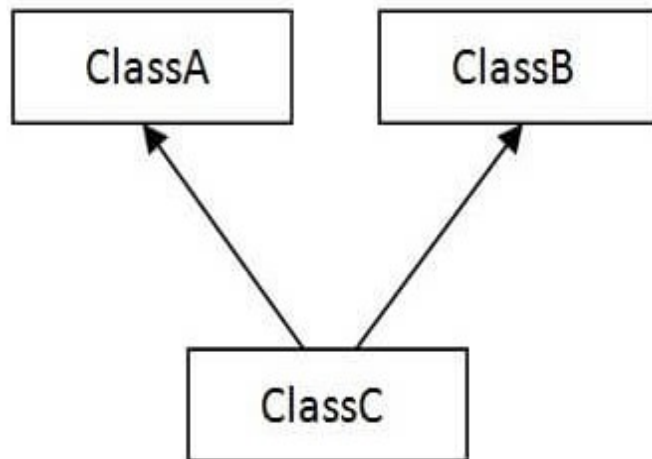
- Types of Inheritance



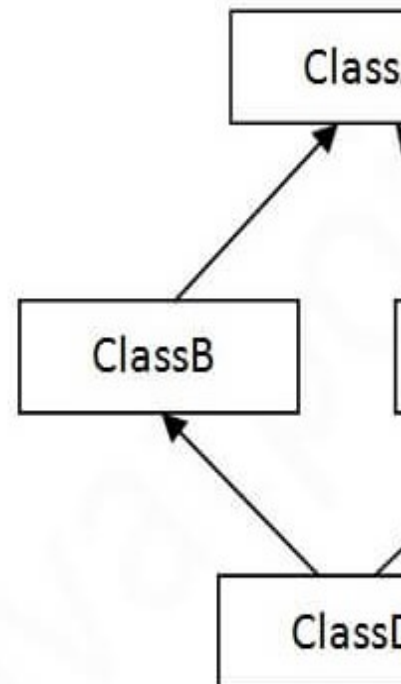
1) Single



3) Hierarc



4) Multiple



## 3.1 Aggregation in Python(HAS-A)

if a class have an entity reference, it is known as Aggregation

```
class Salary:
    def __init__(self, pay):
        self.pay = pay

    def get_total(self):
        return (self.pay * 12)

obj_sal = Salary(600); # // it is aggregation ,, bcz not possiable to relationship between Employee and
Salary (Salary contain many attribute like get_total etc)
class Employee:
    def __init__(self, pay, bonus):
        self.pay = pay
        self.bonus = bonus

    def annual_salary(self):
        return "Total: " + str(obj_sal.get_total() + self.bonus) #it is aggregation return get_total

obj_emp = Employee(obj_sal, 500)
print(obj_emp.annual_salary())
```

Why-?

Employee HAS a Salary

Address

**4.Polymorphism->Polymorphism** is a concept by which we can perform a *single action in different ways*.

In **Python, Polymorphism** allows us to define methods in the child class with the same name as defined in their parent class

## 4.1 Duck Typing

## 4.2 Operator Overloading

## 4.3 Method Overloading

## 4.4 Method Overriding

## 4.1 Duck Typing ->

Does not matter what type of object pass

```
class Duck:
    def walk(self):
        print("Duck");
class Hourse:
    def walk(self):
        print("Hourse");

def fun(obj): # does not matter what type object you pas
    obj.walk();
s=Duck();
fun(s); # pass object
s1=Hourse();
fun(s1); # pass object
```

## 4.2 Operating Overloading ->

```
class emp:
    def __init__(self,sal):
        self.sal=sal;

    def __add__(self, other): # + operator override
        return self.sal + other.sal
s=emp(12);
s1=emp(13);
print(s+s1)
```

## 4. Polymorphism->

**4.1 Method Overloading** -> python does not supports method overloading, We may overload the methods but can only use the latest defined method.

```
class Salary:
    def get(self):
        print("ok")
    def get(self,name):
        print(name);
s=Salary();
#s.get(); # show error
s.get("name");
```

**4.2 Method Overriding->**

```
class Salary:
    def get(self):
        print("ok")

class Employee(Salary):
    def get(self): #override method
        print("ok2");
s=Employee();
s.get();
```

## 4.3 Super Keyword

```
class Salary:
    a=20;
    def __init__(self):
        print("hello i am Salary Constructor")
    def get(self):
        print("ok")

class Employee(Salary):
    a=50;
    def __init__(self):
        super().__init__(); # super through parent
        # constructor call
        print("hello i am Employee Constructor")
    def get(self): #override method
        super().get(); #parent function call
        print("ok2");
        print(self.a);
        print(super().a); #parent variable call

s=Employee();
s.get();
```

## 4.4 Property keyword -> As like final class

```
@property
class aa:
    def get(self):
        print("ok");
class aa2(aa):
    def get2(self):
        print("ok2");
s=aa2(); # show error cz aa not inherit
```

```
class aa:
    @property
    def get(self):
        print("ok");
    def gett(self):
        print("i am gett")
class aa2(aa):
    def get(self):
        #super.get(self);
        print("oooo");

    def get2(self):
        print("ok2");
s=aa2();
s.get(); # show error not override
```

Note-> There are no constant value in python , if you want a define constant variable in python so use CAPITAL variable name like "PI=3.14"

# Encapsulation

## 5.1 Access Modifiers

A Class in Python has three types of access modifiers –

- **Public Access Modifier**
- **Protected Access Modifier**
- **Private Access Modifier**

**How to Declare Protected and Public and Private**

```
class aa:
    #private variable
    __roll=None;
    #Protected variable
    _name=None;
    #public function
    def get(self):
        #public variable
        print("i am public function ")
    #protected Function
    def _get2(self):
        print("i ma protected function")
    #private Function
    def __get3(self):
        print("i am private function ")
```

**Private:** The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

**Protected:** The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

**Public:** The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

**Note:-> Python class are public by default.**

**Data Encapsulation** -> The process of binding data and corresponding methods (behavior)

together into a single unit is called **encapsulation**.

In other words, encapsulation is a programming technique

that binds the class members (variables and methods) together

and prevents them from being accessed by other classes, thereby we can keep variables and methods safe from outside interference and misuse.

```
class AA:
    __act=None;
    def banlc(self):
        self.__act=5000;
        print(self.__act);

s=AA();
s.banlc();
```

## Encapsulation

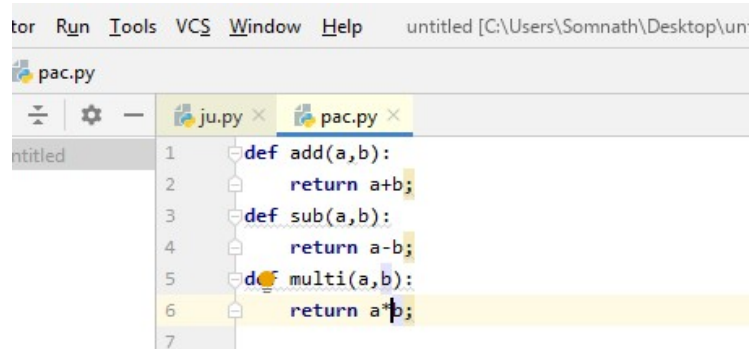


## 6.5 Python Modules:->

Consider a situation, you have to create a project that project name is ABCD it combination of A,B,C and D that time programmer divide to Module like A module , if you want to change something to A module does not effect B,C and

D Module .one module mean one file(.py) like that Java Package

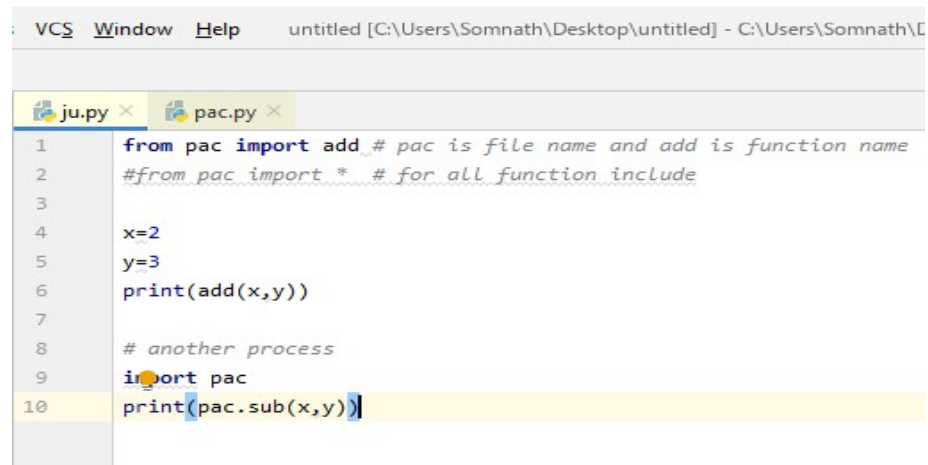
### Create a Module :-



```
def add(a,b):  
    return a+b;  
def sub(a,b):  
    return a-b;  
def multi(a,b):  
    return a*b;
```

The screenshot shows a code editor with two tabs: 'ju.py' and 'pac.py'. The 'pac.py' tab is active, displaying the code for the module. The code defines three functions: 'add', 'sub', and 'multi'. The 'add' function takes two arguments 'a' and 'b' and returns their sum. The 'sub' function takes two arguments 'a' and 'b' and returns their difference. The 'multi' function takes two arguments 'a' and 'b' and returns their product.

### Use Module :->



```
from pac import add # pac is file name and add is function name  
#from pac import * # for all function include  
  
x=2  
y=3  
print(add(x,y))  
  
# another process  
import pac  
print(pac.sub(x,y))
```

The screenshot shows a code editor with two tabs: 'ju.py' and 'pac.py'. The 'ju.py' tab is active, displaying the code that uses the 'pac.py' module. The code imports the 'add' function from the 'pac' module and prints the result of 'add(2,3)'. It also shows an alternative way to use the module by importing the entire module and then accessing the 'sub' function.

### -> How to install pre-define Module

1. Set path or this path open through cmd C:\Users\EASY\AppData\Local\Programs\Python\Python39\Scripts
2. Open cmd
3. Pip install (your module name)



## 6.1 Abstraction

**Abstraction** is a process of hiding the implementation details and showing only functionality to the user.

How to create Abstract class and Abstract Method ->

```
from abc import ABC, abstractmethod
class Ab(ABC):#Abstract class
    @abstractmethod
    def get(self):
        pass;
    def get2(self): #non aBstract
method
        print("get2");
class child(Ab):
    def get(self):
        print("ok");

s=Ab(); # not possiable to create
object
s.get2();
```

```
# Real Life Example of Abstraction
from abc import ABC,abstractmethod
class RBank:
    @abstractmethod
    def Rate(self):
        pass;

class Sbi(RBank):
    def Rate(self):
        return int(7)

s=Sbi();
a=s.Rate();
print(a);
```

## **6.1 Interface**

Interface contain only Abstract Method

```
from Interface import implements, Interface
```

```
class MyInterface(Interface):
```

```
    def method1(self, x):
```

```
        pass
```

```
    def method2(self, x, y):
```

```
        pass
```

```
class MyClass(implements(MyInterface)):
```

```
    def method1(self, x):
```

```
        return x * 2
```

```
    def method2(self, x, y):
```

```
        return x + y
```

```
s=MyClass();
```

```
a=s.method1(2);
```

```
print(a);
```

## Python Array :->

Array is mutable for Python ,Python does not support MultiDimensional Array using Third Party NumPy Library

Exmple :-

```
from array import *;
```

```
# a=array("12",14,"io"); # array pass two argument 1. type of  
array 2.element
```

```
a=array("i",[1,2,3,9]) # i = integer
```

```
print(a[0]);
```

```
print(a[-1]);
```

```
a[0] = 65      # Update 0 index value
```

```
print(a[0])
```

```
# changing/replace 2 index to 4th index
```

```
a[2:5] = array('i', [4, 6, 8])
```

```
print(a)
```

```
del a[2]          # removing 2 index
```

```
#length return
```

```
print(len(a));
```

```
print(a.index(8)); # search value 8
```

```
# Type Define Array  
from array import *;  
#Signed int value  
a=array("i",[12,-15,89]); #positive and negative value support  
#unsigned int value  
#b=array("I",[12,-15,89]); # only positive value support  
#float value  
c=array('f',[12.5,58.5,78])  
#double value  
d=array('d',[12.5,58.5,788.89])  
#char value  
e=array('c',['a','b','c']);  
  
print(a[1]);  
#print(b[1]); # unsigned int show error  
print(c[1]);  
print(d[1]);  
print(e[1]);
```

## 6.2 Python Lambda Functions:-

Python allows us to not declare the function in the standard manner, by using the def keyword. Rather, the anonymous functions are declared by using lambda keyword. However, Lambda functions can accept any number of arguments, but they can return only one value in the form of expression.

**Syntax:**

**lambda** arguments : expression

```
x=lambda x:x+50;
print(x(10));

y=lambda a,b:a+b;
print(y(10,20));
```

```
l = {1,2,3,4,10,123,22} # find odd and even number
even = list(filter(lambda x:(x%2 == 0),l)) # the l contains all the items of the list
for which the lambda function evaluates to true
print(even)
```

```
def multi(n):
    return lambda a:a*n;
b=multi(10);
#print(b);
for i in range(1,10):
    print(b(i)) # the l contains all the items of the list for which the lambda
function evaluates to true
#print(even)'''
```

## Writing a file:->

```
# open the test.txt in append mode. Creates a new file if no such file exists.
f = open("D:\\test.txt", "a");

# appending the content to the file Update my test.txt file
f.write("I want to learn Python.")

# closing the opened file
f.close();

# open the test1.txt in write mode.
f1 = open("D:\\test1.txt", "w");

# overwriting/Replace the content of the file
f1.write("I want to Learn Python ")

# closing the opened file
f1.close();
```

## Creating a new file:-

```
f = open("D:\\file2.txt", "x");
print(f)
if f:
    print("File created successfully");
#Using with statement with files
with open("D:\\test.txt", 'r') as f:
    c = f.read();
    print(c)
```

## 6.3 Python File Handling

Open a file for Read mode:->

```
# opens the file file.txt in read mode
f = open("D:\\test.txt", "r")

if f:
    print("file is opened successfully")
f.close();
```

### Reading a file

```
# open the file.txt in read mode. causes error if no such file exists.
f = open("D:\\test.txt", "r");

# stores all the data of the file into the variable content
c = f.readline();
# prints only one line
print(c)

f1=open("D:\\test1.txt","r");
for i in f1:
    print(i) # i contains each line of the file

# closes the opened file
f.close()
```

## 6.4 Python os module:->

The os module provides us the functions that are involved in file processing operations like renaming, deleting, etc.

```
import os;
# rename file2.txt to file3.txt
os.rename("D:\\test.txt", "D:\\raja.txt")
#deleting the file named file3.txt
os.remove("D:\\raja.txt")
#creating a new directory with the name new

os.mkdir("D:\\new")
# changing the current working directory to new
os.chdir("D:\\new")
#printing the current working directory
print(os.getcwd())
#removing the new directory
os.rmdir("D:\\new")
#Open Notepad exe file
import subprocess as sp
p = "notepad.exe"
f2 = "D:\\raja.txt" # raja.txt file open through Notepad
sp.Popen(p)
sp.Popen([p,f2]);
```

## 6.6 Python Exceptions:-

### Error :->

- \_1. Compile Time Error(syntax mistakes)
- 2. Logical Error (Wrong value Output )
- 3.RunTime Error (a=7/0 )

An exception can be defined as an abnormal condition in a program resulting in the disruption in the flow of the program.

### Common Exceptions:-

- **ZeroDivisionError:** Occurs when a number is divided by zero.
- **NameError:** It occurs when a name is not found. It may be local or global.
- **IndentationError:** If incorrect indentation is given.(Space error )
- **IOError:** It occurs when Input Output operation fails.
- **EOFError:** It occurs when the end of the file is reached, and yet operations are being performed.



```

# Divided by Zero
def zero():
    try:
        a = 6 / 0;
        print(a);
    except Exception:
        print("divided by zero");

# File Not Found Exception
def file():
    try:
        # this will throw an exception if the file doesn't exist.
        f = open("D:\\file.txt", "r")
    except IOError:
        print("File not found")
    # Syntax Complete or Not

# Variable Define or Not
def name():
    try:
        x += 1;
        print(x);
    except:
        print("variable not define/ name error ")

# Finally Example
def fil():
    try:
        a = 5 / 0;
    except Exception:
        print("Exception occurred")
    finally:
        print("thank you");

s = zero();
s1 = file();
s2 = eof();
s3 = name();
s4 = fil();

```

```

# Custom Exception
class ab(Exception):
    """Base class for other exceptions"""
    pass

a=9;
if(a>0):
    raise ab("sry your value") # its xustom exception

```

**What is the difference between exception and Except Exception**

except:" handles all exceptions.

"except Exception:" handles exceptions derived from the Exception class.

Example:->

```

try:
    b=4/0
except IOError: # this class belongs to the Exception class
    print("dev")
except:
    print("Error!!")

```

- **Python Time and Date**

```
import time;

import datetime;

import calendar;

#returns a time tuple

print(time.localtime(time.time()))

#returns the formatted time

print(time.asctime(time.localtime(time.time()))))


for i in range(0,5):

print(i)

#Each element will be printed after 1 second

time.sleep(1)

#Consider the following example to get the datetime object representation for the current Date and time.

print(datetime.datetime.now())

#calender Show

cal = calendar.month(2020,2)

#printing the calendar of December 2020


print(cal)

#printing the calendar of the year 2020

calendar.prcal(2020)
```

- Python Regular Expressions

## Why We Use Regular Expressions?




How to find the date and time from the log file?

Jul 1 03:27:12  
syslog:  
[m\_java][  
1/Jul/2013  
03:27:12.818][j:  
[SessionThread  
<]^lat  
com/avc/abc/m  
agr/service/find  
.something(abc  
/1235/locator/a  
bc;Ljava/lang/S  
tring;)Labc/abc/  
abcd/abcd;(byt  
ecode:7)

A blue arrow points from the man's question to a log file snippet. The log file snippet contains a date and time that are highlighted in blue and underlined, indicating they have been successfully extracted using a regular expression.

## Why We Use Regular Expressions?



Email or Phone Password Log In  
Forgotten account?

Sc # .com  
sk@gmail.com  
dk@gmail.com  
sd@gmail.com  
Cf @ cm

How to verify these e-mail addresses?

A blue arrow points from the login form to a list of email addresses. The list contains several email addresses, some of which are highlighted in blue and underlined, indicating they have been successfully verified using a regular expression.

```
import re
import re
import datefinder
from datetime import datetime

#regular expression through find number

s = 'My age is 30 and my fnd age is 32'
lst = re.findall('[0-9]+', s) # + for repeats a character one or more times
print(lst);
# find Email
s = 'Hello my email is rajubhadraa@gmail.com to som@ yahoo.com and my fnd email @2PM'
lst = re.findall('\S+@\S+', s) # \s means not allow white space
print(lst);
#capital later find
s="hello I am somnath BHADRA";
lst=re.findall('[A-Z]+',s);
print(lst);

# Date Find
s="my date of birth is 09-06-1991 my fnd age is 09-06-1992"
x=re.search("([0-9]{2}\-[0-9]{2}\-[0-9]{4})", s)

print(x.group(1));
#more than one date find
matches = datefinder.find_dates(s)
for match in matches:
    print(match)
```

## Python read/Write csv file:->

A csv stands for "comma separated values", which is defined as a simple file format that uses specific structuring to arrange tabular data.

We have taken a txt file named as ab.txt that have default delimiter comma(,) with the following data:

```
# Read CSV File
import csv

with open('D:\\ab.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    for row in csv_reader:
        print(row); # show data by default List
    #print(row[0])
```

```
#write CSV File
import csv

with open('D:\\ab.csv','r') as csv_file:
    csv_reader = csv.reader(csv_file)

    with open('D:\\ab1.csv','w') as new_file:
        csv_writer = csv.writer(new_file, delimiter="-") # delimiter sseparate the
column with "-" this symbol
        for row1 in csv_reader:
            csv_writer.writerow(row1)
```

## Python Assert Keyword:->

Python assert keyword is defined as a debugging tool that tests a condition. The Assertions are mainly the assumption that asserts or state a fact confidently in the program.

**Syntax :-**

**assert** condition, error\_message(optional)

**Example :**

```
def raju(age):  
    assert age>=18, "assert error vote age only 18"  
    print(age);  
print("i am next line ") # if assert condition true do not  
execute this line  
raju(17);
```

## Python Collection Module:->

The Python collection module is defined as a container that is used to store collections of data, for example, list, dict, set, and tuple, etc

```

#collection : Counter,namedtuple,orderDict,defaultdict,deque
from collections import Counter
from collections import namedtuple
import collections
#Counter Example
a="aaaabbbccc";
my_c=Counter(a);
print(my_c); # how many times a or b or c
print(my_c.most_common(1)) # heighest character contain value return
# namedtuple Example
# Declaring namedtuple()
Student = collections.namedtuple('Student', ['name', 'age', 'DOB'])
# Adding values
S = Student('somnath', '30', '09/06/1991')
# Access using index
print("The Student age using index is : ", end="")
print(S[1])
# Access using name
print("The Student name is : ", end="")
print(S.name)
# defaultdict
d= collections.defaultdict(int); # default value
d['a']=1
d['b']=1
print((d['c'])); # defalut value 0
# Deque collection
d=collections.deque()
d.append(1)
d.append(2)
d.append(3)
d.append(2)
print(d)
d.appendleft(5) # Left position value add
print(d)
d.insert(1,10) # 1 no position entry 10
print(d)
print (d.index(2,0,5)) # 2 find between 0 to 5
# count number
print (d.count(2)) # how many times 2
# remove value
d.remove(2)
# pop()
d.pop() # de.popleft() ,
# extended
d.extend([4,5,6]) # add list at the end
print((d))

```