**Python**

* General purpose programming language. “Guidov van Rossum” “1993”
* Python is a loosely typed language with no need to write variable type. it automatically changes into that type.
* Memory in python: **stack(store variable name**) and **heap**(**store actual values)…**and they are joint by references.
* Python Interpreted/scripted language. (line by line execute)
* Case sensitive. **ex**: name & Name are different.
* In single variable we can store different type of data. (dynamic type)

Ex:

val = 1212

val = 12.23

val = 4e6 -> 4000000

val = “Pawan” or ‘Pawan’

val = “hi

my

name

is

pawan”

val = True

* print(“ .. ”, “..“, “… ”) (print anything)

every print function inbuild sep=’ ’(space) after comma || end=’\n’ at the end

* sep=, end= (properties on print() )
* input() (take input from user as a string)
* len() (length of string or integer)
* abs(-number) (-ve value)
* id(variable\_name) (address of variable)
* pow(2, 5) (32 i.e. 2\*2\*2\*2\*2)
* round(3.33) (3 i.e. round of)
* formate(variable1, variabl22) (formating)
* min(list\_variable) (minimum value from list)
* max(list\_variable) (maximum value from list)
* sum(list\_variable) (sum of list)
* sorted(list\_variable) (new sorted list created)
* list\_variable . sort() (same list sort)
* as (for short name convention)
* + (concatenate strings)
* , (separate the string and other type data or same datatype)
* # (single line comment)
* “””…..””” (multi line String/comment)
* int(..) (change the type of variable which is int)
* float(..) (change the type of variable which is float)
* str() (use for change into the string)
* python **support infinite number** as well.. we have to save in **float**..

a = float("inf")            # +ve infinite

b = float("-inf")           # -ve infinite

* “self” just like “this” keyword
* help() (any kind of help………..check all keyword)
* **Ex**:

val = (12) assume as int

val = (12,) use ‘ , ’..now it is tuple

* **pass keyword:** if we want to give definition later that time use.
* **function\_a( function\_b )**  (function as a parameter..function\_a is higher order)
* type(..) (can check the type of variable )

num = 12.232

print(type(num))

* round(..) (round off the divided number)

div3 = round(num1/num2)                 # 8/3 = 2.666666665 so return O/P: 3

div4 = round(num1/num2, 3)              # return UPTO 3 decimal place

* fstring (**formatting**: way by which we print other datatype to string type)

num1 = 12

num2 = 12.334

num3 = True

print(f"first: {num1}, second: {num2}, third: {num3}")

print("1st: {}, 2nd: {}, 3rd: {}".format(num1,num2,num3))

* [] print character of string

print("Welcome"[2]) # l

print("Welcome"[-2])   # m

* Separator character in between string { sep(“*some\_thing*”) }

print("my", "name", "is", sep="....") # my….name….is

* separator character at end of string { end(“*some\_thing*”) }

print("my", "name", "is", end=".|.|.") # my name is .|.|.

* lower() , upper() [change is the lower case and upper case]

name = "Pawan Bisht"

l = name.upper()

u = name.lower()

* count(“name\_of\_string“) [count the string and character]

a = lower\_case\_str.count("t") # no of time coming “ t ”

b = lower\_case\_str.count("r")

* dir() [find all function for an particular object]

print(dir(str))

* range(start, end, jump) [it is use in for loop]
* **Slicing**: syntax- [start:stop:step]

li = ['a','b','c','d','e','f','g','h','i']

print(li[1:5])

print(li[:-1])                  # apart form last  [a to h]

print(li[:])                   # complete list

print(li[0:-1])                 # apart from last value [a to h]

print(li[-4:-1])                # [f,g,h]

print(li[0:8:2])                # [a,ce,g]

print(li[8:0:-2])               # \*\*2 step reverse order [i,g,e,c]

print(li[::-1])                 # \*\*reverse     [i,h,g,f,e,d,c,b,a]

* **Encoding: | Decode: ord() & chr()**

print(ord('😀'))

print(ord('🐕'))

print(ord('a')) # op: 97

print(ord('स')) # op: 2360

……………..

print(chr(2360)) # 'स'

print(chr(65))

print(chr(128512)) #'😀'

* **Type Casting:** can cast *int to string* or *string to int*

**Ex:** float(‘1.22’) **o/p:** 1.22 (float type)

**Ex:** int(‘one’)  **o/p**  error (error……not possible)

**Ex:** str(12) **o/p** ‘12’ (now string type)

**Ex:** int(TRUE) **o/p** 1 (now int type)

**Ex:** bool(1) **o/p**  True (now Boolean type)

Str -> float -> int (E.g. “5.3” -> 5.3 -> 5) # step by step convert

* inside function can define function
* **global** keyword: local variable can access and change the global variable as well.

def fun():

    global x

    x = 300

    print(x)

print(x)     # error

fun()

print(x)            # 300 (print local variable)

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

x = 2

def fun():

    global x

    x += 300 # here x is global

    print(x) # 302 (change)

print(x)       # 2

fun()

print(x)            # 302 (change)

* **Keywords:** reserved word

Class, for, from, while, continue, break, if, elif, not, or, and, true, false, print, return, try, except, def, lambda, yield, import, global, assert, exec, raise, in, async, as etc

* **Primitive Datatype:**

1. Numeric(int, complex, float)
2. Boolean
3. Set
4. dictionary,
5. sequence(list, tuple, string)
6. None Ex: a = None

str = 1212 (int)

str = 123\_123\_121 (BigInt)

str = 121.44 (float)

str = 2e6 (2000000) (float)

str = 2+4j (complex)

str = True (Boolean)

str = {‘2’, ‘a’, ‘e’} (set)

str = {1: “Pawan”, “hi”: “bisht“} (dictionary)

str = “WelCome” (String)

str = [1,2,3,4,5] (list)

str = (0,1,2,3,4) (tuble)

Large number use ‘\_’ act like comma

Ex:

# num = 123\_4567\_789

* **Type Casting:** type conversion

int()

float()

str()

* **Operator:**

1. Arithmetic operator (+, %...)
2. assignment operator (=, +=, ^=, >>>=, >>=, …)
3. comparison (conditional) operator (>, ==, ===, !==, …)
4. logical operator (and, or, not, ^(XOR))
5. bit wise op (&, |, >>, << [left shift( no\*2 )])
6. identity operator (is, is not) {return true or false}
7. membership operators (in, not in) {return true or false}

() (paranthisis)

+ (add)

print(100+300)                                     #..add

print(str(100)+str(300))                        #..cancatinate

/ (divide float value),

// (divide int value) (left side value for -ve)

round()

div1 = num1/num2                        # return float value

div2 = num1//num2                       # return int value

div3 = round(num1/num2)                 # 8/3 = 2.666666665 so O/P: 3

div4 = round(num1/num2, 3) # 2 decimal places

\* (multiplication)

\*\* (power)

mul = num1\*num2

pow = num1\*\*num2

=== [equal value or same type]

!== [not equal value or not same type]

is [(a is b) return true if both are equal]

is not [(a is not b) return true if both are not equal]

**bitwise** (&, |, ^, ~, <<, >>)

# in operator…..

name = "Coding the python programming"

print("the" in name)        # True

print("The" in name)        # Flase

**MODULE: Library, package**, where all file and function is there.

* import [use for import other file/module data]

import newFile  
  
print(newFile.name)

1. **random** [**module** that generate random number]

* .randint(from, to) [function, generate random integer number]
* .random(from, to) [generate floating random number from (0.0\*\* to 0.9\*\*)]
* .random(from, to)\*5 [generate floating random number from (0.0\*\* to 9.0\*\*)]
* .choice(variable) [get random variable]
* .Shuffle(variable) [shuffle list value]

# from random import randint, random # can write this line as well

import random

random\_int = random.randint(1,10)

random\_f = random.random()

random\_f2 = random.random() \* 10

from random import choice, randint

li = [1,2,3,4,5]

print(choice(li))

print(randint(100,200))

1. **math** [module use for mathematical operation]

* .ceil(val) [bottom value]
* .floor(val) [upper value]
* .factoial()
* .sqrt() [square root]

#.......Math module.........

import math

# from math import function

print(math.factorial(5))            # 120

print(math.sqrt(16))                # 4.0

print(math.ceil(3.2))               # 4

print(math.floor(3.2))              # 3

print(math.log(10))                 # 2.302..

print(math.pow(2,4))                # 16

1. **Datetime** [ ]
2. **Regular expression(RegEx): “**import rm” string handling.. It is use for character pattern.

Search(), match(), sub(), findall(), split(), stc

***2 type of function:***

1. Match Function. Search from beginning

2- Search Function. Searches from whole string

anss = re.search(patt, str)             # search intire string

ansss = re.match(patt, str)             # search from beginning

import re

print(re.sub("IS", "XXXXXX", str, flags=re.IGNORECASE))                # replace sub String

print(re.findall("is", str))           # list of the all occurence otherwish []

print(re.search("is", str))           # tuple range

print(re.split("i", str))                    # return the list....break into the List

1. **heroes, turtle**

………………………………………………………………………………………………………………

**OTHER DATA STUCTURE:** List[], Dictionary(key-value){}, Tuple(), Sets,

* **List: []**
* just like array store **but other type of data as well**…
* ordered data structure..
* arr[index] - access the elements….
* **can modify(mutable) E.g :** name[2] = “new\_name”
* clone list ex: B = A[:]
* list constructor also uses. val = list((1,2,3,4,5)) O/P: [1,2,3,4,5]

# # ......access....slicing...

name = ["pawan", "bisht", "Ravi", 45, 23.23]

print(name[2]) # ravi

print(name[1:4]) # ['bisht', 'Ravi', 45]

print(name[:20])        # upto last 19…… ['pawan', 'bisht', 'Ravi', 45, 23.23]

print(name[-1]) # 23.23

print(name[::-1]) # reverse order

**OR**

# # list constructor..........

val = list((1,2,3,4,5))

print(val) # [1, 2, 3, 4, 5]

# list comprehension..........

list = [x for x in range(10)]

print(list)

***Methods:***

.append(value) [add as a **one element & one List** at last]

Ex: [1,2,3,4,[99,88,77]]

Ex: [1,2,3,4, “RAMA”]

.extend(arr2) [add as a **list items at end** (merge list)]

Ex: [1,2,3,4,99,88,77]

Ex: [1,2,3,4,’R’,’A’,’M,’A’]

clear() [remove all element]

copy(),

count(value) [how many times]

index(value) [but 1st occurrence]

insert(index, value)

pop() [last element remove]

pop(index), or **del** name[0]

remove(value)

reverse()

sort()

split()

L1.extend(L2) [2 list merge]

* **2D List:**

li1 = [1,4,7,9]

li2 = [23,6,8,”prem”]

gene = [li1, li2]

print(gene[1][3])                           # access 2D list element

# iterate……..

for i in gene:

for j in i:

print(j)

* **Tuple: ()**
* same as list……but can’t modify (**immutable**)if once initialized…..
* can change into a list…
* can modify but after converting into a List

**Method:**

tuple((2,4,)) [create tuple]

Count(value) [no of time value (first occurrence)]

Index(value) [but first occurrence]

list(tuple\_variable) [change into a list]

sorted(name) [sort the tuple]

tup = ((2,))

tup = tuple((1,2,3,4,5))

tup = (2,[3,6,8],['pawan','loksg','mohan'])

print(type(tup)) # tuple

print(tu[1]) # [3,6,8]

**Unpacking:**

# Unpacking......assign values to different variable..

tu = (22,44,77,99, 'hina')

a,b,c,d,e = tu

print(a,b,c,d,e)

* **Set: {}**

repetition is not allowed **Unique element**…..

order not preserver…

can’t modify(**unmutable**) any value……..

but can add new or duplicate value by “add()”…

index not support

* **Method:**

add() if already value present then not add....otherwise add on random position

set() list/set/tuple to set change

clear()

copy()

arr1.difference(arr2) different value print on arr1

arr1.difference\_update(arr2) remove items that exist in both sets (update)

a.intersection(b) common value print

a.intersection\_update() which is not present in 2nd set delete that value in 1st list..(update 1st list)

a.isdisjoint(b) if both set value are different return **TRUE**

a.issubset(b) true and false

a.issuperset(b)

a.pop() random element deleted and retrieve

a.remove(value) delete specify element

a.discard(value) delete specify element

a.update(b) a set now large…..same as union but not created new set

val **in** setName (return Boolean….. value present or not)

set1.intersection(set2) OR set1 **&** set2 (intersection of 2 set)

set1**.union**(set2) (union of 2 set)

set1.difference(set2) (set one unique element)

set1.issubset(set2) (TRUE …..if set1 value present in set2)

* **Dictionary:** just like a map….key-value….

can modify values(**mutable**)…..{ }

Key can be only Boolean, tuple, string, int**. ex: {1: “Pawan”, (1,2,3): “Bisht”, true : 1212}**

**But here example true and 1 are same.**

Ordered preserved (after v3.7)

Get element by key(not present error) and get method…. not by index (scliecing)

* **Method:**

Items() return key-object as tuple in list (view-object is the key-value pairs)

dict(keys: values) constructor

Keys() get keys

Values() get values

Items() key-values pair

Pop(key)

Popitem() last element deleted

Clear()

.copy() OR .dict(dic\_name)

get(key) get value at particular key

setdefault(key, value) if key not present new key-value insert otherwise override

update(dictionary) override and who is not present add the that items

or

dict\_name[key] = value update if key not present **error**

fromkeys(x, y) new key value generated

dic\_Name[ keys ] access value of key name

dic\_name[key] = value add value or update

dic\_name.update(key : value) update key - value

del dic\_name[key] delete value by key

key in dic\_name true or false

list(dic\_name.keys()) retrieve all keys and convert them into list (same for values)

list(dic\_name.item()) retrieve all key-value pair as tuple and convert list

x = {"k1", "k2", 3} //keys

y = "Hari" //values

dictionary = dict.fromkeys(x,y)

**zip() :** create dictionary.

# zip().....create method..........

name = {"Apple", "Orange", "Banana"}

price = {120, 70, 30}

fruits = dict(zip(name, price))

print(fruits)

***nested Some examples:***

student\_scores = {"pawan": 77, "Harry": 12, "Radha": 98, "Bawana": 51}

student\_grade = {}

for i in student\_scores:

    score = student\_scores[i]

    if score > 90:

        student\_grade[i] = "A+"

    elif score > 70:

        student\_grade[i] = "A"

    elif score > 60:

        student\_grade[i] = "B"

    elif score > 50:

        student\_grade[i] = "C"

    else:

        student\_grade[i] = "Fail"

print(student\_grade)

O/P:

{'pawan': 'A', 'Harry': 'Fail', 'Radha': 'A+', 'Bawana': 'C'}

#......append dictionary from function...

std\_details = [

    {

        "name": "Ganesh",

        "marks": [12,23,34,12],

    },

    {

        "name": "Omprakash",

        "marks": [33,11,77,99],

    },

]

def add\_fun(name, marks):

    new\_std = {}                    # 1st

    new\_std["name"] = name           # 2nd

    new\_std["marks"] = marks

    std\_details.append(new\_std)        # 3rd

add\_fun("payal", [22,545,999])          # from here

print(std\_details)

O/P: [{'name': 'Ganesh', 'marks': [12, 23, 34, 12]}, {'name': 'Omprakash', 'marks': [33, 11, 77, 99]}, {'name': 'payal', 'marks': [22, 545, 999]}]

* List contains dictionary and also dictionary contains list as well.

Key: [List],

key2 : {Dict}

* Nesting:

# Nested List...........

std\_mrks = {

    "pawan": [75,44,69,77,42],

    "Ome": [12,34,56,89,12,122222],

}

# Nested dict............

std\_mrks = {

    "Person1": {"pawan": [75,44,69,77,42], "age":22, "Home":263632},

    "Person2": {"Ome": [12,34,56,89,12,122222]},

}

# mix nested list.............

details = [

    {

        "pawan": [23,34,12,2],

        "clg": "SLIET"

    },

    {

        "Harry": "Ram shayam",

        "marks": [23,44,{1:"fisrt", 2:"Second"}]

    },

]

……………………………………………………………………………………………………………………………………

**Branching: Conditional operator**



1. If-else

if b>a:

    print("b is greater")

else:

    print("a is greater")

1. Multiple if:

**input()**

if condition:

**input()**

    if condition:

    elif condition:

    elif condition:

    else:

**input()**

    if condition:

    else:

**input()**

    if condition:

    elif condition:

    else:

    if condition:

else:

1. If-elif-else

if b>a:

    print("b is greater")

elif(b<a):

    print("a is greater")

elif(a==b):

    print("a and b equal")

else:

    print("nothing")

1. If-elif ladder

if num>a:

    print("num is greater then a")

    if num<a:

        print("a is greater then num")

    elif num<b:

        print("b is greater then num")

    elif num==30:

        print("num is 30")

    else:

        print("all condition false")

else:

    print("else condition")

# truely value: [2], (1,3), "ba", 7, -23

# falsy value: [], (), '', 0

if [2,3]:

    print("truely value")

else:

    print("Wrong")

1. Match case (switch-case)

# match and case (switch-case).......

match variable:

    case 1:

        print("Sunday..")

    case 2:

        print("Monday..")

    case \_:

        print("no days..")

* **String:**

string is **immutable i**.e. cannot be changeable by index (str[0] = ‘f’ #error). But concatenate 2 or more string by “+” operator.

**Single line**: single quote (‘….’) & double quote(“…”) both can use

**Multiline line**: three quotes (“””…”””) or (‘’’….’’’)

String handing have a “Regular Expression” module. Which is **“import re”**

print(3\*str2)                                  # 3 times print

**String slicing:**

* str[2:10] – access value index 2 to 10
* str[-1] – get last character

str = "pawan bisht IS my Name is me."

print(str[:3])                              # 0 to 2 char. Print

print(str[3:])                              # 3 to last char. print

print(str[6:12]+"single")                   # (6 to 11) update string

print(str[::2])                             # every 2nd character print

print(str[5:12:2])                          # 5 to 12 index, every 2nd character print

print(re.sub("IS", "XXXXXX", str, flags=re.IGNORECASE))           # replace sub String

**comparison : ==, > <**

str2 = "HAD"

str3 = "had"

print(str2==str3)           O/p: FALSE              # string equal or not

print(str2>str3)           O/p: TRUE              # string equal or not

**methods:**

str = "pawan bisht IS my Name is me."

str2 = "HAD"

print(str.capitalize())                 # only 1st charater of string is Capitalized and

other are small

print(str.title())                      # every character starts with upper case

print(str.swapcase())                   # lower to upper and upper to lower

print(str.casefold())                   # string in lower case

print(str.lower())

print(str.islower())                    # TRUE or FALSE

print(str.upper())                      # string in upper case

print(str.isupper())                    # check upper … TRUE and FALSE

print(str.count('is'))                  # number of times character is comming

print(str.endswith('me.'))              # return TRUE & FALSE

print(str.find("bish"))                # index of the 1st occurence otherwish -1

print(str.index('bish'))               # index of the 1st occurence otherwise error

print(str.rfind('pawan'))              # index of the 1st form right side occurence

print(str.rindex('is'))                # index of the 1st form right side occurence

print(str.isalnum())                   # return true if all character are number and alphabates

print(str.isalpha())                   # return true if only alphabates is present

print(str.isascii())                   # return true if only ascii value

print(str.isdecimal())                 # return true if only decimal

print(str.isdigit())                   # return true if only digit are present

print(str.isnumeric())                 # return true if all character are numeric

print(str.isprintable())               # return true if only printable

print(str.isspace())

print(str.join(str2))

print(str.replace('pawan', str2))        # replace with other string

print(str.split())                       # break into the List

print(str.split(“a”))                     # break into the List with a

print(str.lstrip('p'))                   # left side p is trim

print(str.strip())                       # left and right side spaces are removed

print(str.startswith("pawan"))           # check by which character start the string or not

* **Loop:**
* **For loop:**

**for** *variable* **in** *list:*

*statement*

* for loop with else
* **While loop:**

Initialized

**While** *condition*:

*Statement*

* while loop with else

#.............traverse in list.....................

li = [23, 56, 12, 78, 123, 56]

for x in li:

    print(x)

# #.....traverse in string…………

li = "Naveen" # in string index

for x in li:

    print(x)

# #.............break.................

li = [11,22,33,44,55,66,77,88,99]

for x in li:

    print(x)

    if x==55:

        break

#..........range..........

for x in range(5):             # 0 to 4 [1,1,1,1]

arr[x] = 22 # [22,22,22,22]

    #...or.....

for x in range(5, 10):            # 5 to 9

    #...or.....

for x in range(3, 30, 2):           # 3 to 30 but 2-step

    print(x)

for x in range(10, 0, -1):           # 10 to 1 but 2-step

    print(x) O/p: 10 9 8 7 6 5 4 3 2 1

# #......else in for loop....at the end else block executed then loop is finished ………………………………………………………………………..but break stmt is there than else not executed.....

for x in range(2, 10, 3):

    print(x)

else:

    print("end loop")

# #.......nested for loop.........

li = [1, 2, 3]

li2 = ["pawan", 'bisht']

for x in li:

    for y in li2:

        print(x,y)

#.........pass......"for" loop can't be empty but have "for" loop but no content

...........................to avoid getting error put "pass" in the loop

li = [3,4]

for x in li:

    pass

#........while loop........

i=0

while i<=10:

    print(i)

    if i == 6:

        break

    i = i+2

#........while with else.....after executing the else, the while loop will be …………………………………………………………………………….terminated........

i=1

while i<=10:

    print(i)

    i = i+1

else:

    print(f'Now i value is {i}')

    print("now i value is",i)

* **access modifier:**

1. public :

i = 12

def my\_fun():

1. Private: can access private data’s

Syntax: **objectName.\_className\_\_dataOrFunction**

\_\_i = 12

def \_\_fun():

**Ex:**

class Demo:

    def \_\_init\_\_(self, name, age):

        self.name = name

        self.\_\_age = age            # private attribute

    def dispName(self):

        return self.name

    def dispAge(self):

        self.\_\_fun()                # ...\*  this way access private function

        return self.\_\_age           # access private attribute

    def \_\_fun(self):

        print("Hiii")

obj = Demo("Kamal", 12)

print(obj.dispName())

print(obj.dispAge())

print()

print(obj.name)

# print(obj.\_\_age)                # not accessible private attribute

print(obj.\_Demo\_\_age)             # but access private attribute

print()

# obj.\_\_fun()                     # not accessible method

print(obj.\_Demo\_\_fun())           # but access private method

* **Function: def** keyword

1. Build-in (print, range, len round, max, min)
2. User define
3. Anonymous function (one line function)

Method/constructor overloading not supported. But achieve by default parameter method

*Method type:*

1. Instance method 2. Class method 3. Static method

*mutable function arg*: **list, dictionary** (support pass by reference (permanent change))

*immutable function arg:* **integer, string, tuple** (support pass by value (tempra. change))

**Syntex**:

def my\_fun():

    print("Hello World")

my\_fun() # function calling is must

* In python function calling is must with arguments.
* **Default argument value:** if we do not pass value in argument, it is use default value from parameters. Default argument write **at the end after** non-default arguments.

def demo(name="Ravi"):

    print("names are:",name)

demo("pawan")

demo() # get “Ravi”

demo("Harry")

* **Keyword Argument** : send argument with the key=value syntax. **Order doesn’t matter.**

def my\_fun (name1, name2, name3):

    print("My favorite name is: ",name2)

my\_fun(name3="Radhe", name1=12, name2="Pawan")

* **Arbitrary argument:** When we don’t know how much argument then use “ \* “ with a single parameter. Function definition **receive tuple of argument**.

def my\_fun(\*cars):

    print(f'car is: {cars[1]}')   #tuple of argument

my\_fun("BMW", "audi", "Tyata")

* **Arbitrary keyword argument: {\*\* }** When we don’t know how much **key=value** **argument** then use “ \*\* “ with single parameter. Order doesn’t not matter. Receive a dictionary of argument.

def demo(\*\*name):

    print("names are:",name["n1"]) # receive a dictionary of argument

demo(n2="Pawan", n4=2332, n1=12.12, n3=True)

* **Passing list as an argument:** can send any type of data string, int, list, dictionary etc

name = ["pawan", "Harry", "Ravi"]

def demo(name):

    for x in name:

        print(x+" ")

demo(name)

* **Return multiple values statement**: butreturn as a tuple

# return multiple function: (but in the form of tuple)

def fun(age, name, course):

    return age, course, name

ans = fun(12,"Harry", "JAVA")

a,b,c= fun(12,"Harry", "JAVA")

print(ans) # (12, harry, JAVA)

print(b) # JAVA

* **pass keyword:** if we want to give definition later that time use.
* **Lambda function: should a single expression**

***syntax***: variable = **lambda arguments,.. : expression**

***call***: variable(parameters,..)

# lambda function:  a,b,c is argument

func = lambda a,b,c: c\*a\*b

print(func(2,3,4))

# largest number.

(lambda a,b: print(a) if a>b else print(b)) (2,3)

* **file handing**

when we create file **close()** is must.

But by using “**With**” statement it automatically closes the file when indented block are completed. It helps manage resources efficiently.

Find file name : file\_variable.name

Find file mode : file\_variable.mode

If file open in **read mode** 1st closed.. then open and **write mode** and vice-versa. At a time one operation can do.

*File type:* 1. Text 2. binary

**Permissions (mode)**:

“r” = read only

“w” = write value (override)

“a” = append values at the end of file

“x” = create file for writing if already exits raise error

“rb” , “wb” = read/ write binary mode

“r+” “w+” “a+” = open and create both reading / writing mode

**Function**:

**Open(“path/file\_name”, “permision”)**

.close()

.write(variable)

.readlines(vari) read line by line

.readline() read 1st line

.writelines()

.tell() check position of file handle position

.seek(int) new position choose

# ........write()...........

fil = open("demo1.txt", "w")                # open(“file\_name”, “permision”)

fil.write("I am Wring on a file..OKK")

print("created file..")

fil.close()

str = "op  op o p o po popopo p op "

with open("demo.txt", "w") as file1: # automatically close file

    file1.write(str)

    file1.write("Pawan pawan \n bisht")

print("written")

# ........read()...........

fil = open("demo1.txt", "r")

filCon = fil.read()

print(filCon)

fil.close()

with open("demo1.txt", "r")) as f: # automatically close file

filCon = f.read()

print(filCon) # close file here

* **Exception handing**

**try**:

## code

**except** ExceptionType **as** vaiable:

## code

**except** :

## code

**else**:

## if try block executed

**finally:**

## always executed

a = int(input("Enter 1st value: "))

b = int(input("Enter 2nd value: "))

try:

    c = a/b

    print(c)

except ZeroDivisionError as e:

    print(e)

else:                               # if except block not print

    print("else block")

finally:                            # compolsury print

    print("end")

**OOPS:**

* Python is an object-oriented programming language.
* Class – PascalCase
* Call function but parameter list have “self” keyword…
* Class attribute always initialized. And call by class name
* Not supported method overloading…
* Access private attribute and function: ***syntax***: **obje.\_className\_\_privateAttributeOrFunction**

**Crete Class && object:**

When we declare any data (class variable) in the class must define value.

class Student:              # class create

    name = “Pawan” # must initialized...class variable

s1 = Student()              # s1 is object (object create)

**Constructor**: special method used to create and initialize an object

* define by “ \_\_init\_\_ “
* “self” is always 1st parameter (“self” is just line “this” keyword)
* Only one constructor create (can create more then one but last one is consider)

def \_\_init\_\_(self, name, id):             # constuctor

        self.name = name # instance varibale

        self.id = id

**class variable:** comman for all object(change not for single object)

# class variable

class Demo:

**data = []**            # comman for all object(change not for single object)

    def \_\_init\_\_(self, name):

        self.name = name

**Demo.data.append(self.name)**

    def fun(self):

        return self.name

obj = Demo(11)

print(obj.fun())

print(obj.data)

obj2 = Demo(99)

print(obj2.fun())

print(obj.data)

**Inheritance: type- simple, multilevel, multiple, hierarchical, hybrid**

* Syntex: **Child (Parent)**
* Call super class: **parentClass . \_\_init\_\_(..)**
* Multiple inheritance: ***syntax***: class Man(A, B)
* If same method are written in difference class that time first method get firstly

Can check order: ***syntax***: className . \_\_mro\_\_

class Person:

    def \_\_init\_\_(self, name, id=11):

        self.name = name

        self.id = id                    # initial id value is 1

    def dis\_parent(self):

        print(f"Parent class. {self.id}")

class **Man(Person):**

    def \_\_init\_\_(self, na,id,age):

        # Person.\_\_init\_\_(self, nam)   # or super() (no need to use self) || here call to parant class

        super().\_\_init\_\_(na)

        # self.na = na

        self.id = id

        self.age = age

    def dis\_child(self):

# access perent data

        print(f"Child class: {self.id}, {**self.name**}, {self.age}")

emp = Man("Harry", 101, 34)

emp.dis\_parent()        # can access parent function

emp.dis\_child()

**Polymorphism**:

***Method overloading is not supported in Python*** *because it is interpreted language(line by line i.e. last line executed).*

* Method overriding supported but by “super()” function call to parent class

class Poly:

    def fun(self, a):

        print("Poly1 class...",a)

class PM(Poly):

    def fun(self, a):               # overriding....

        print("PM class...",a)

        super().fun(10)                 # call to perent call by "super()"

p = PM()

p.fun(23)

other way:

# # using Duck typing concept

class Poly:

    def fun(self, a):

        a.fun2()

class Poly2:

    def fun2(self):

        print("Hii")

class Poly3:

    def fun2(self):

        print("Hello")

p = Poly2()

q = Poly3()

obj = Poly()

obj.fun(p)

obj.fun(q)

**Abstraction:**

* Use “pass” in function bo

# Abstaction...........child class should implemented......

from abc import ABC,abstractmethod

class Animal(ABC):

    @abstractmethod

    def eat(self):

        pass

class Dog(Animal):

    def eat(self):

        print("Dog clss..")

# a = Animal()           # abstract class can't instantiate

a = Dog()

a.eat()

**Encapsulation:**

* “\_”: protected (data can access other class)
* “\_ \_”: private

class A:

    \_\_a = 10                      # private data

    \_b = 555                      # protected data

    def \_\_init\_\_(self,b):

        self.\_b = b

    def fun(self):

        print("A class: a value",self.\_\_a, ",& b value",self.\_b)

class B(A):

    def \_\_init\_\_(self,x):

        super().\_\_init\_\_(x)                      # call to parent class

    def disp(self):

        # print("B class",self.\_\_a)              #can't access private data

        print("B class:",self.\_b)                 #can access protected data

rv = B(22)

rv.fun() O/P: 10 & 22

rv.disp() O/P: 22

**Aggregation:** if we pass class object in constructor .. when we initialized constructor..

**Composition:**

class Engin:

    def enginDetails(self):

        print("Print engin details..")

class Tyre:

    def tyreDetails(self):

        print("Print tyre details..")

class Car:

    def \_\_init\_\_(self):

        self.en = Engin()               # create object

        self.ty = Tyre()

    def detailes(self):

        self.en.enginDetails()                  # imp

        self.ty.tyreDetails()

obj = Car()

obj.detailes()

**Monkey patching**: change the behavior of function

# monkey patching..........change the behavior of funtion at run time

class Demo:

    def \_\_init\_\_(self, name):

        self.name = name

    def fun(self):

        print("My name is: ",self.name)

    def f1(self):

        self.fun()

    def f2(self):

        self.fun()

obj = Demo("Harry")

obj.f1()

obj.f2()

def new\_fun():

    print("Update")

obj.fun = new\_fun           # change the proterties of funtion

obj.f1()

obj.f1()

**Completed 54**

**Codes:**

**1. Second largest:**

class Solution:

    def solve(self, A):

        # Initialize the largest and second largest elements

        largest = float('-inf')

        second\_largest = float('-inf')

        # Traverse the array to find the largest and second largest

        for num in A:

            if num > largest:

                second\_largest = largest

                largest = num

            elif num > second\_largest and num != largest:

                second\_largest = num

        # If second largest is still negative infinity, no such element exists

        return second\_largest if second\_largest != float('-inf') else -1

A = [2,5,8,9,1]

Solve(A)