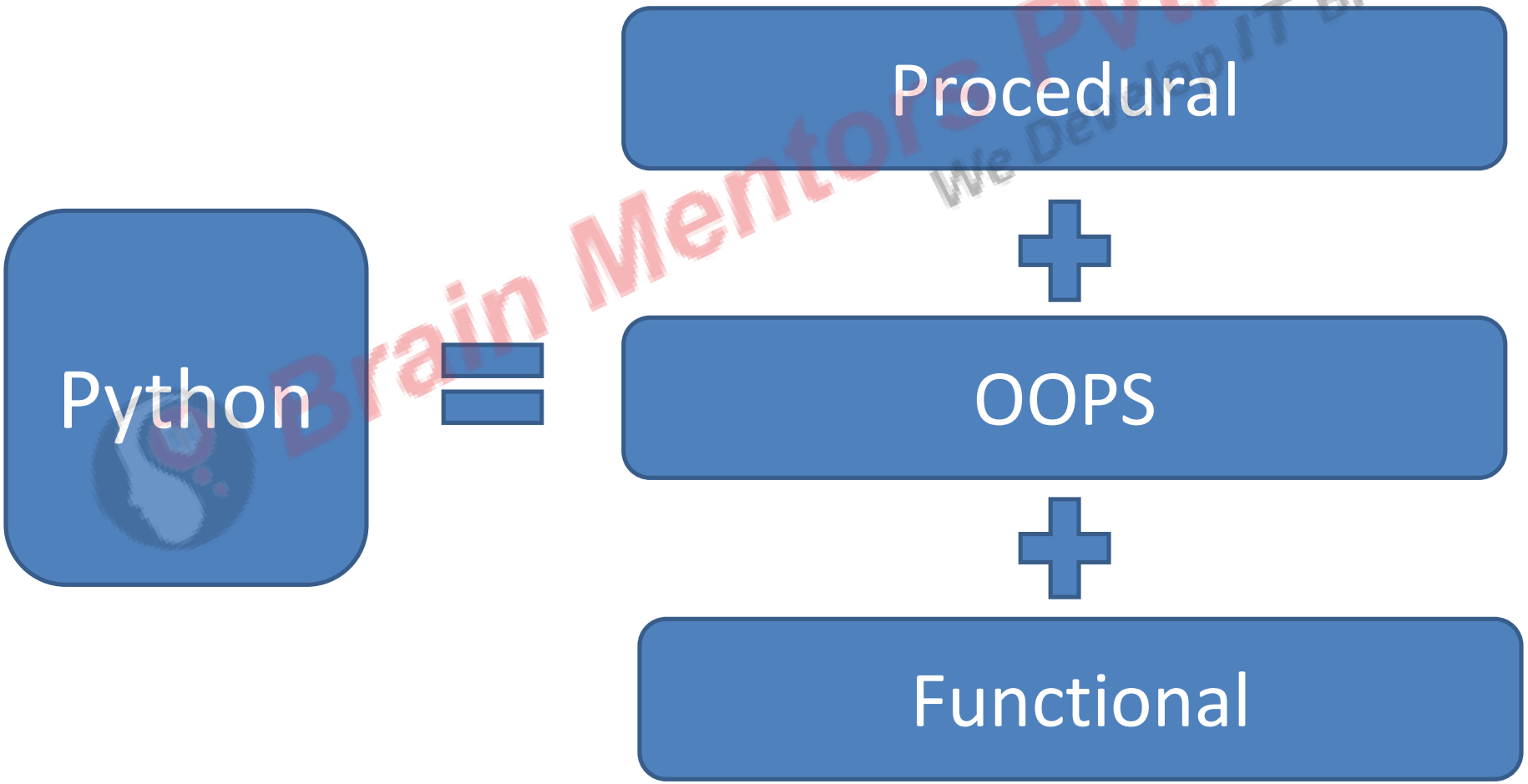
Brain Mentors Pvt. Ltd.
We Develop IT Brains

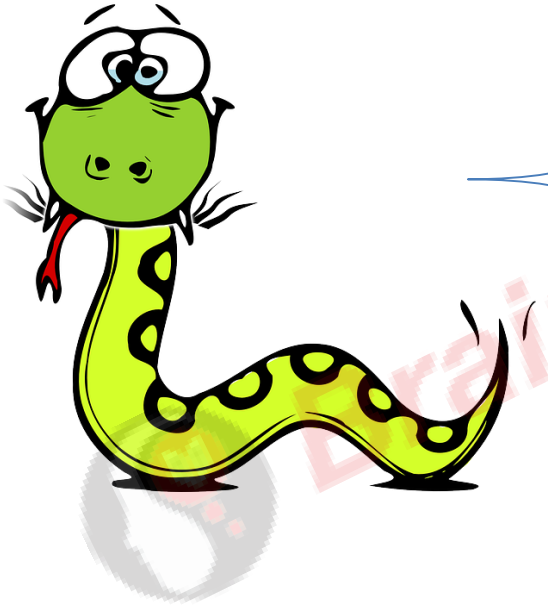
Python

What is Python…?

Python is a High level general purpose programming language.



MY FEATURES



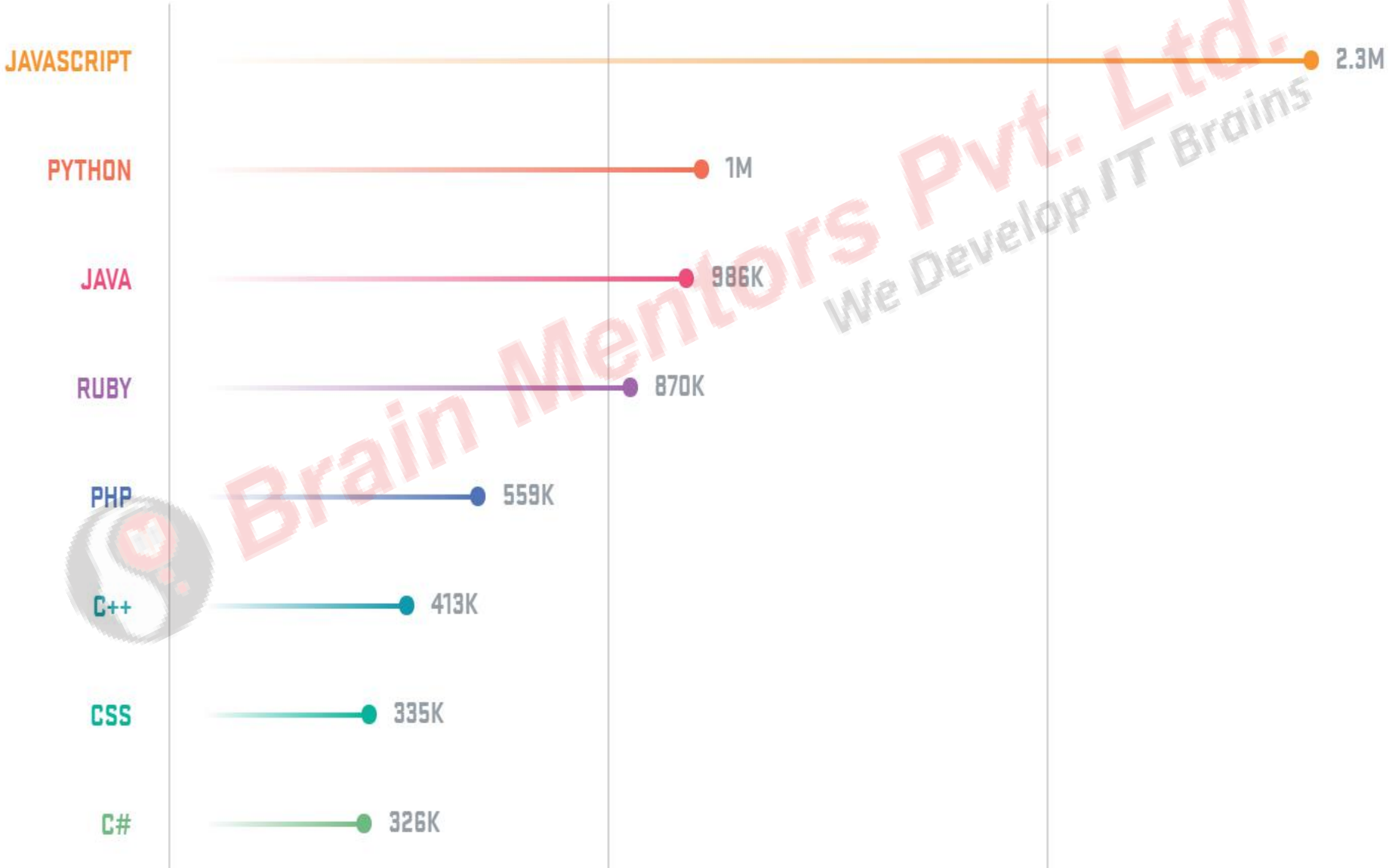
- General Purpose
- High Level
- Interpreted
- Dynamically Typed
- Multi-paradigm

HISTORY

- Guido Van Rossum invented Python in 1991.
- He named it Python because he was highly inspired by a show “Monty Python’s Flying Circus ”.
- Python was the successor of ABC Language

Guido van Rossum

POPULARITY OF PYTHON



COMPANIES USING PYTHON

PYTHON APPLICATIONS

GAME DEVELOPMENT

FACE RECOGNITION

WEB DEVELOPMENT

ROBOT PROGRAMMING

GAME DEVELOPMENT

FACE RECOGNITION

WEB DEVELOPMENT

ROBOT PROGRAMMING

JOBS IN PYTHON

The Ten Most Common Data Science Skills in Job Postings

Skill	Percentage of Job Listings
Python	**72%**
R	**64%**
SQL	**51%**
Hadoop	**39%**
Java	**33%**
SAS	**30%**
Spark	**27%**
Matlab	**20%**
Hive	**17%**
Tableau	**14%**

The Ten Most Common Data Science Skills in Job Postings

Skill	Percentage of Job Listings
Python	72%
R	64%
SQL	51%
Hadoop	39%
Java	33%
SAS	30%
Spark	27%
Matlab	20%
Hive	17%
Tableau	14%

HOW TO DOWNLOAD PYTHON

Download Python :
https://www.python.org/downloads/

Note :
in Mac and Linux , You have Python2 Pre-Installed already. So, after Installing python3 , you need to first check where is the Python3 by Using (which python3) command , Then alter the path for python3.

STEPS TO SET PATH IN MAC

1.

```
amits-MacBook-Air-2:~ amit$ which python3
/Library/Frameworks/Python.framework/Versions/3.7/bin/python3
```

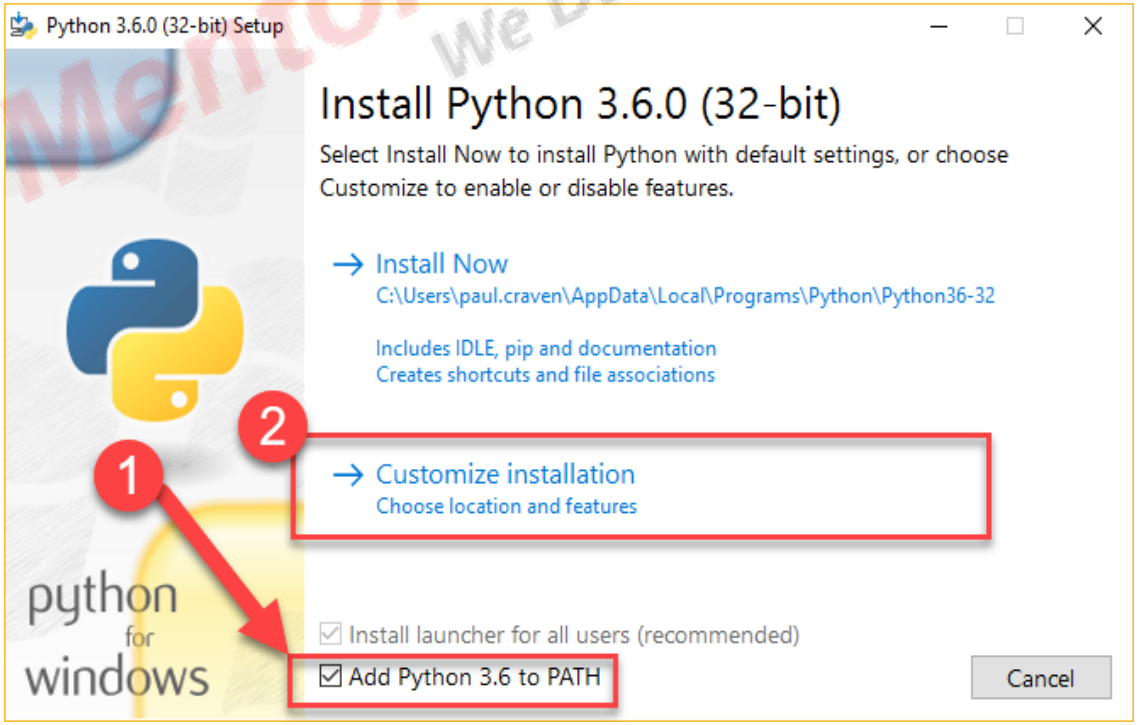
2.

nano .bash_profile

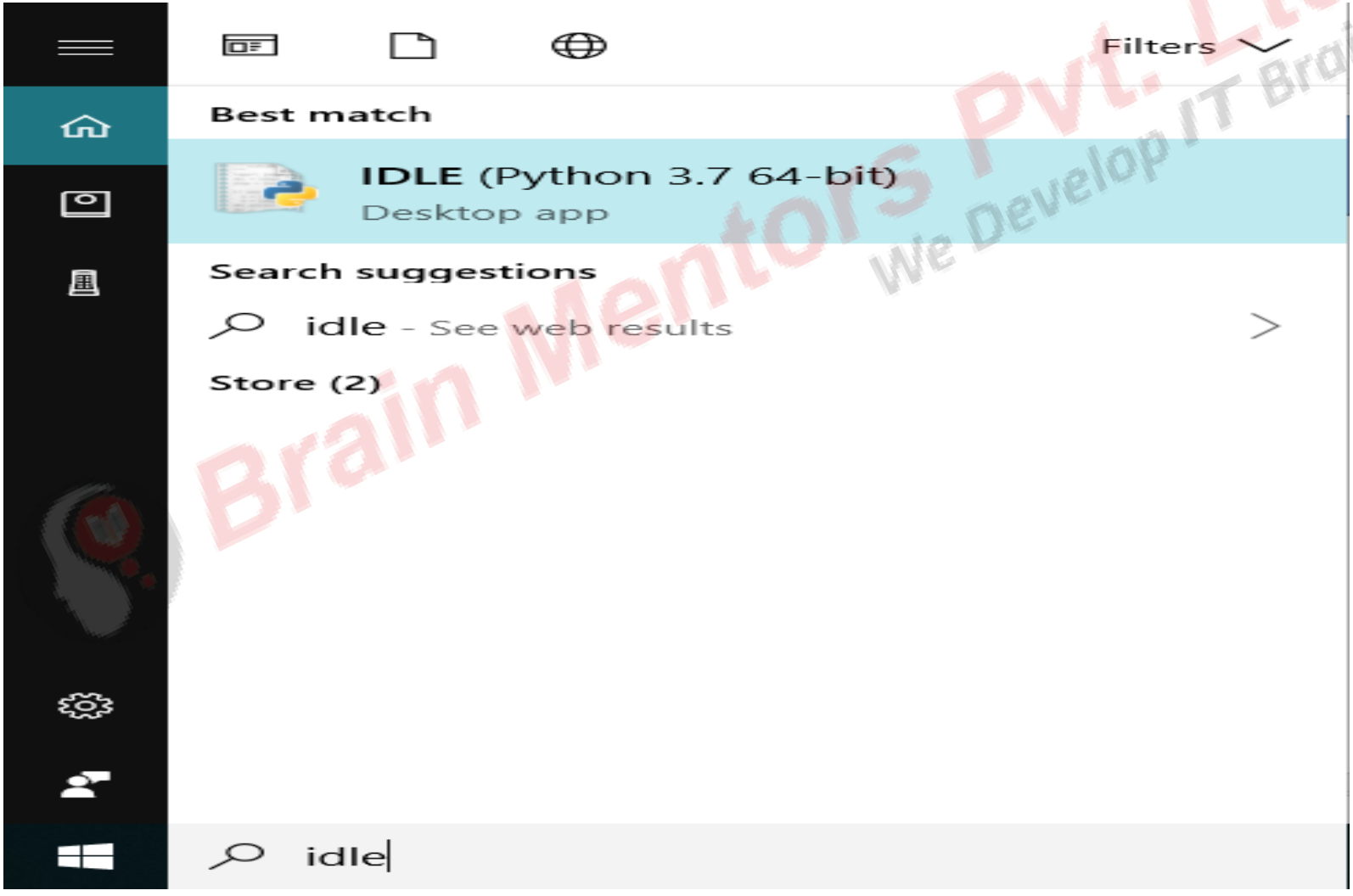
```
# Setting PATH for Python 3.7
# The original version is saved in .bash_profile.pysave
PATH="/Library/Frameworks/Python.framework/Versions/3.7/bin:${PATH}"
export PATH
alias python=python3
```

INSTALLATION IN WINDOWS

Note the following Steps and install it at correct destination folder As shown .



GET STARTED WITH PYTHON



PYTHON CODE EXECUTION

SOURCE

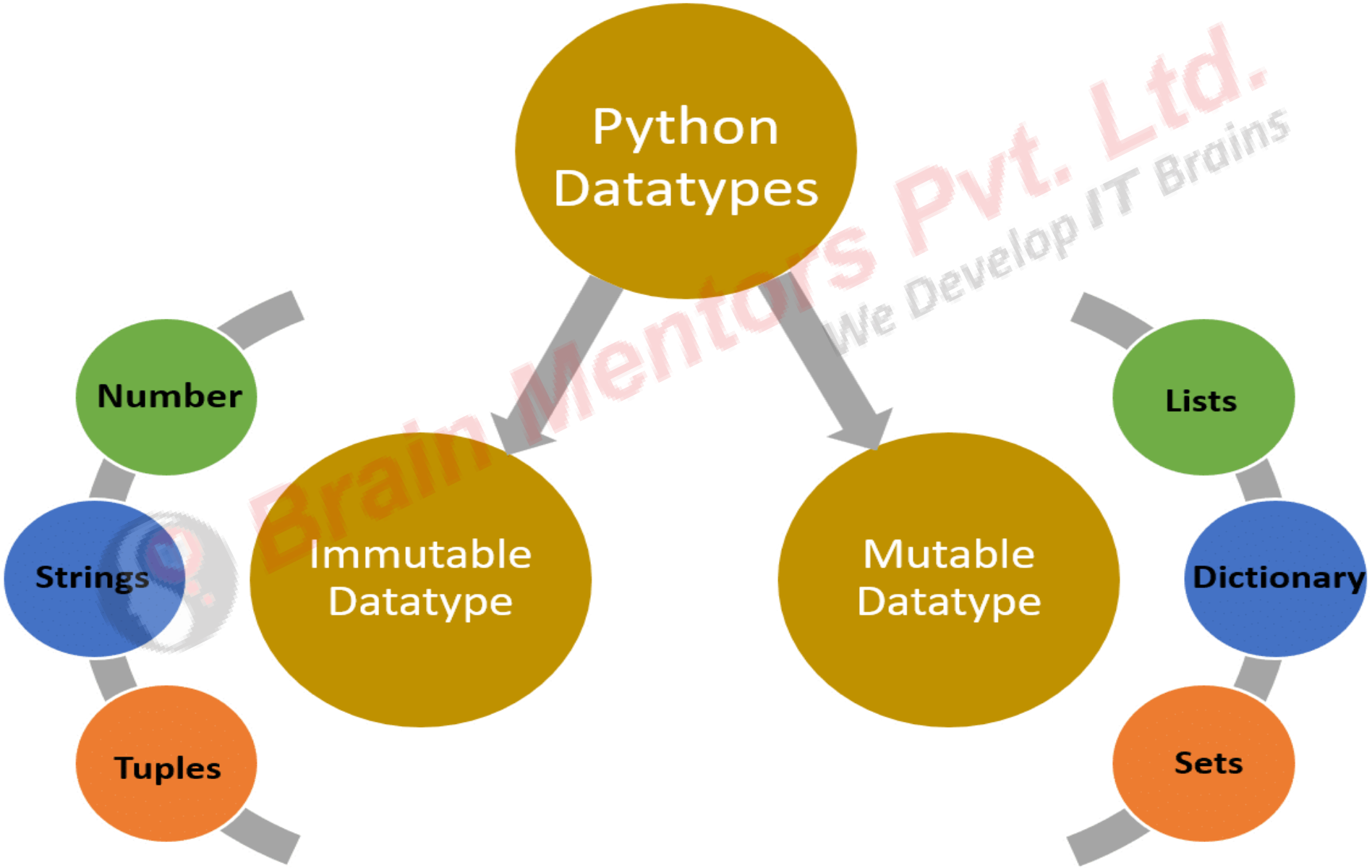
BYTE CODE

PVM

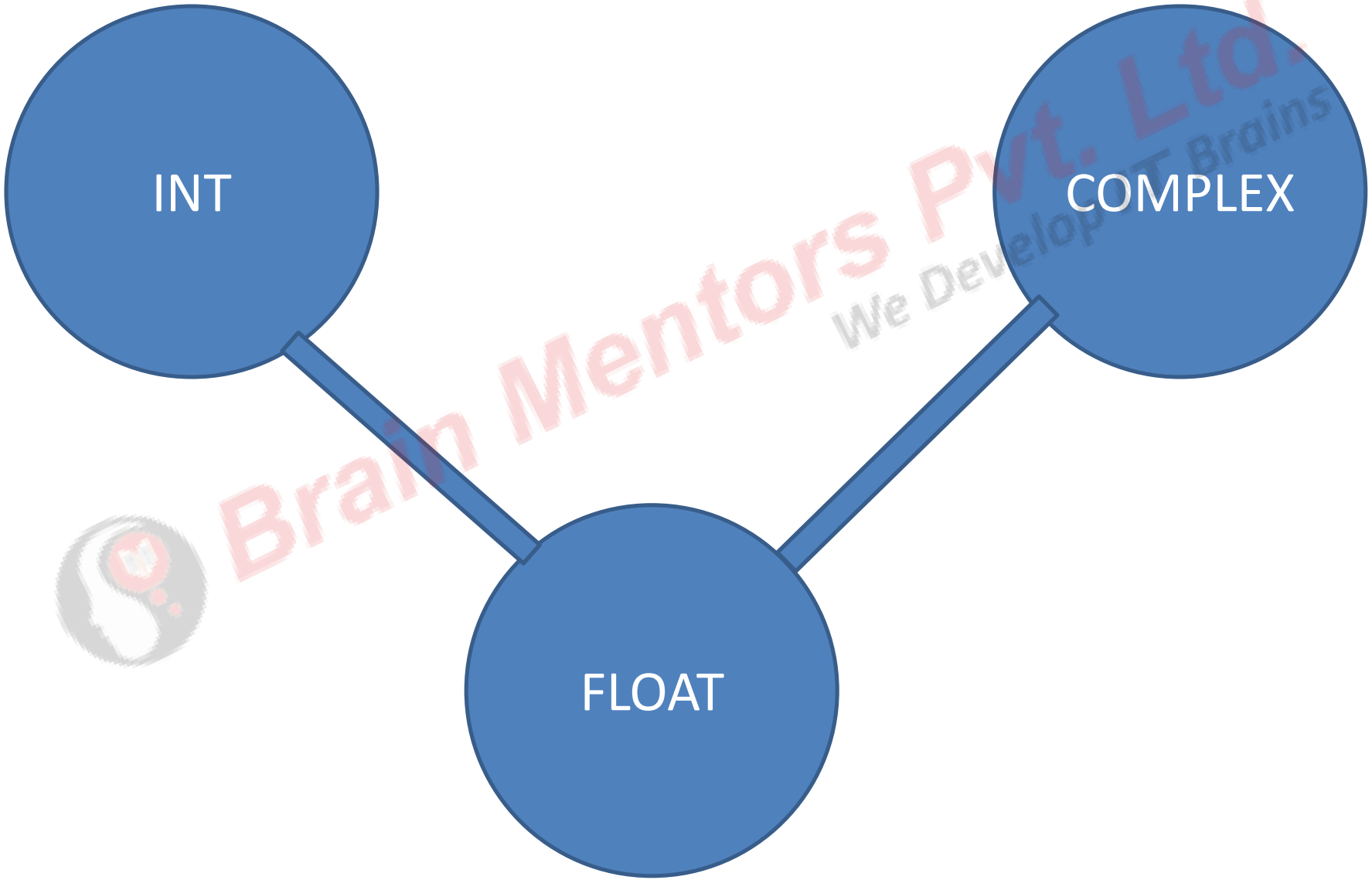
PYTHON PROGRAMMING's ABC

What is Identifier?

DATA TYPES



NUMERIC DATA TYPES

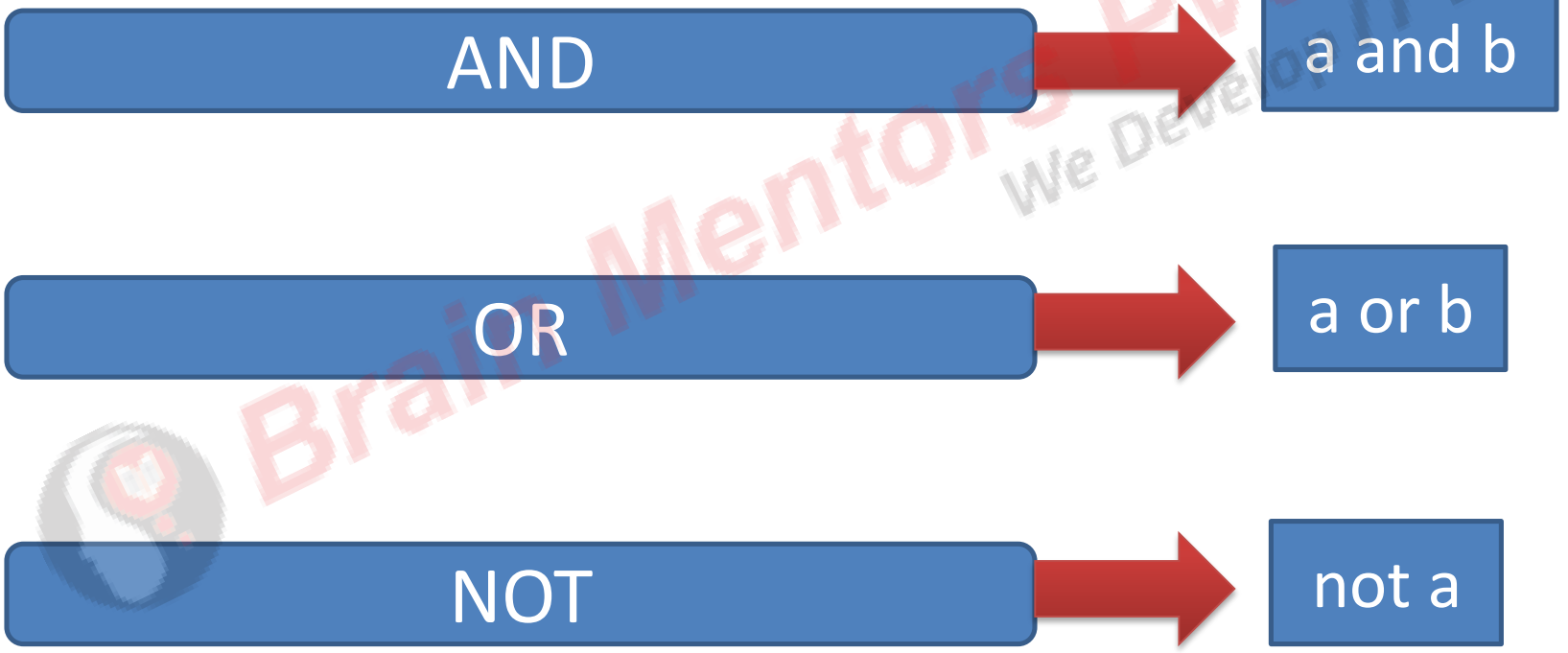


OPERATORS

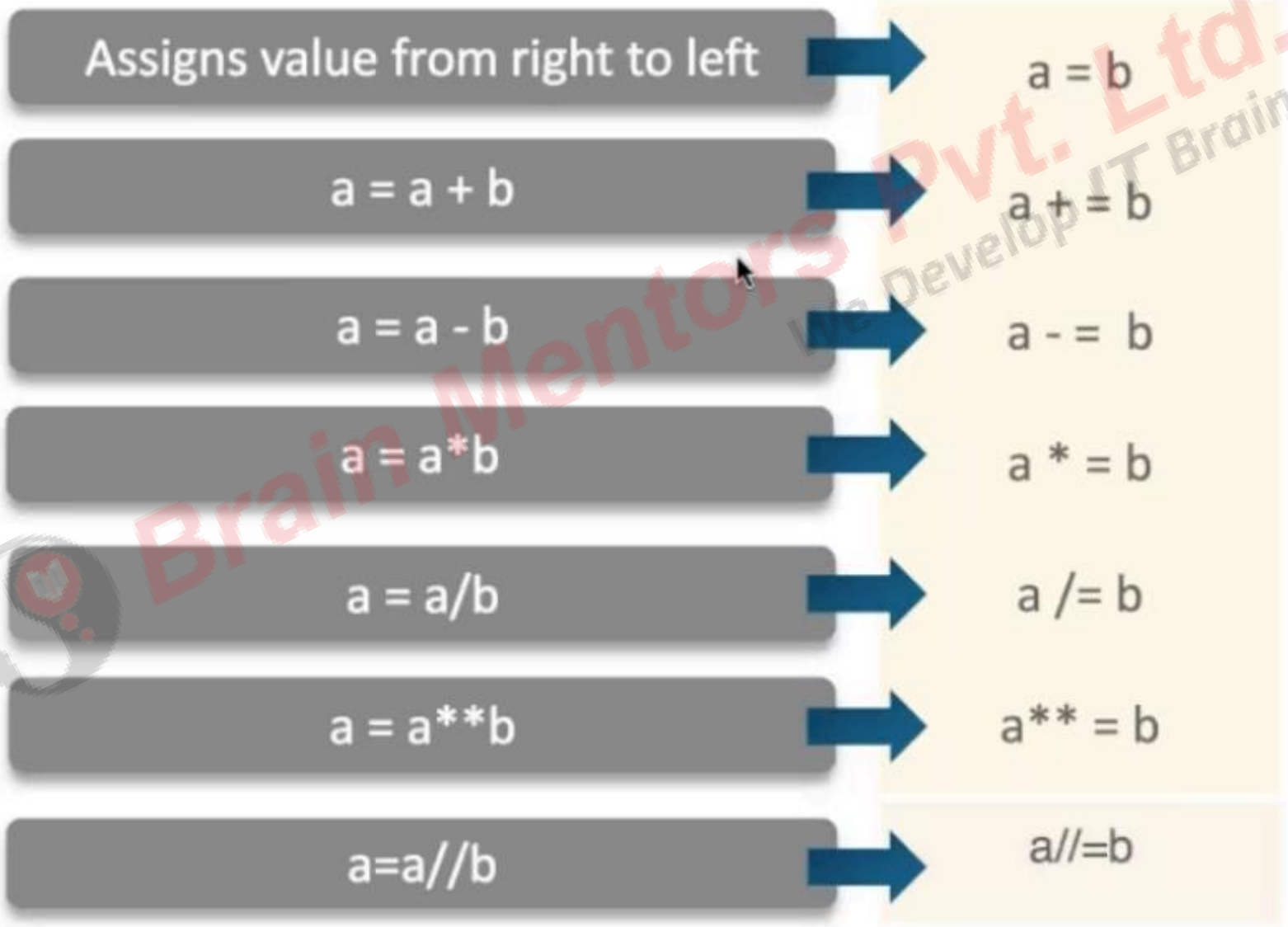
- Arithmetic Operator
- Logical Operator
- Assignment Operator
- Comparison Operator
- Identity Operator
- Membership Operator

Arithmetic Operator

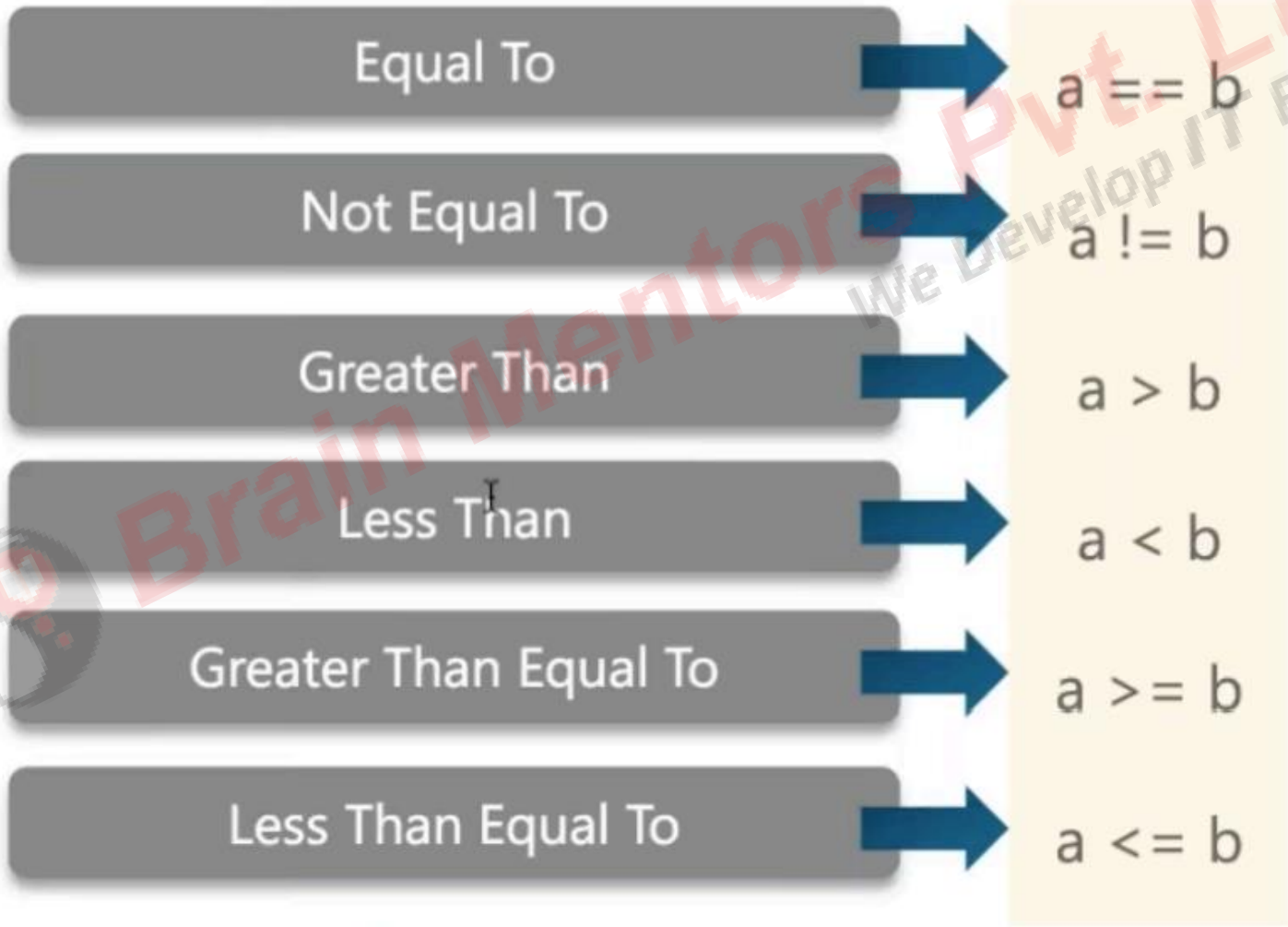
Logical Operator



Assignment Operator



Comparison Operator



Identity Operator

IS → Returns true if both variable points to same object

IS NOT → Returns false if both variable doesn't points to same object

```
>>> a1=5
>>> b1=5
>>> a2="python"
>>> b2="python"
>>> a3=[4,5,6]
>>> b3=[4,5,6]
>>> print(a1 is b1)
True
>>> print(a2 is not b2)
False
>>> print(a3 is b3)
False
>>>
```

Membership Operator

IN → Returns true if it finds a variable in specified sequence

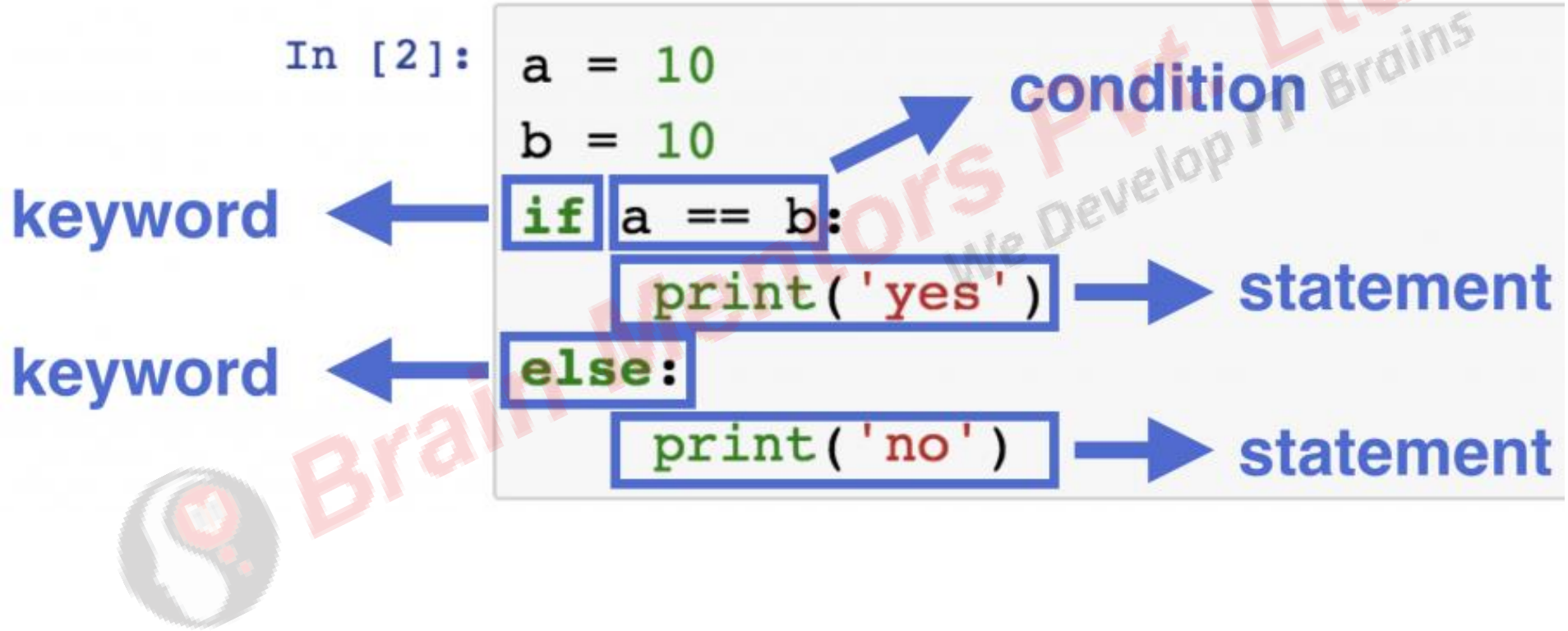
NOT IN → Returns false if it didn't finds a variable in specified sequence

```
>>> string1='sahilkumar'
>>> 'a' in string1
True
>>> 'k' in string1
True
>>> 'f' not in string1
True
>>> 's' not in string1
False
```

CONDITIONAL STATEMENT

if-else

The if-else statement is used to conditionally execute a statement or a block of statements based on required condition. Condition can be true or false, execute one thing when the condition is true, something else when the condition is false.



One Liner **if-else** Statement

```
i = 5 if a > 7 else 0
```

translates into

```
if a > 7:
    i = 5
else:
    i = 0
```

Note :

If you want to use multiple if-else conditions / nested if-else then, if-elif-else statement is used.

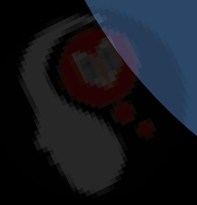
EXAMPLE :

In [79]:
```
a = False
b = True

if a:
    print("Now you will see a")
elif b:
    print("Now you will see b")
else:
    print("Now you will not see a or b")
```

```
Now you will see b
```

LOOP



A **loop** is a sequence of instruction s that is continually repeated until a certain condition is reached.

Why Loops are used…?

Syntax Comparison

Print * pattern triangle

```
print("*")
print("**")
print("***")
print("****")
print("*****")
print("******")
```

OUTPUT

LOOPS Comes to rescue

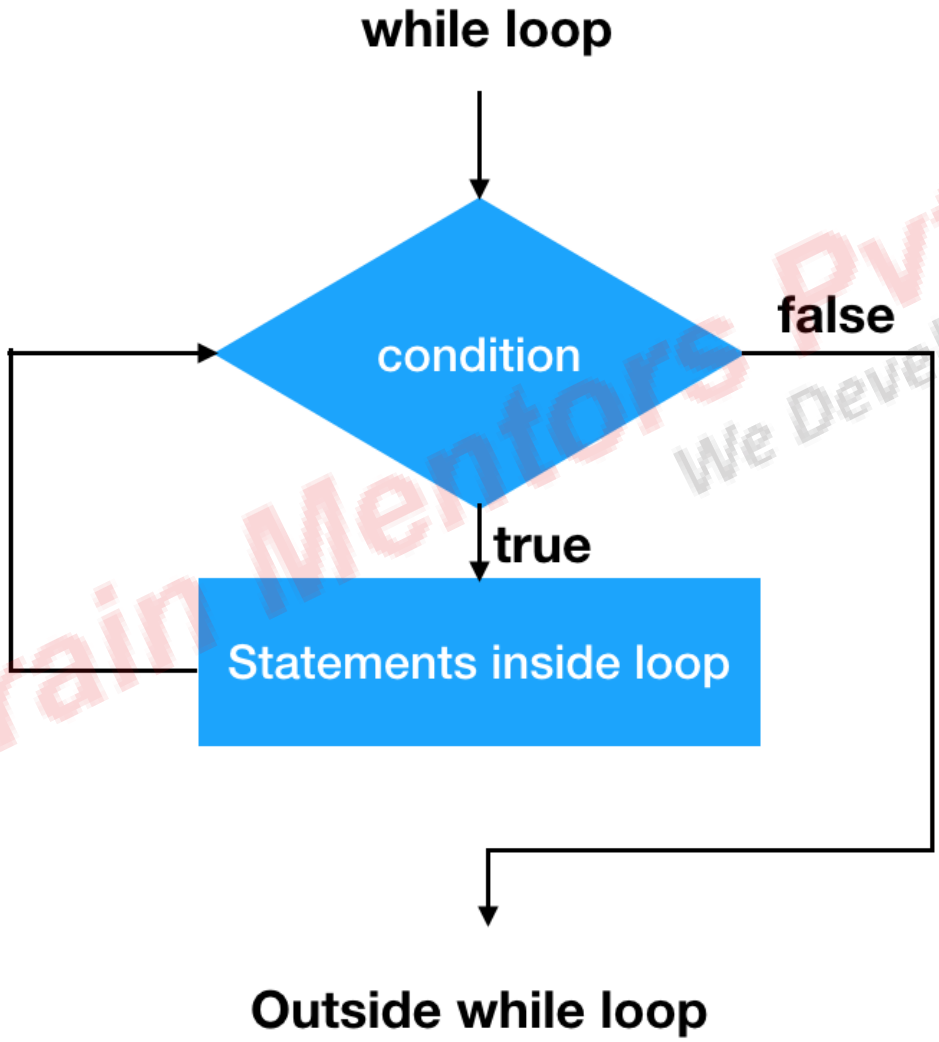
```
for i in range(1,7):
    print("*" * i)
```

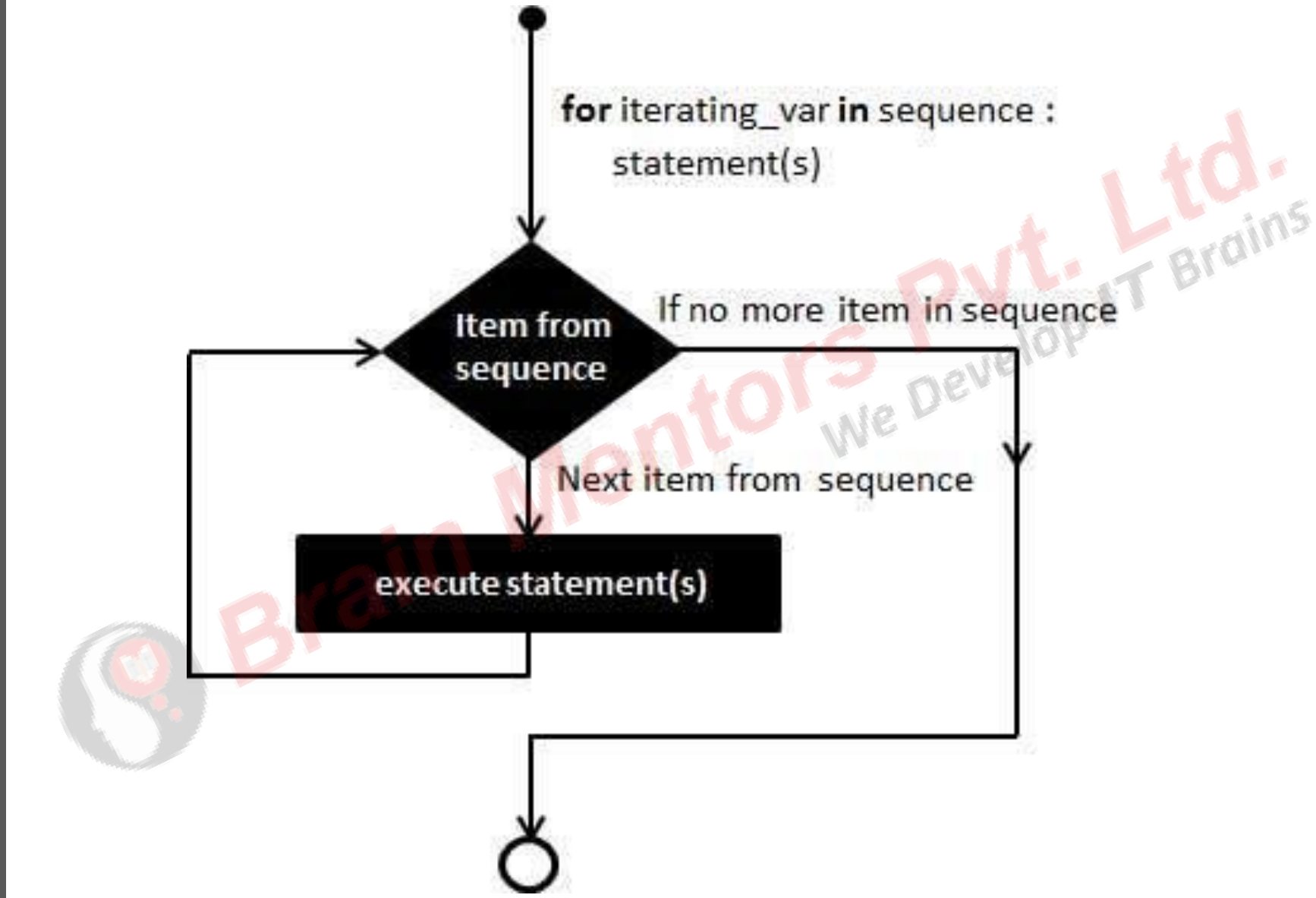
OUTPUT

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

TYPES OF LOOPS

- While Loop
- For Loop





FOR LOOP V/s WHILE LOOP

Lets us take an example, Suppose we have to print the first ten numbers

```
for i in range(1,11):
    print(i)
```

```
num=1
while num<11:
    print(num)
    num+=1
```

KEYWORDS USED INSIDE LOOP

- BREAK
- PASS
- CONTINUE

BREAK

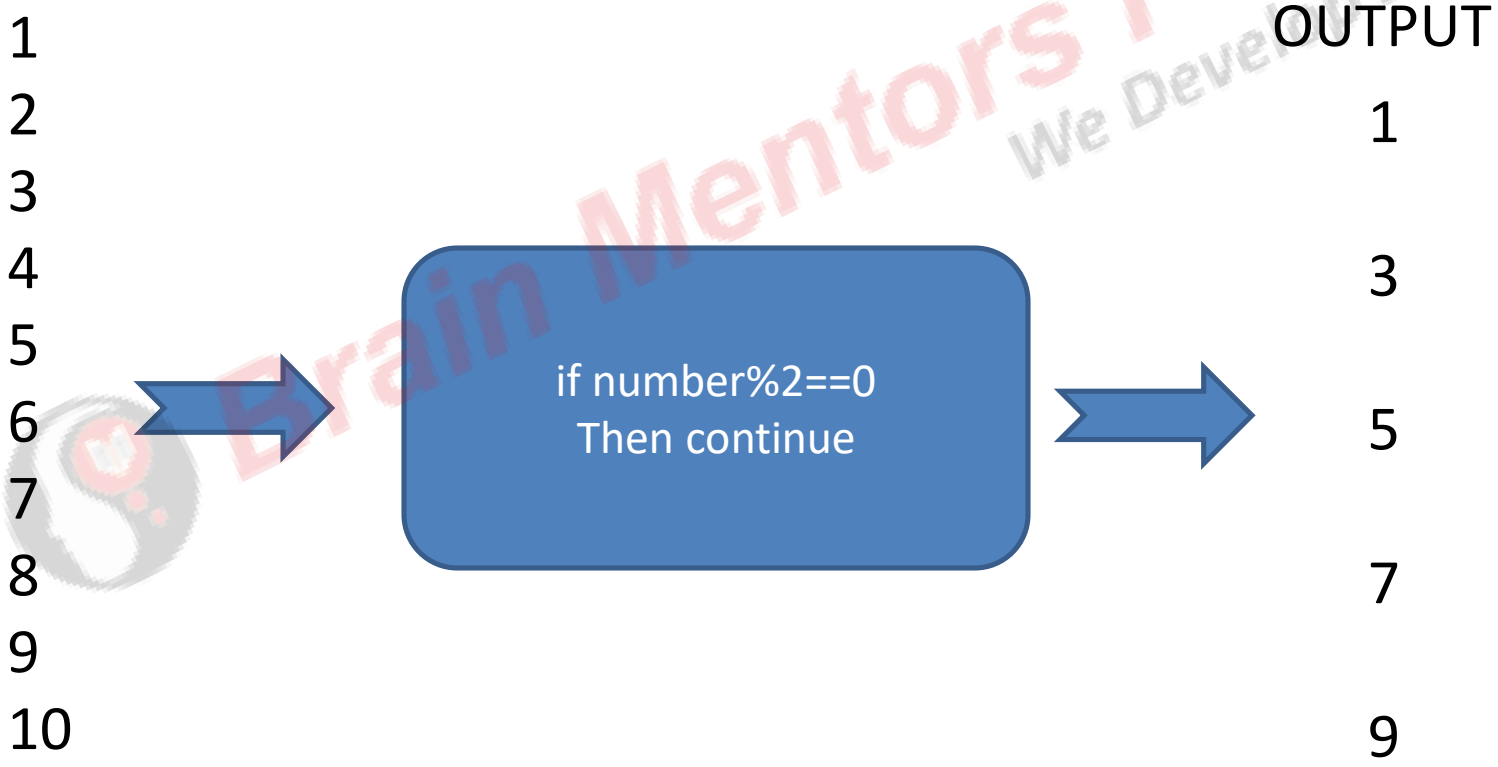
Break Keyword is used to escape from while or for loop.

```
num=1
while num<11:
    print(num)
    num+=1
    break
```

OUTPUT : 1

CONTINUE

Continue Keyword is used to skip current iteration and jump into next iteration in Loop.



SET OF VALUES

PASS

The **pass** statement is a null operation; nothing happens when it executes.

```
for i in range(6):
    print(i)
    if i==5:
        pass
    else:
        print("hello")
```

FILE OPERATIONS IN PYTHON

OPEN FUNCTION

- You can open Files using Python's built-in *open()* function

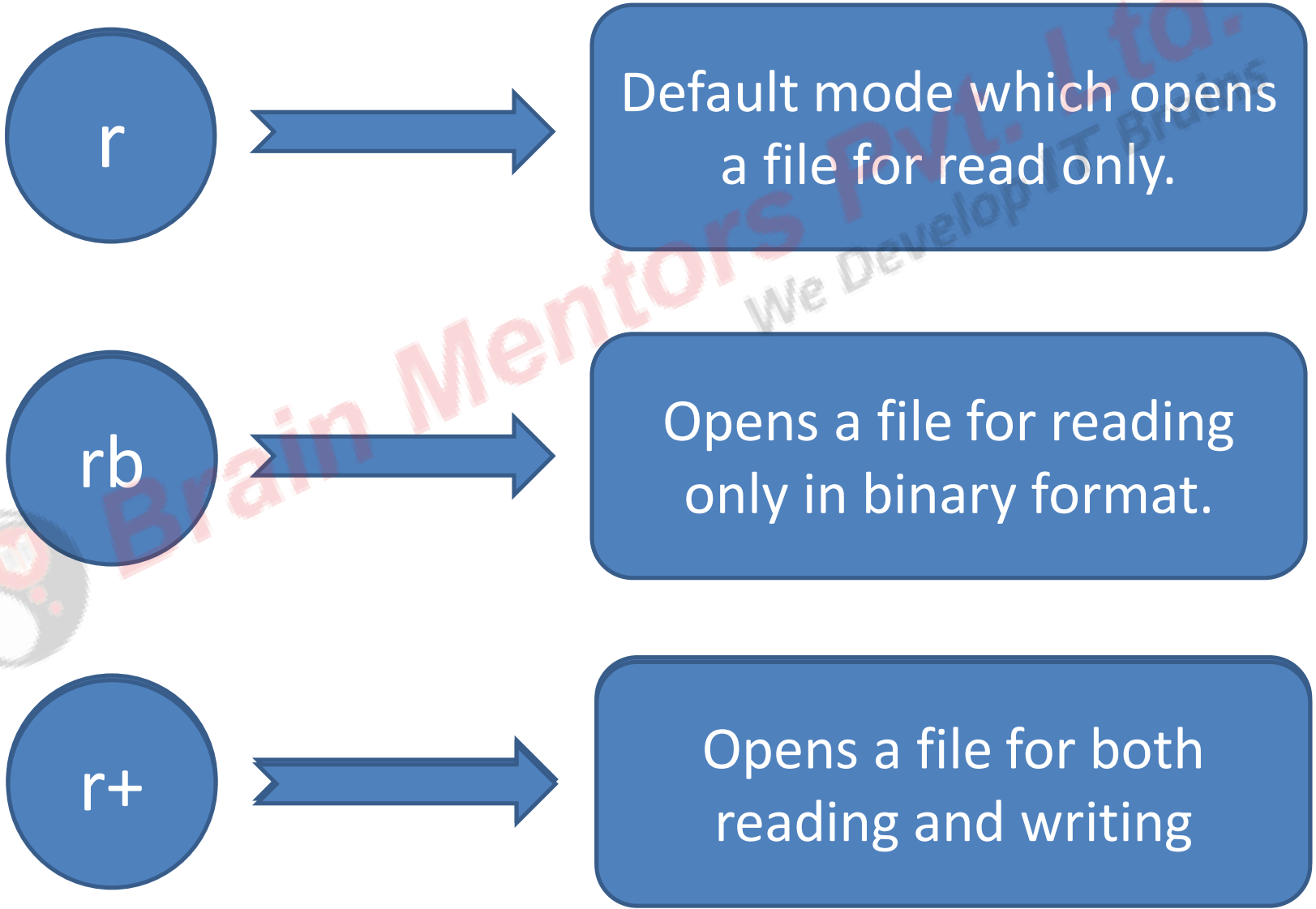
```
file_Object=open(file_name,[access_mode])
```

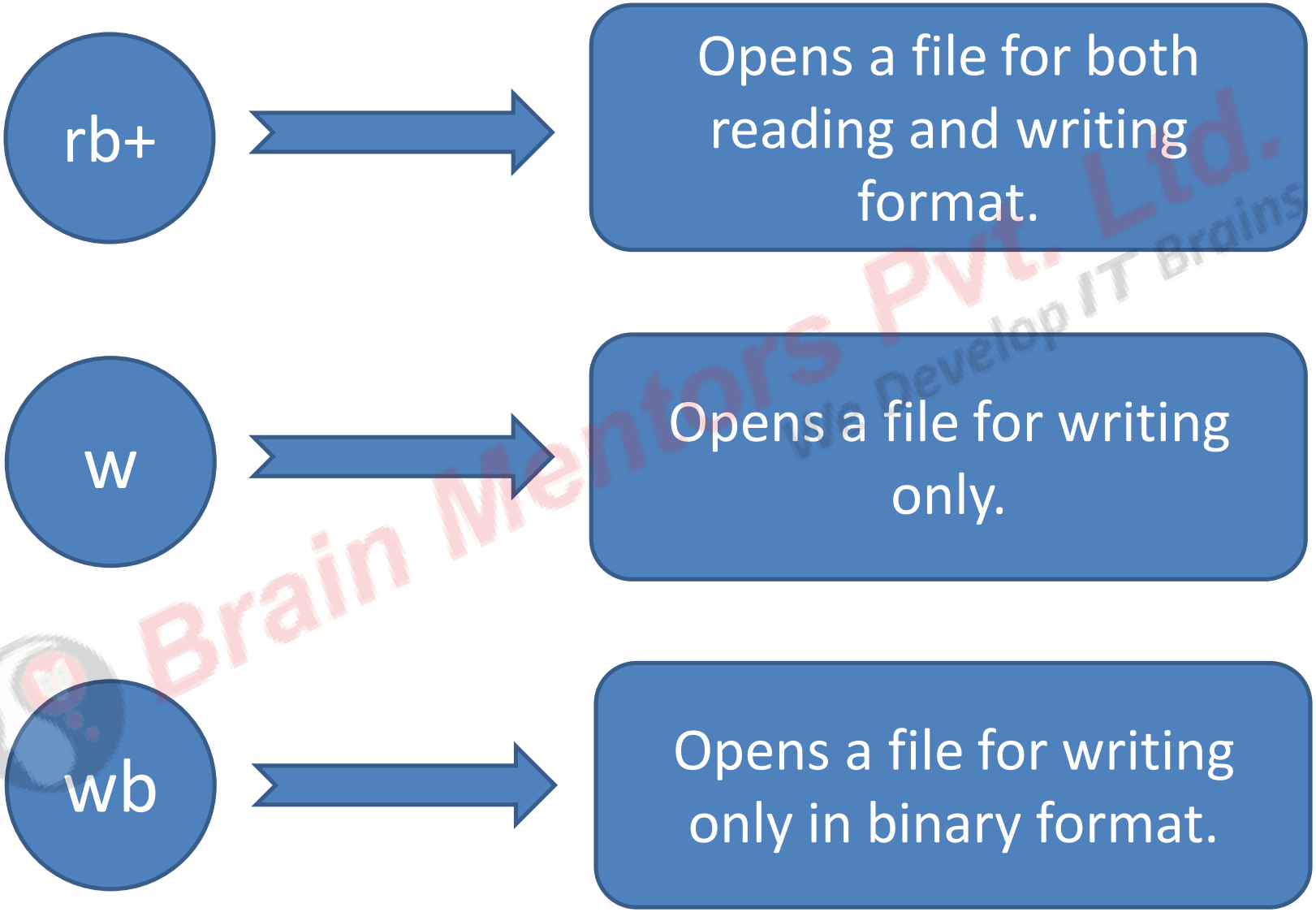
- Here are parameter details:

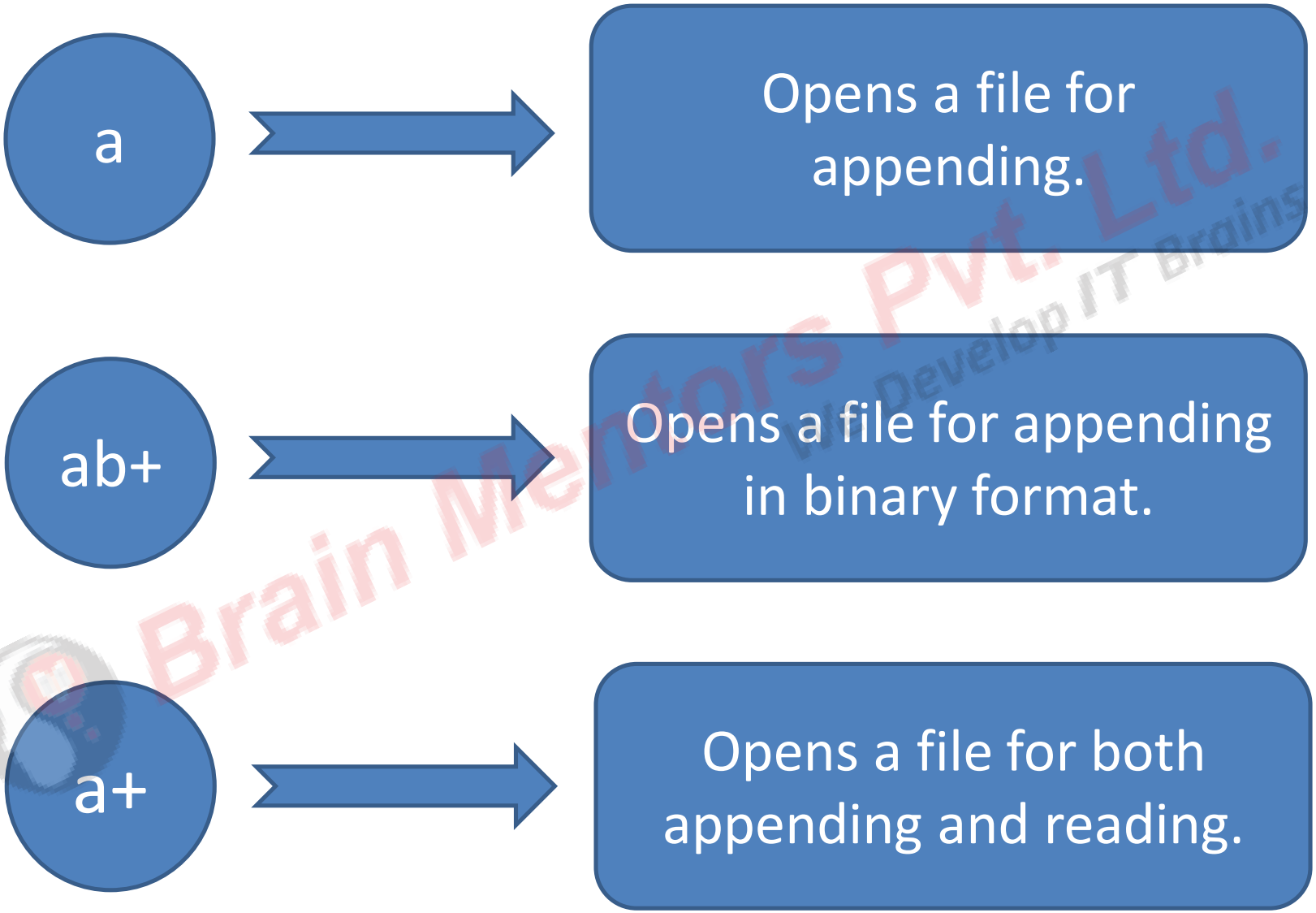
  file_name: The file_name argument is a string value that contains the name of the file that you want to access

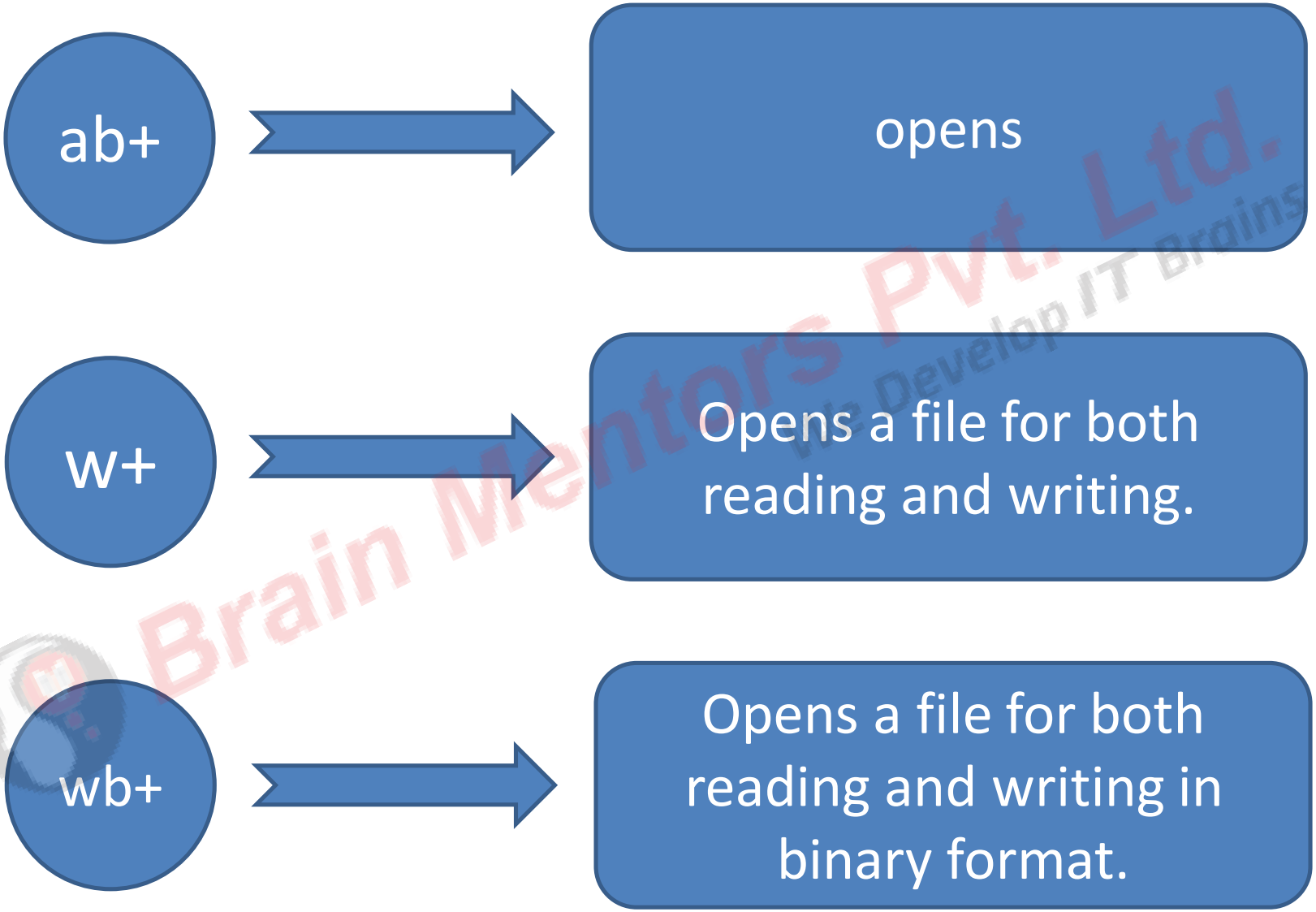
  access_mode: The access_mode determines the mode in which the file has to be opened, i.e., read, write, append

OPEN FUNCTION – ACCESS MODES









Open a file

```
>>> f = open("test.txt")      # open file in current directory
>>> f = open("C:/Python33/README.txt")  # specifying full path
```

Close a file

```
f = open("test.txt",encoding = 'utf-8')
# perform file operations
f.close()
```

Read a file

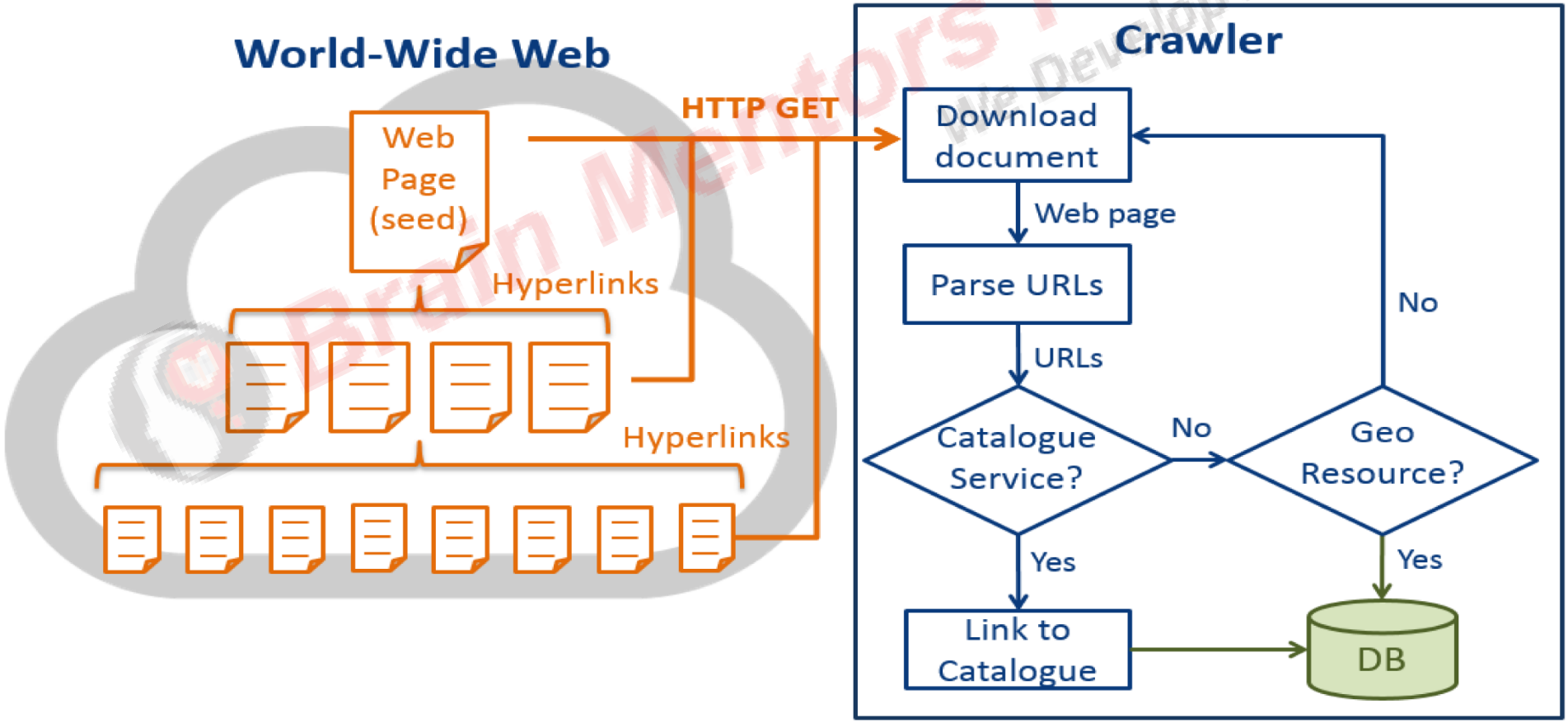
```
file_obj = open('hello.txt', 'r')
data = file_obj.read()
print(data)
file_obj.close()
```

Write a file

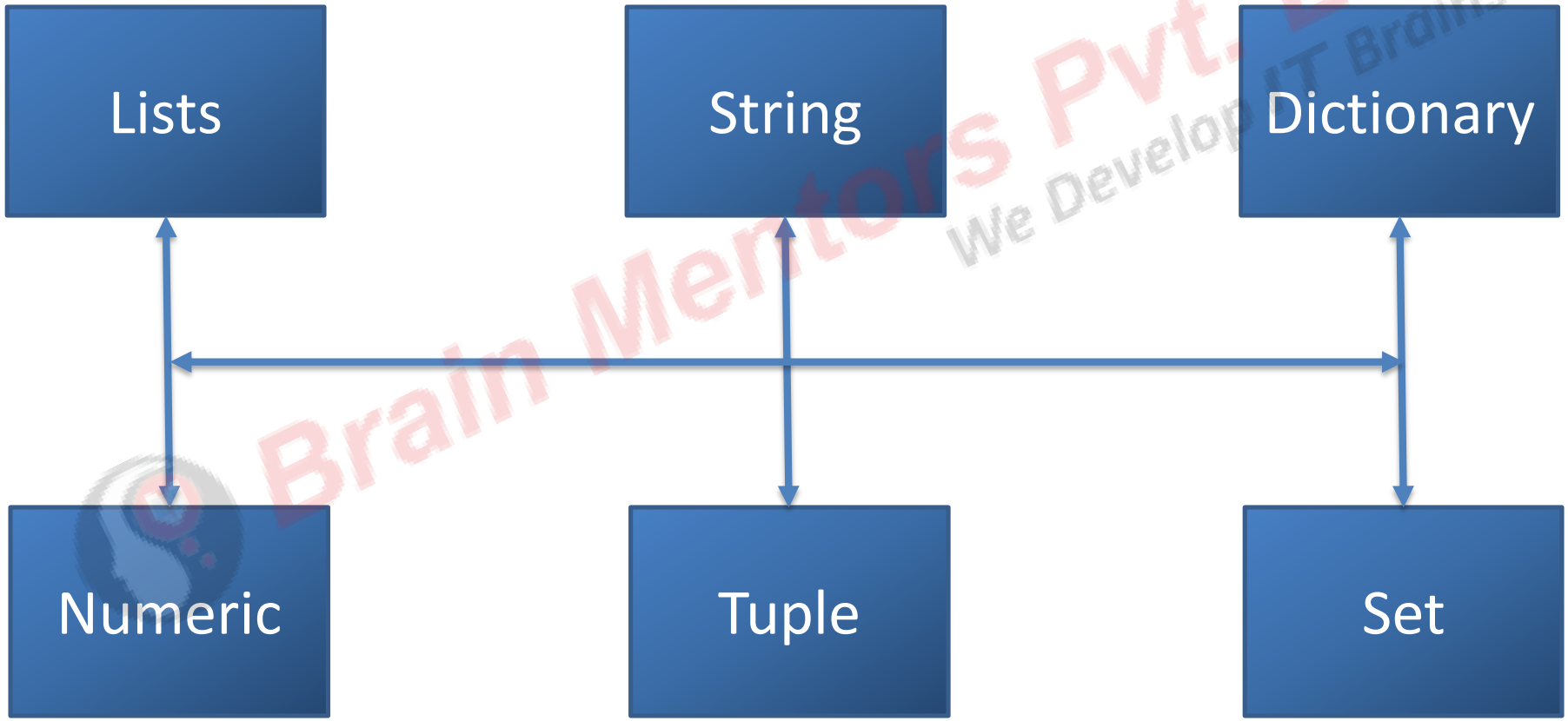
```
file_obj = open('hello.txt', 'r')
file_obj.write("welcome to Brain Mentors")
file_obj.close
```

WEB CRAWLING

Web CRAWLER is a program or automated script which browses the World Wide **Web** in a methodical, automated manner and the process is known as WEB CRAWLING

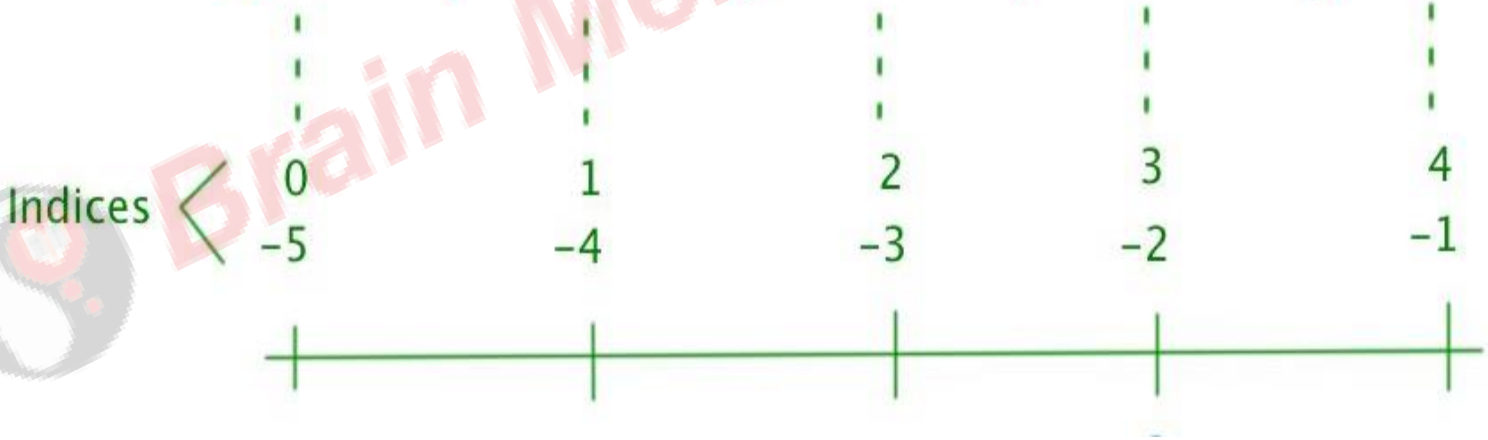


Types of Sequence in Python



LIST

```
t = ['a', 'b', 'c', 'd', 'e']
```



Appending items to LIST

```
>>> list1 = []
>>> list1.append("Brain")
>>> list1.append("Mentors")
>>> list1
['Brain', 'Mentors']
```

Note:- Only one element can be inserted into list at a time using append function.

Therefore, we have to use loop in order to append multiple values….

```
>>> list1=[]
>>> for i in range(5):
        value=input("enter value to be append")
        list1.append(value)

enter value to be append5
enter value to be append4
enter value to be appendbrain_mentors
enter value to be append10
enter value to be append1
>>> list1
['5', '4', 'brain_mentors', '10', '1']
```

Append function append the value at the last of the list.
If we have to insert a value in between list.
Then the code will be as follows:

```
>>> list1
['5', '4', 'brain_mentors', '10']
>>> list1.insert(1,100)
>>> list1
['5', 100, '4', 'brain_mentors', '10']
```

Deleting items from LIST

```
>>> list1
['5', '4', 'brain_mentors', '10', '1']
>>> list1.pop()
'1'
>>> list1
['5', '4', 'brain_mentors', '10']
```

Pop function will delete the value from the last of the list.
If we have to delete a value from between list.
Then the code will be as follows:

```
>>> list1
['5', 100, '4', 'brain_mentors', '10']
>>> list1.pop(0)
'5'
>>> list1
[100, '4', 'brain_mentors', '10']
```

List Slicing

```
>>> list2 = [10,20,30,40,50,60,70]
>>> list2[2:6]
[30, 40, 50, 60]
>>> list2[0:7:2]  # here 2 is no of steps
[10, 30, 50, 70]
>>> list2[:]
[10, 20, 30, 40, 50, 60, 70]
>>> list2[::-1] # -1 to list from backside to get its reversed
[70, 60, 50, 40, 30, 20, 10]
```

List Methods

append

Used to insert items into list .

```
>>> list1=[]
>>> list1.append("Brain_Mentors")
```

Returns the count of no of items in a list.

count

```
>>> list1=[]
>>> list1=[10,12,12,12,10,1]
>>> list1.count(12)
3
```

extend

Add all elements to a list of another list.

```
>>> list1=[1,2]
>>> list2=[3,4]
>>> list1.extend(list2)
>>> list1
[1, 2, 3, 4]
```

index

Returns index of the first matched item.

```
>>> list1 = [1,2,3,4]
>>> list1.index(4)
3
```

insert

Insert an element at defined index.

```
>>> list1=[]
>>> list1=[1,2,3]
>>> list1.insert(0,100)
>>> list1
[100, 1, 2, 3]
```


Removes and returns an element at the given index.

pop

```
>>> list1=[1,2]
>>> list1.pop()
2
```

copy

Returns a shallow copy of list.

```
>>> list1=[[1,2],3,4]
>>> list2=[]
>>> list2=list1.copy()
>>> list2
[[1, 2], 3, 4]
```

remove

Removes an item from list

```
>>> list1=[1,2,3,4]
>>> list1.remove(2)
>>> list1
[1, 3, 4]
```

reverse

Reverse the order of item in a list

```
>>> list1=[1,2,3,4]
>>> list1.reverse()
>>> list1
[4, 3, 2, 1]
```

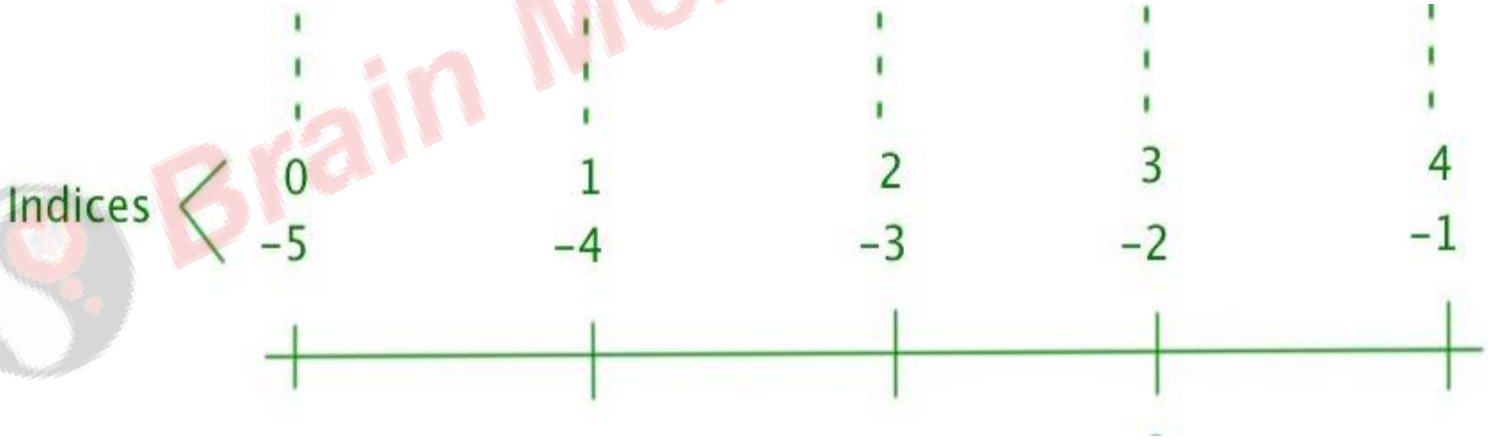
Sort items in a list in ascending order

sort

```
>>> list1=[4,2,1,9]
>>> list1.sort()
>>> list1
[1, 2, 4, 9]
```

TUPLES

T = (‘a’ , ’b’ , ’c’ , ’d’ , ’e’)



Function

Function Definition

Function Call

```
def print_name(str):
    print("Welcome to Python, ",str)
    return()


str = input("Enter your name : ")
print_name(str)
```

A simple function to find sum of two numbers.

```
In [2]:
def add(a,b):
    return a + b
add(100,200)

Out[2]:

300
```

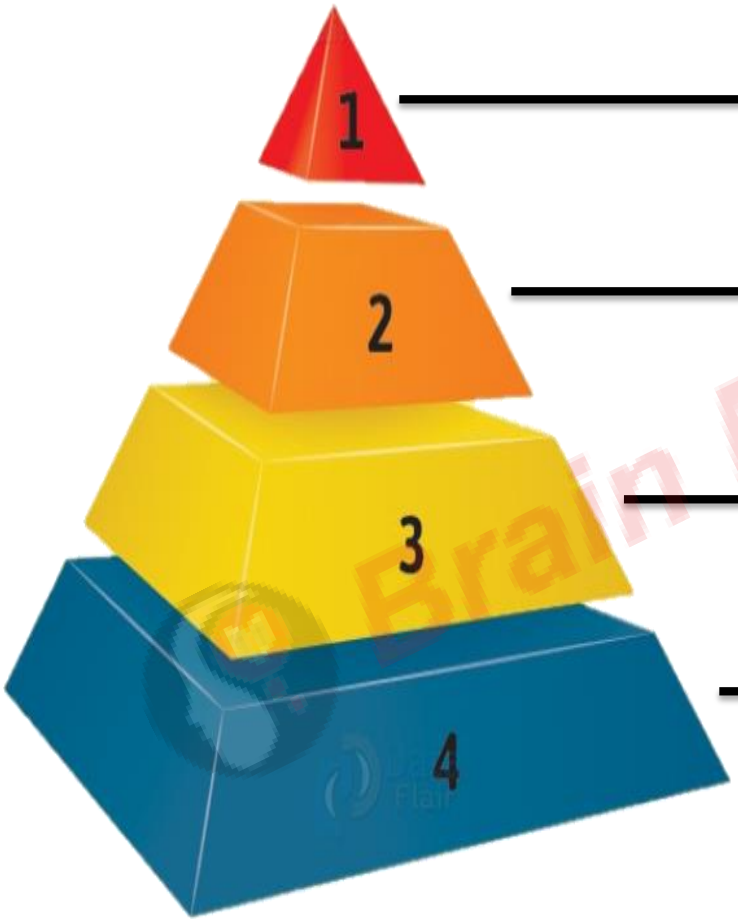
Note : By Default, Function returns none

```
def test():
    print("I am test")
y = test()
print(" y  is {}".format(y))
```

OUTPUT :

```
I am test
 y  is None
```

FUNCTION TYPES



- User Defined Function
- Built-in Function
- Lambda Function
- Recursion Function

Built In Functions

sorted()

Returns a new sorted list from the items of the iterables.

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

print "Sorted List returned :",
print sorted(x)
```

Output :

```
Sorted List returned : [1, 2, 3, 4, 6, 7, 8]
```

all()

Returns True if all variables of supplied iterables are true

```
mylist = [True, True, True]
x = all(mylist)
print(x)
```

Output : `True`

any()

Returns True if any variable of supplied iterables are true

```
myList = [True , False , False, False]
x= any(myList)
print(x)
```

Output : True

bool()

Returns Boolean value either True or False based on logical testing procedure.

```
x = 5
y = 6
print(bool(x==y))
```

Output :

```
False
>>>
```

enumerate()

Returns an enumerate Object with items and there index values.

```
list1 = ["eat","sleep","repeat"]
for i in enumerate(list1):
    print(i)
print()
# changing index and printing separately
for count,i in enumerate(list1,100):
    print(count,i)
```

Output :

```
(0, 'eat')
(1, 'sleep')
(2, 'repeat')

100 eat
101 sleep
102 repeat
```

Lambda Function

- A **lambda function** is a small anonymous **function.**
- A **lambda function** can take any number of arguments, but can only have one expression.

Syntax : lambda *arguments* : *expression*

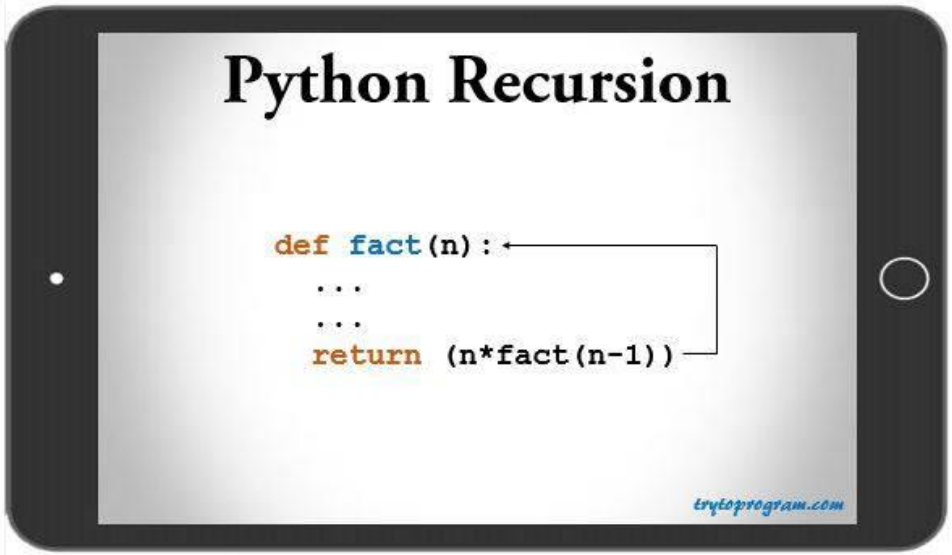
Example : Double of a Number using lambda Function

```
>>> double = lambda x: x * 2
>>> double(5)
10
```

Recursion Function

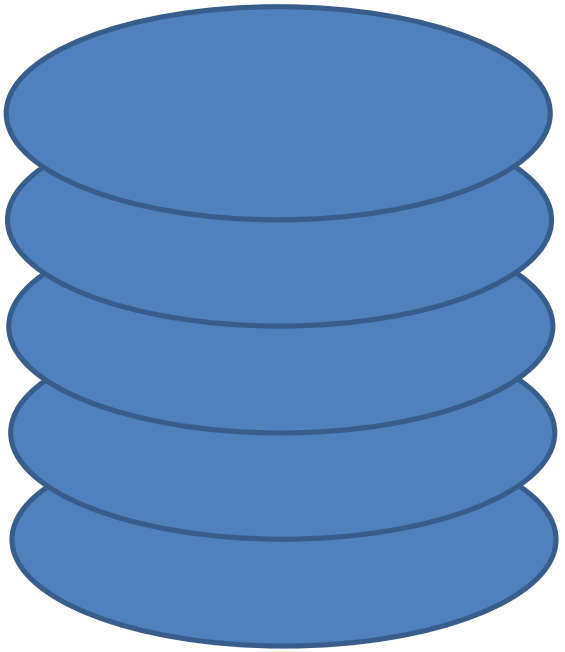
1. **Recursion** is a way in which a function calls itself one or more times in its body. Usually, it is returning the return value of this function call.

2. If a function **definition** fulfils the condition of **recursion**, we call this function a **recursive** function.



HOW RECURSION IS HANDLED

Every time a function is called, the function values, local variables, parameters, return addresses are pushed onto the stack, Over and Over again.



Example

Recursion

```
>>> def countdown (n):
...       if n <= 0:
...             print 'Blastoff!'
...       else:
...             print n
...             countdown (n - 1)
...
>>> countdown (3)
3
2
1
Blastoff!
>>>
```

For Loop

```
>>> def countdown (n):
...     for i in range (n, -1, -1):
...         if i <= 0:
...             print "Blastoff!"
...         else:
...             print i
...
>>> countdown (3)
3
2
1
Blastoff!
>>>
```

While Loop

```
>>> def countdown (n):
...     while n > 0:
...         print n
...         n = n - 1
...     print "Blastoff!"
...
>>> countdown (3)
3
2
1
Blastoff!
>>>
```

Advantage of recursion function

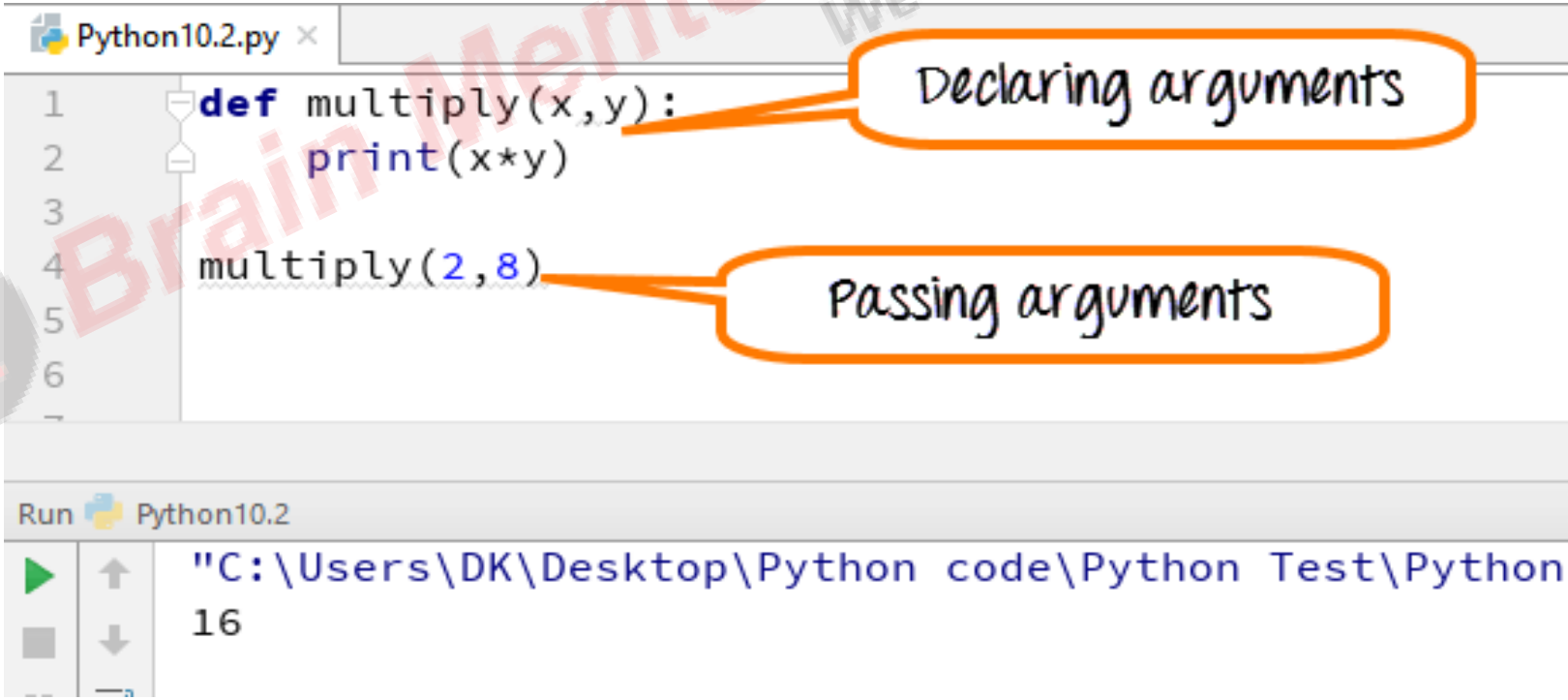
1. Recursive functions make the code look clean and elegant.
2. A complex task can be broken down into simpler sub-problems using recursion.
3. Sequence generation is easier with recursion than using some nested iteration.

Disadvantage of recursion function

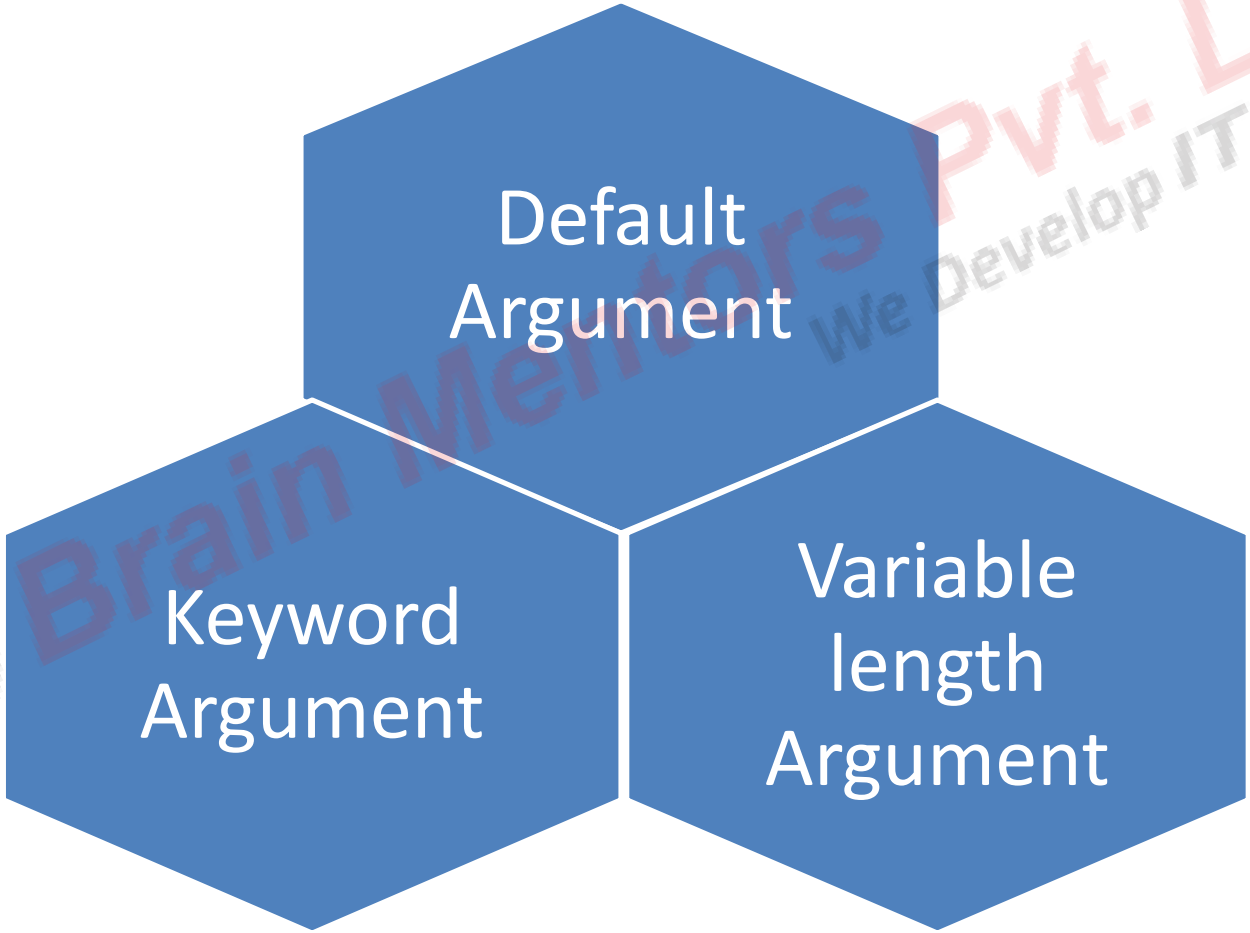
1. Sometimes the logic behind recursion is hard to follow through.
2. Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
3. Recursive functions are hard to debug.

Function Arguments

Arguments are the values that are passed to the function at run-time so that the function can do the designated task using these values.



TYPES OF ARGUMENTS



DEFAULT ARGUMENT

Default values indicate that the function argument will take that value if no argument value is passed during function call. The default value is assigned by using assignment(=)

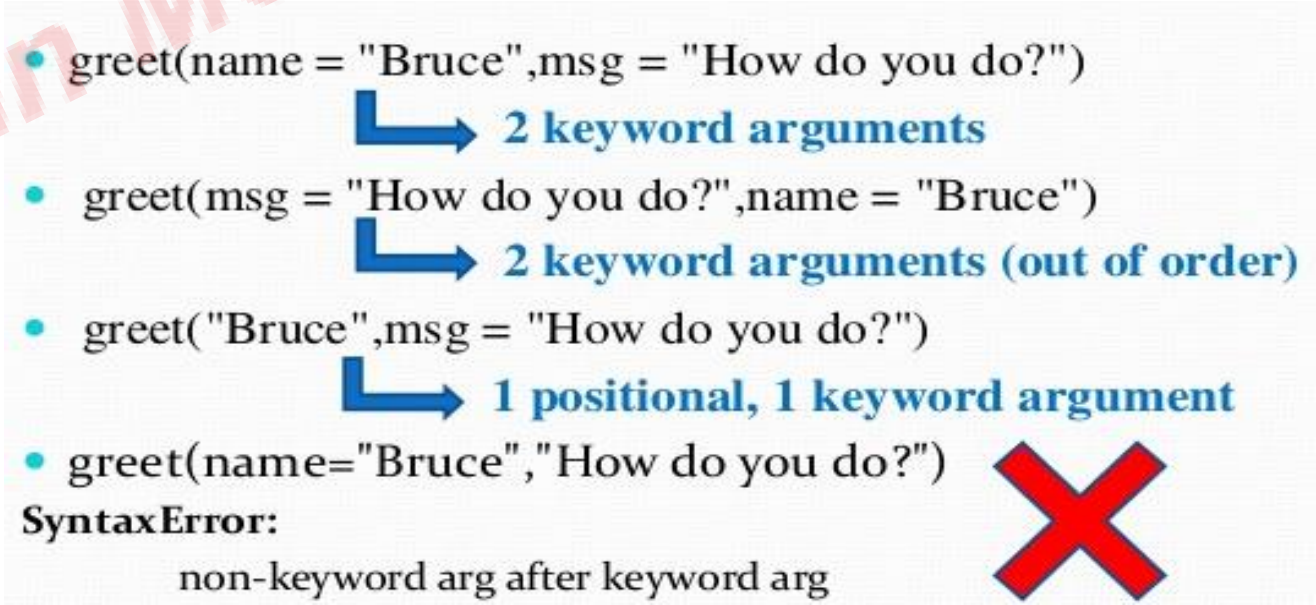

```
>>> def student(firstname, lastname ='Mark'):
        print(firstname, lastname,)

>>> student('Ramu','Prasad')
Ramu Prasad
```

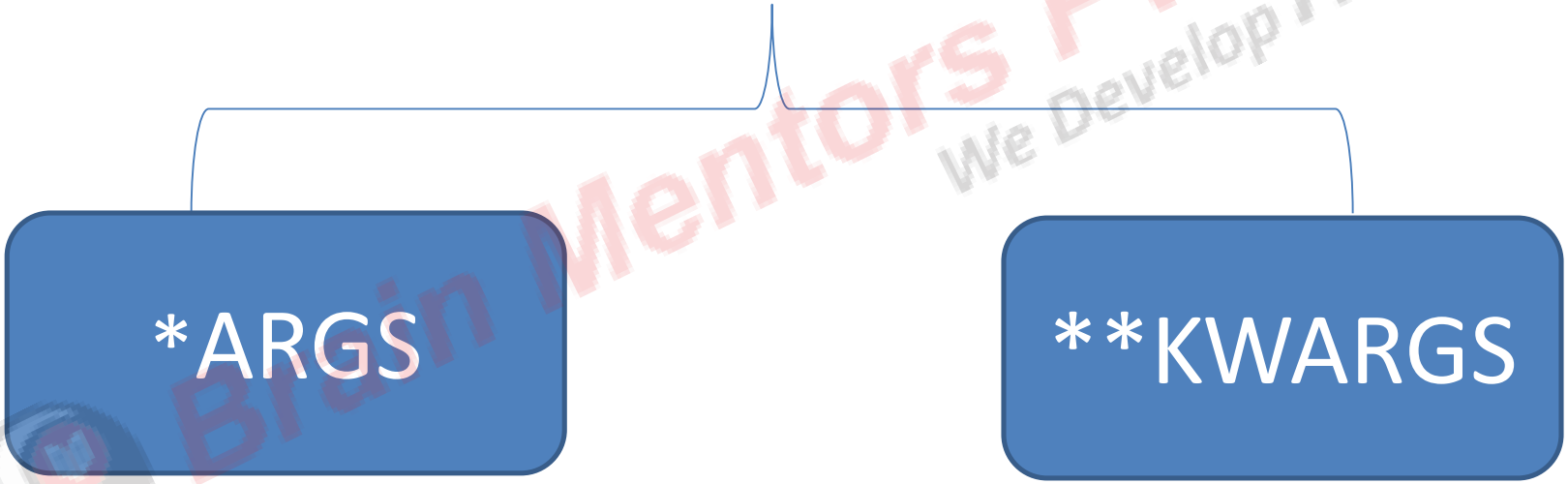
KEYWORD ARGUMENT

Python allows functions to be called using keyword arguments.

```
Function :
    def greet(msg = "Good morning!", name):
```



ARBITRARY ARGUMENTS



Non – Keyword Argument

Keyword Argument

*args - Non keyword Arguments

```
>>> def details(*name):
        name1   = name
        print(name1)

>>> details(10,20,30,40,'Brain-Mentors')
(10, 20, 30, 40, 'Brain-Mentors')
```

**kwargs - Non keyword Arguments

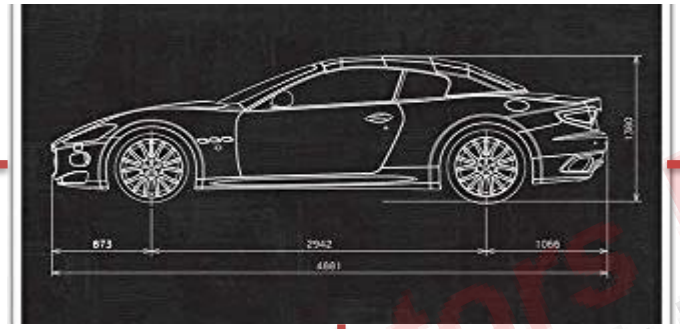
In [1]:
```
def foo(**kwargs):
    print(kwargs)

kwargs = {'$$$': 4, '+-=': 5}
foo(**kwargs)
```

```
{'$$$': 4, '+-=': 5}
```

OBJECT ORIENTED PROGRAMMING

CLASS AND OBJECT

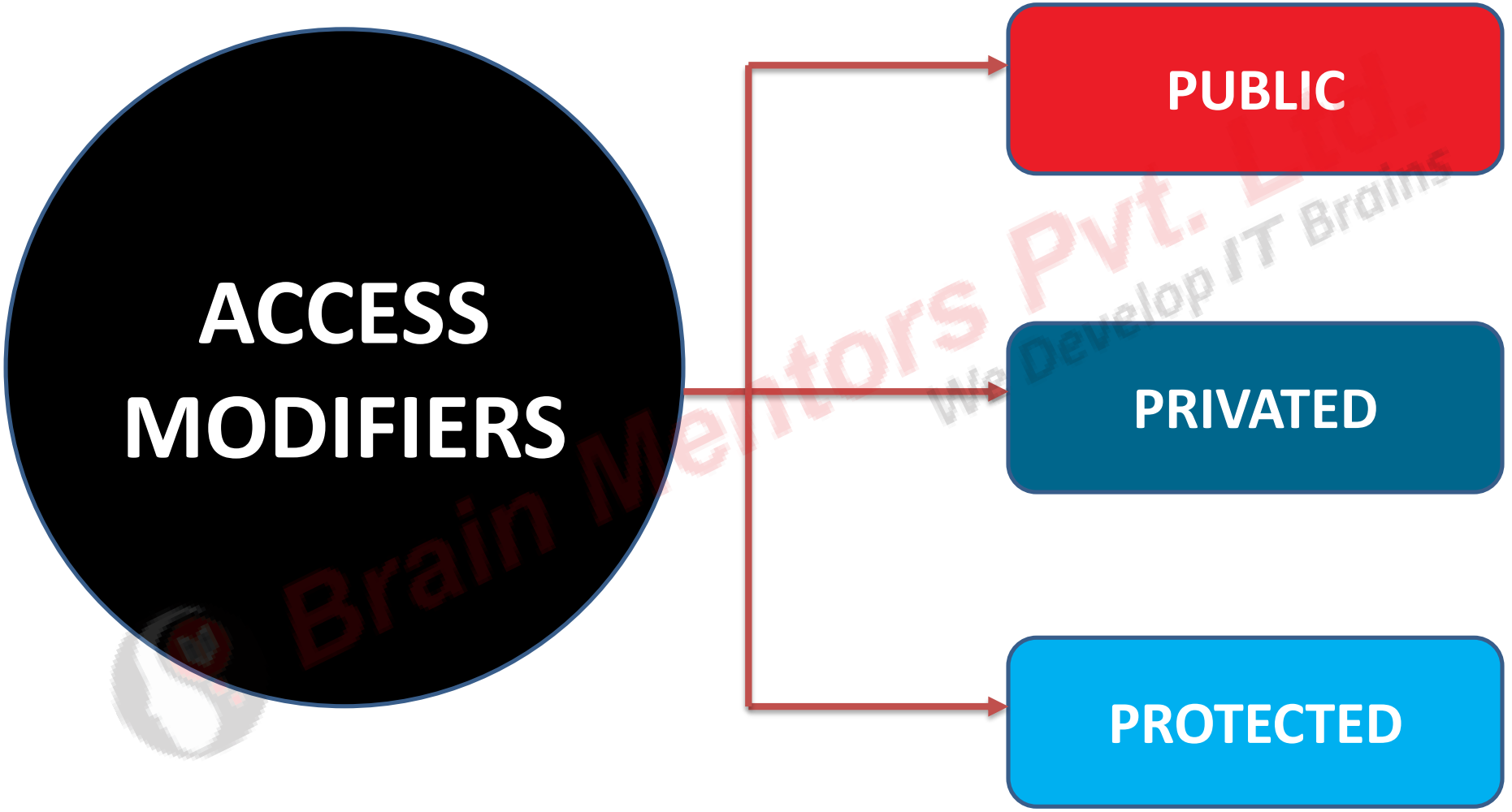


EXAMPLE

```
# class is started by keyword Class
class Person:
#All classes have a function called __init__(),
#which is always executed when the class is being initiated
  def __init__(self, name, age):
      #self is a reference variable
    self.name = name
    self.age = age

p1 = Person("John", 36) #Object Declaration
print("My name is {} and age is {}".format(p1.name,p1.age))
```

OUTPUT : My name is John and age is 36



PUBLIC ACCESS MODIFIER

All members in a Python class are **public** by default. Any member can be accessed from outside the class environment.

```
>>> class employee:
    def __init__(self, name, sal):
        self.name=name
        self.salary=sal

>>> #here name, salary are public data members of class
>>> #which can be accessed outside the class  by using object.
>>> object1 =  employee("RAMU", 100000)
>>> object1.name
'RAMU'
>>> object1.salary
100000
```

PRIVATE

Private members of a class are denied access from the environment outside the class. They can be handled only from within the class.

```
>>> class employee:
    def __init__(self, name, sal):
#double underscore is used to declare data members as private.
        self.__name=name  # private attribute
        self.__salary=sal # private attribute

>>> e1 = employee("KISHAN",10000)
>>> e1.name
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    e1.name
AttributeError: 'employee' object has no attribute 'name'
```

PROTECTED

Protected members of a class are accessible from within the class and are also available to its sub-classes. No other environment is permitted access to it.

```
>>> class Employee:
    def __init__(self, name, sal):
        self._name = name   # protected attribute
        self._sal = sal     # protected attribute

>>> emp = Employee("RAMU",10000)
>>> # we want to access the protected members then use underscore
>>> emp._name
'RAMU'
>>> #otherwise you cant access it
>>> emp.name
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    emp.name
AttributeError: 'Employee' object has no attribute 'name'
```

Note : Also can use anything instead of self, You can this also and self is just a variable name holding a reference of current object it is not a keyword. Using this in below example instead of self class Customer:

```
def __init__(this, id, name):
 this.id = id
 this.name = name
 def show(this):
 print("Id is ",this.id," Name is ",this.name)

ram = Customer(1002, 'Ram')
ram.show()
```

Class & Instance Attributes in Python

Class Attributes :

Class attributes belong to the class itself they will be shared by all the instances. Such attributes are defined in the class body parts usually at the top, for legibility.

```
class A:
    a = "I am a class attribute!"
x = A()
print(x.a)
```

OUTPUT : I am a class attribute!

Instance Attributes :

Like class attributes, instance attributes are not shared by objects. Every object has its own copy of the instance attribute

To list the attributes of an instance/object, we have two functions:-

1. **vars() :** This function displays the attribute of an instance in the form of an dictionary.

2. **dir() :** This function displays more attributes than vars function,as it is not limited to instance. It displays the class attributes as well.

```
class emp:
    def __init__(self):
        self.name = 'xyz'
        self.salary = 4000

    def show(self):
        print(self.name)
        print(self.salary)
  e1 = emp()
print("Dictionary form :", vars(e1))
print(dir(e1))
```

OUTPUT

```
Dictionary form : {'name': 'xyz', 'salary': 4000}
['__class__', '__delattr__', '__dict__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__le__', '__lt__', '__module__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', '__weakref__', 'name', 'salary',
'show']
```

Static method

- Simple functions with **no self argument**.
- Nested inside class.
- Work on **class attributes**; not on instance attributes.
- Can be called through both class and instance.
- The built-in function `staticmethod()` is used to create them.

```
class Hello():
    def __init__(self):
        print('Hello, World!')

    @staticmethod
    def static_ex():
        """ This method is a static method! """
        print('I\'m a static method!')

x = Hello.static_ex()
```


Output : I'm a static method!

Class method

- Functions that have first argument as **class** name.
- Can be called through both class and instance.
- These are created with `classmethod` inbuilt function.
- These always receive the lowest class in an instance's tree. (See next example)

```
class Human:
    age = 20
#cls stands for class
    def Age(cls):
        print('The age is:', cls.age)

#class method
Human.Age = classmethod(Human.Age)
Human.Age()
```

Output : The age is: 20

@classmethod is also act as a factory method because it return the object

```
@classmethod
def mday(cls,month,day,year):
    cls.month = month
    cls.day = day
    cls.year = year
    #order of return should be same as init
    return cls(cls.month,cls.day,cls.year)
```

Instance method

- By default, methods defined in classes are bound to the class's instances. They are called from instances and receive these instances in their first parameters.
- When you assign an instance method to a variable, its parent instance is "bound" to it. Example:

```
v = Vehicle()
x = v.move
y = Vehicle.move
x()          → Calls Vehicle.move(v)
y()          → Error! Not enough parameters!
```

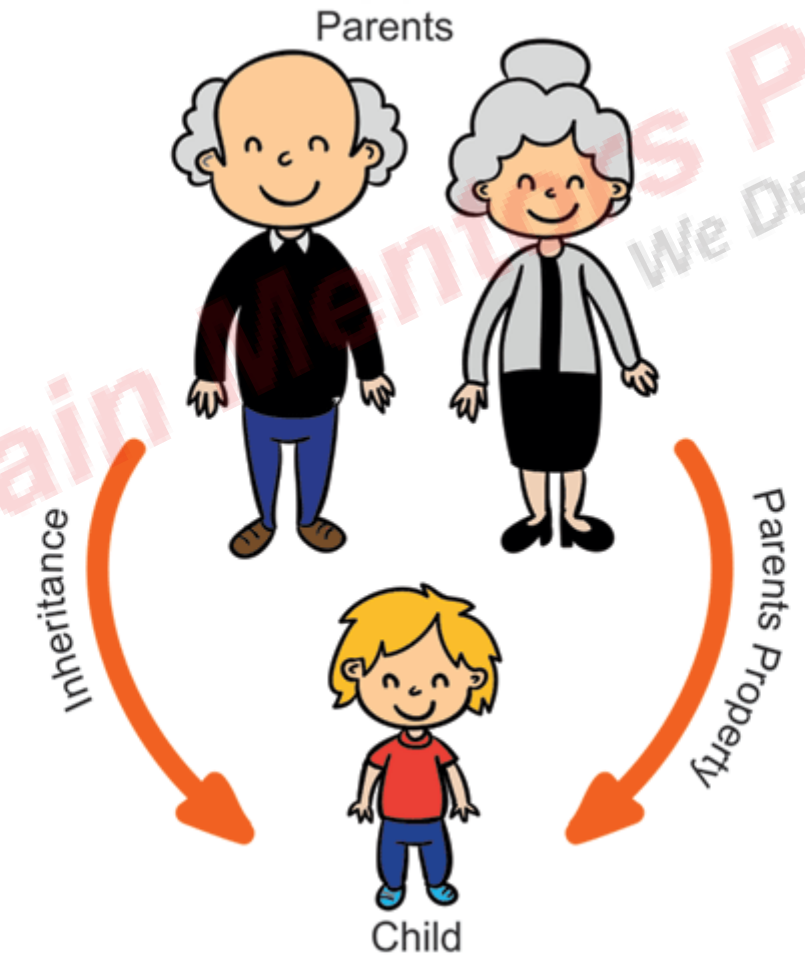
Dynamic Creation of Instance / Class var

```
D.yyy = 1000
r   = D()
r.nnn = 2000
print(D.yyy)
print(r.nnn)
```

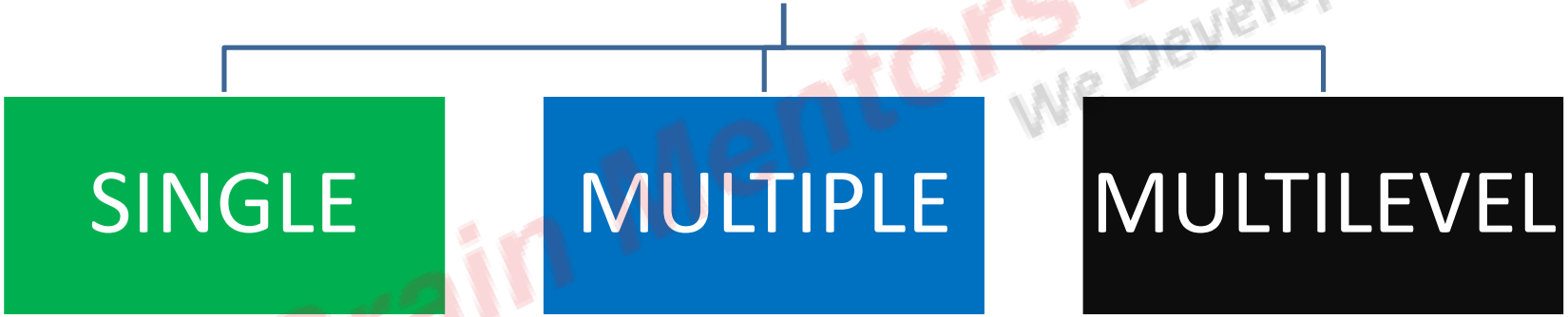
Scanning of an Object

```
obj.new_object_var = "hello"
for elem in dir(obj):
    print(elem)
```

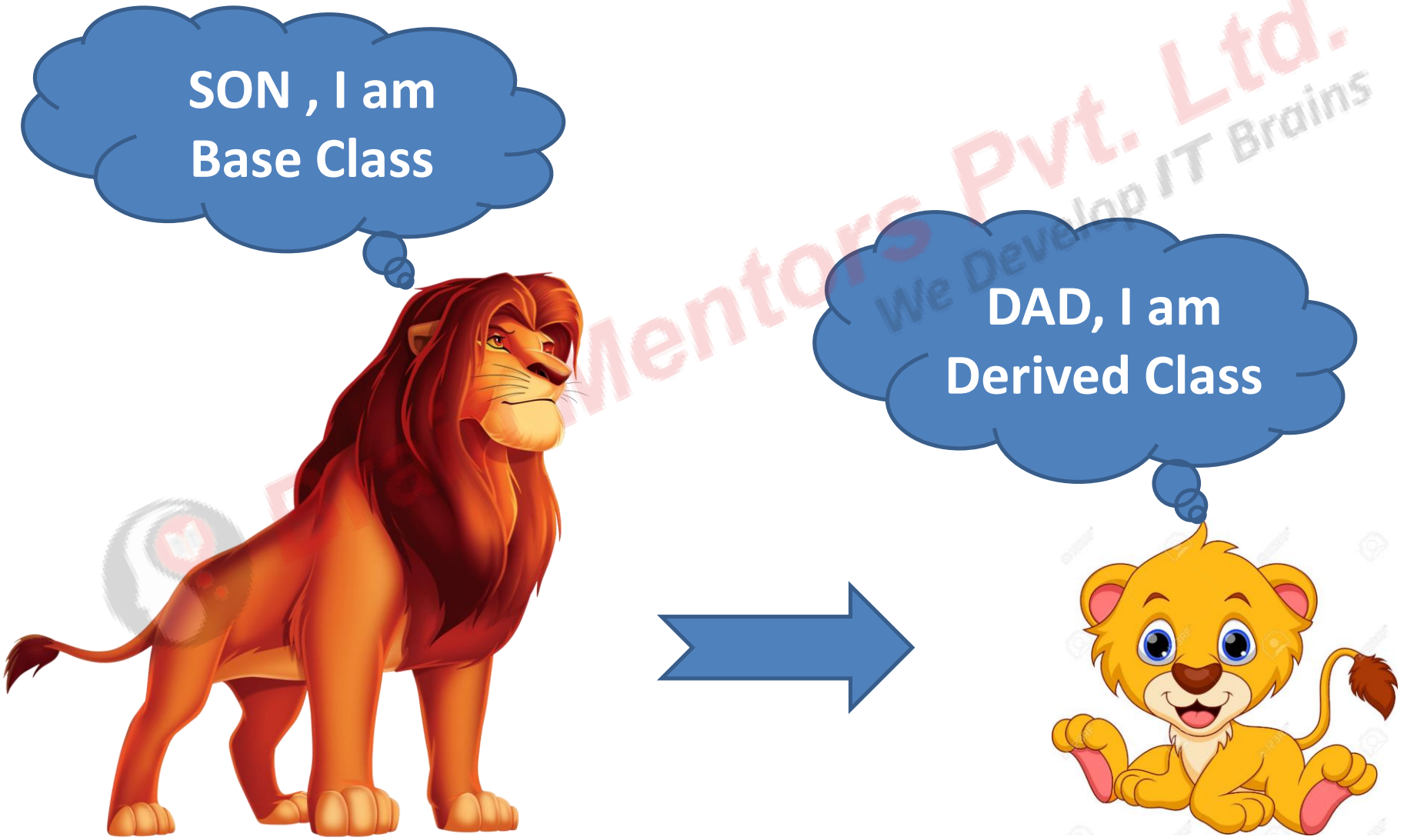
INHERITENCE



TYPES OF INHERITENCE



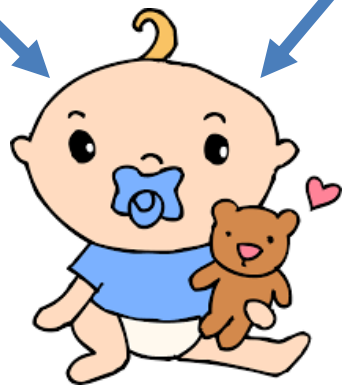
SINGLE



MULTIPLE

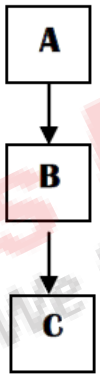
Base class1

Base class2



Derived Class

MULTILEVEL



Base Class, Grandfather

Intermediate base Class, Father

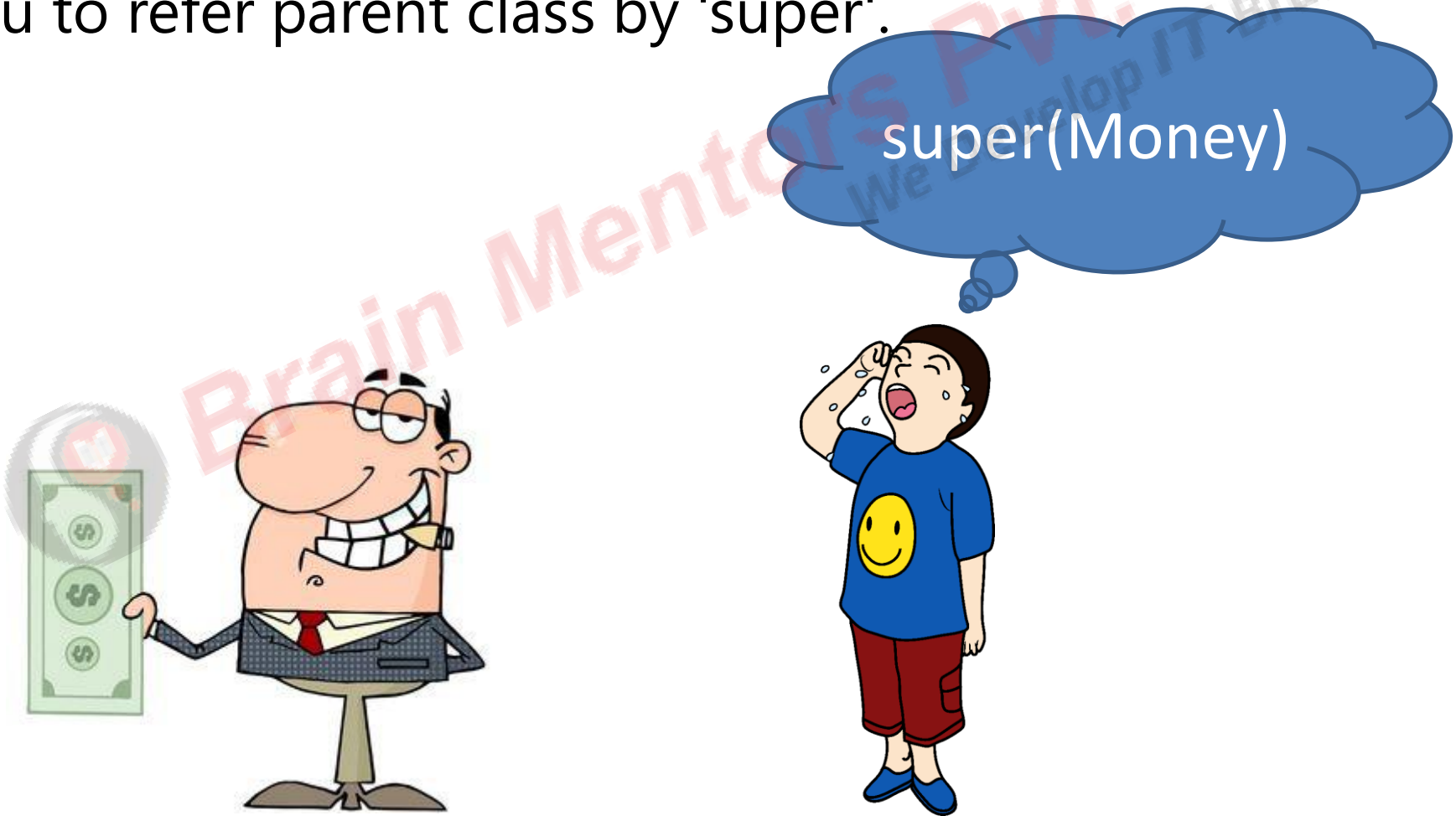
Derived Class, Child

Python Inheritance Syntax

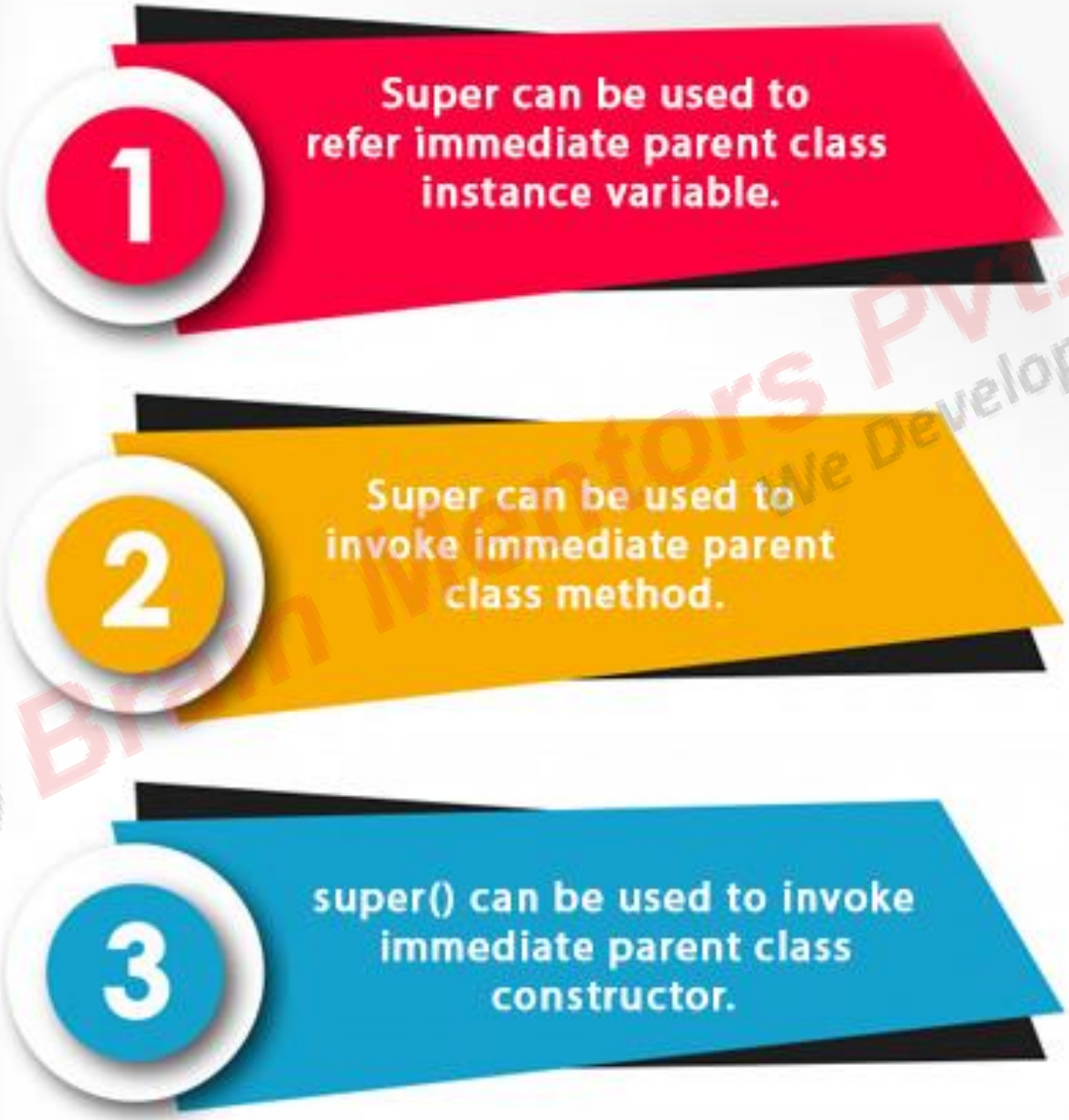
```
class BaseClass:
   Body of base class
class DerivedClass(BaseClass):
   Body of derived class
```

super() METHOD

The super() builtin returns a proxy object that allows you to refer parent class by 'super'.



Usage of Super Keyword



```
class Parent:
    def __init__(self,x,y):
        print("Parent 2 Args Cons Call")
class Child(Parent):
    def __init__(self,z):
        super(Child, self).__init__(1000,2000)
        print("Child Call One Args")
d = Child(9)
```

```
Parent 2 Args Cons Call
Child Call One Args
```

POLYMORPHISM

It means that an object can have many forms

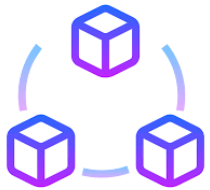
STANDARD LIBRARIES

Standard Library is a collection of tools that come with python.

The Standard Library include the following items :

Standard Libraries

Built-in functions



Modules

Packages

What is module..?

A Module allows you to logically organize the code.

Python Modules are nothing but python files with .py extension.

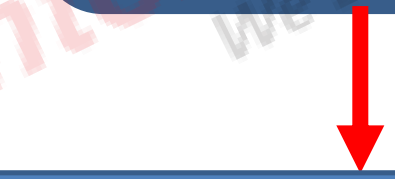
A module is a file containing Python definitions and statements

Importing Module

Use ‘import’ to load a module in Python. When the interpreter encounters an import statement, it imports the module, if the modules is available in search path.

See a function inside module

```
import math
print (dir(math))
```



['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']

WAYS TO IMPORT MODULE

#full import, import all functions inside a module

```
import math
dir(math)
```

#specific import

```
from math import pow,pi # this will import only pow and pi
pow(2,3)
pi
```

#full import, import all functions inside a module , But in this case no need to write os.methodname now u can write method name directly

```
from os import *
```