Python programming language

→ Variables

* To print the statement → print ("The statement").
  Variables
* To know type of the variable

Eg >> name = "John"

then

>>> type(variable-name)

input → >>> type (name)

output < class 'variable_type'>

means < class 'str' >

Python Variables :-

→ assinging a single value to variable

a = 10

age = 45 then >> print (a)
>> print (age)

output
a
45

→ assinging a multiple vaules

Eg a = b = c = 10

x, y, z = 10, 20, 30

then >> print (a)
>> print ("b")
>> print ("x")
>> print ("z")

output

>> 10
>> 10
>> 10
>> 30

python tokens

→ Operators, There are 7 kinds of operators

* Arithmatic operators @
  * Ny for c language (same in C)

* Assignment Operators ✓

Operator	Operation Eg
=	x = 10
+=	x = x + 2
-=	x = x - 29
*=	x = x * 12
/=	x = x / 3
|=	x = x | 6

* comparision operator [to compare the vaules]

Operator	operation
==	Equal
!=	Not Equal
<	less than
>	greater than
>=	greater than or Equal to
<=	less than or Equal to.

* Logical Operators [are used to combine conditional statements]

Operator	Description
and	if both are true

or — tr if one of the st is tr

not — if tr, the return False.

Eg of Syntax Operations

```
>> x = 15
>> y = 16.
>>> x = = 15
True
```

```
>>> y != 16
False
```

[statement] by this true state ment says y is not 16

```
>> x = = 15 and y = = 16.
>> True.
```

Types in python

integer, float, string, boolean, list,

Eg

list → [1, 2, 3], ['a', 100]

tuple → (1, 2, 3)

dictionary → {'programming python', 'powerful Obj oriented'}

Set — {'Harvard', 'MIT'}.

↳ here Elements are {'x'}

x is elements

→ Elements in set are seperated by,

String → 'fish', '24b'.

Type conversion

int() → Eg

```
>>> int(5.7)
>> 5.
```

float() → Eg

```
>>> float(2)
>>> 2.0
```

Strings → represented as [str] ' ' = S_1

* to find the lenght of String

```
>>> So = 'some words'
>>> len(So)
>>>
```

Different Kinds of String processings

```
>>> so = 'some\n words'
>>> so
   some\n words
```

but

```
>>print (so) → Input
some
words → output
```

```
>>> str1 = 'very nice'
>>> str2 = 'and good'
>>> print (str1 + str2)
```

very niceand good → here no space

to get space.

```
>>> print (str1 + ' ' + str2)
   very nice and good
>>> school = 'Trips'
>>> print (school[0]) → by this we can access a string containing characters.
T
>>> print (school[-1])
s
```

* >>> school = 'Quantum Energy'

>>> print (school [0:7]) → This was we can get access to part in String.

Quantum

→ here 0:7 means, accessing 0th element to 7th element in String.

* to find wheather alaphabet is present in String or not by

>>> so = 'I have car'

>>> so.find('e')

5 → as per syntax 'e' at 5th position in String so.

>>> so.find('have')

2 → as h in 2nd element as per syntax so it represent give ouput as 2

Lists → represent / accessed by

variable name (Eg lst, a) = [ ]

* to add a element to list

>>> lst1 = [ ]

to add 2 to "lst1" list then

>>> lst1.append(element)

⇒ >>> lst1.append(2)

>>> lst1

[2].

lly we can add elements as many as we can.

* to find lenght of list is lly for both list and string.

i.e Eg lst1 = [2,3]

>>> len(lst1)

2

⇒ both strings and list have same functions to access the list and str elements and modify them.

* to Insert an element at desired place in list

Eg >>> lst1 = [2,3]

>>> lst1.insert(1,10)

>>> lst1.insert('position no', element)

Eg (0, 1) (π, t, 2, 4 etc)

>>> lst1.insert(1,10)

>>> lst1

[2, 10, 3]

0 1 2

(*) to remove the element in list is

Eg >>> lst 1 = [2, 10, 3]

>>> lst 1 . pop (position element want to remove).

```
>>> lst 1 = [2, 10, 3]
>>> lst 1 . pop (1)
  10
>>> lst 1
  [2 , 3] .
```

(*) We can replace the elements in list with another element.

Eg >>> lst = [2, 8, 10]

>>> lst [position of element as per syntax, that want to be changed] = [element]

element that what to be place in list.

```
>>> lst = [2, 8, 10]
>>> lst [2] = 5
>>> lst
  [2, 8, 5] .
```

(**) → List can store strings too:

(*) to clear the all elements in list can be done by

list Variable name . clear ( ) .
(means list name)

>>> lst = [2, 4, 5]

>>> lst.clear()

>>> lst → due this function, it erases all elements in list.

[]

*** → As Operations are done with elements as numbers (integers) we apply same concept elements as strings..

Eg >>> lst = ['calcium', 'Helium']

>>> lst.append('oxygen')

>>> lst

['calcium', 'Helium', 'Oxygen'].

This way we can apply all similar functions.

* as in strings we can do in list too.

i.e >>> lst = [2, 'Helium', 'calcium']

>>> lst[1:2]

['Helium', 'calcium']

A small change in apply [x:y] in String ↪ type and lists is

Eg >>> str1 = 'Apple' ↓

>>> str1[1:4] ⇒ ppl.

but in list.

>>> lst = ['H', 'K', 'P', 'T', 'A']

>>> lst[1:4] → by this type access to only those praticular ones them

⇒ ['K', 'A']

* Here list can store lists as elements

Eg lst1 = [[1,4], [2,5], ['a','b']]

→ here lst1 → is a list

lly [1,4], [2,5], ['a','b'] are list.

→ In this type of list we access the element present list of list one.

Eg consider above Eg lst1

lst1 = [[1,4], [2,5], ['a','b']]

by ⇒ lst1[ ][ ] → This for accessing element present in accessed list

↳ this position used for accessing List present in List1

>>> lst1[1][0]

Functions

* Define a function

def name(arg1, arg2, ---argN):

↳ name of function Eg pop, find('1'). etc

Eg defining a function of our own.

step 1 → select a new directory.

eg

name the directory as per use.

step 2 → Create new python file

then we want to write function finding angular velocity of object.

then

```
def angular vel(vel, rad):
    return vel/rad
w = angular vel(5, 2)
print(w)
```

↳ this will be in file.

after console.

2.5 is answer

Relational Operators

	meaning	Eg (rep in a, b)
<	Strictly less than	a < b.
<=	less than or Equal	a <= b.
>	Greater than	a > b.
>=	Greater than or Equal	a >= b.
==	Equal	a == b.
!=	not Equal	a != b.
is	Object Identity	a is b.
is not	negated Object identity	a is not b.

→ These are used to compare Objects.

Console

```
>>> 3 < 4
True.
>>> 4 < 3
Flase
>>> 3 > 4
Flase
>>> 3 == 3
True
>>> 4 != 3
True.
>>> 3 != 3
Flase.
>>> 4 <= 4
True
>>> 4 >= 3
True.
>>> 'me' == 'me'
True
>>> 'me' == 'Me'
Flase
>>> 'Alex' < 'BOB'
True.
```

(**) To sort the elements in list

```
>>> a = [1, 3]        [to arrange in oder]
>>> b = [3, 1]
>>> a == b
False
>>> b.sort(), >>>b
[1, 3]                [1, 3]
>>>a == b
True
```

Usage of Logical Operators

Console 1

```
>>> p = 2>1
>>> q = 3>2.
>>> p, q
(True, True)
>>> p and q
True
>>> not p
False
>>> p or (not p)
```

Console 2

```
>>> 1<2<3 = 1<2 and 2<3
True
```

Execution mechanism

1<2<3 = 1<2

↳ True True

then True = True

⇒ True

2<3 → True.

True and True

⇒ True.

Another console. (create a python file).

```
def funct1(vaule):
    print("funct1 called")
    return value.
```

```
>>> def func2(return_val):
    print('func2 called')
    return return_val

>>> a = func1(True) and func2(True)
```

```
>>> print(a)
```

Output after running this (file.py). is

True.

because of func1(True) it returns (True) so for func2.

Conditional branching (Loops)

→ if-else Statement

Program to find airthematic no. even (or) odd.

```
def checkEven/odd(a):
    if a%2 == 0:
        print("%s is an even number" %a)
    if a%2 != 0:
        print("%s is an odd number" %a)

checkEven/odd(5)8.
```

Output :- 5 is an odd number.

Binary numb

Binary Numbers :-

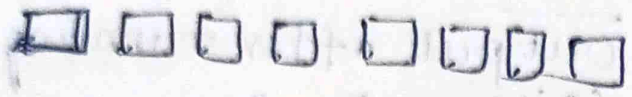
1010 ⇒ decimal number

↳ in binary base

$1\times2^3 + 0\times2^2 + 1\times2^1 + 0\times2^0$

$= 8 + 2 = 10$.

1 Byte = 8 bits



so when max +veno

⇒ 01111111
(5 4 3 2 1 0)

⇒ $2^6 + 2^5 + 2^4$

** while loop:

* conditional branching if-else statement
⇒ here else and if are used some what different way compared with C

Eg programe to find the day of week.

```
def week of day(a):
    if a==1:                    → if [condition statement] :
        print("Monday")
    else:
        if a==2
            print("Tuesday")
        :
        :
    else: if a==6:
        print("Saturday").
```

```
else:
    if a == 7:
        print ("Sunday")
else:
    print ("not defined input")
```

→ Instead of this elif (statement/cond)

Shortcut

```
def week of day(a):
```

output as a=9, so.

not defined input

** `else: if a==7:` == `elif a==7:`

* Loops

while Loop :-

Usage of while loop by defining function to print numbers from 1 to n

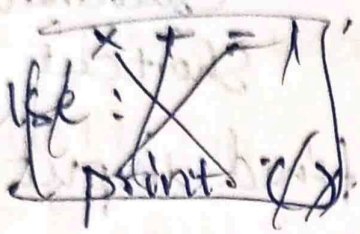
program

```
def naturalnumseries(n):
    x = 1
    while x <= n:
```

loop → while x<=n → # condition/ Statement

print (x, end = ' ')

here [end = ' '] used for printing the number in same line with gap.



x += 1 # [Increment condition : analogous to x++ in C] or [x = x+1].

else:
print().

print("Exit loop x is %d", % x)

def natural num series (12):

Output

1 2 3 4 5 6 7 8 9 10 11 12

Exit loop x is 13.

For Loop:

* maximum time "for loop" is used to access the elements present in lists and strings.

* different cases of using for loop

Ex program [to ACCess elements]

Input digits = ['a', '1', '2', '3', '4', '5', '6', '7', '8', '9'].

for x in ~~range~~ digits: # this is used to access the elements one after another by "for loop" as x, i.e x='a', x='1', --- etc

print(x, end=' ')

output :-

a 1 2 3 4 5 6 7 8 9

* ~~Loop for~~ "for loop" using as print series of numbers or Sequence.

Ex program

Input

for x in range(1, 10, 2):

for x in range(Strat, End, increment):
↳ loop
print(x, end=' ').

output # (here loop Exit at x=10).

1 3 5 7 9

* Ex program for using for loop to print total no. of permutations from (1 to 3).

Input → (*)

```
for i in range(1,4):
    for j in range(1,4):
        if i==j
            continue # continue means to continue in loop.
        for k in range(1,4):
            if k==j or k==i
                continue
            print(i,j,k)
```

Comments:-

* continue :- continue the loop used by leaving that case.

Ex

```
for i in range(1,4):
    for in range(1,4):
        if i==j
            continue
            print(i,j)
```

Output of this program

```
1 2
1 3
2 1
2 3
3 1
3 2
```

here when i=1, j=1 the loop continues and doesn't execute print() function

Output →

1 2 3	2 3 1
1 3 2	3 1 2
2 1 3	3 2 3

* ~~Act~~ Argument passing

COMPUTATION

Ex program to print of fabinoseries upto n.

```
def fabinoseries(n):
    if n<=0:
        print("%d should be greater than 0",%n)
    elif n==0:
        print("0")
    elif n>=1:
        %n1=(%n-1)+(%n+2)
    if n==0:
        print("value error")
    if n>1: fn-1=1
            fn-2=0
        if n==1:
            return fn-2
```

```
if n == 2:
    return fn-2.

fn = 0
i = 2
while i <= n
    fn = fn-1 + fn-2.
    fn-1 = fn-2
    fn-2 = fn
    i = +1
return fn
```

fn-1 = fn-2.

$\Rightarrow f_{n-1} = 1$

f_{n-2} → Empty.

The value of f_n shifts to f_{n-2}

$\Rightarrow f_{n-2} = [f_{n-1} + f_{n-2}]$.

now f_n = empty

```
for i in range(1, 16):
    print(fabinoseries(i), end = ' ')
```

Modules :-

* Module → a file i.e where provides certain functionality means specified functions like numpy module for mathematical operations.).

Eg program to importing numpy → (library/module)

Ex creating array.

```
import numpy as np.
```

num of points = 20.

xArray = np.linespace(-3, 3, number of points).

print(xArray)

np.linespace(-3, 3, number of points)

↳ module ↳ function to create array. ↳ strating ↳ end ↳ no. of points in

Output.

[-3, - - - - - - - - - - - - - - - - - - , 3]

18 elements (any random way)

Ex -2.1, 1.512, etc.

* Solving Linear Equation using "numpy" module.

→ Ex program:- import numpy as np.

solve.

3x + 2y = 5

2x + 4y = 6.

a = np.array([3,2], [2,4])

b = np.array[(5,6)]

x = np.linalg.solve(a, b)

print(x).

* plotting curves/graphs using
  * numpy module
  * matplotlib.pyplot module → required for graphical representation

Exprogramme

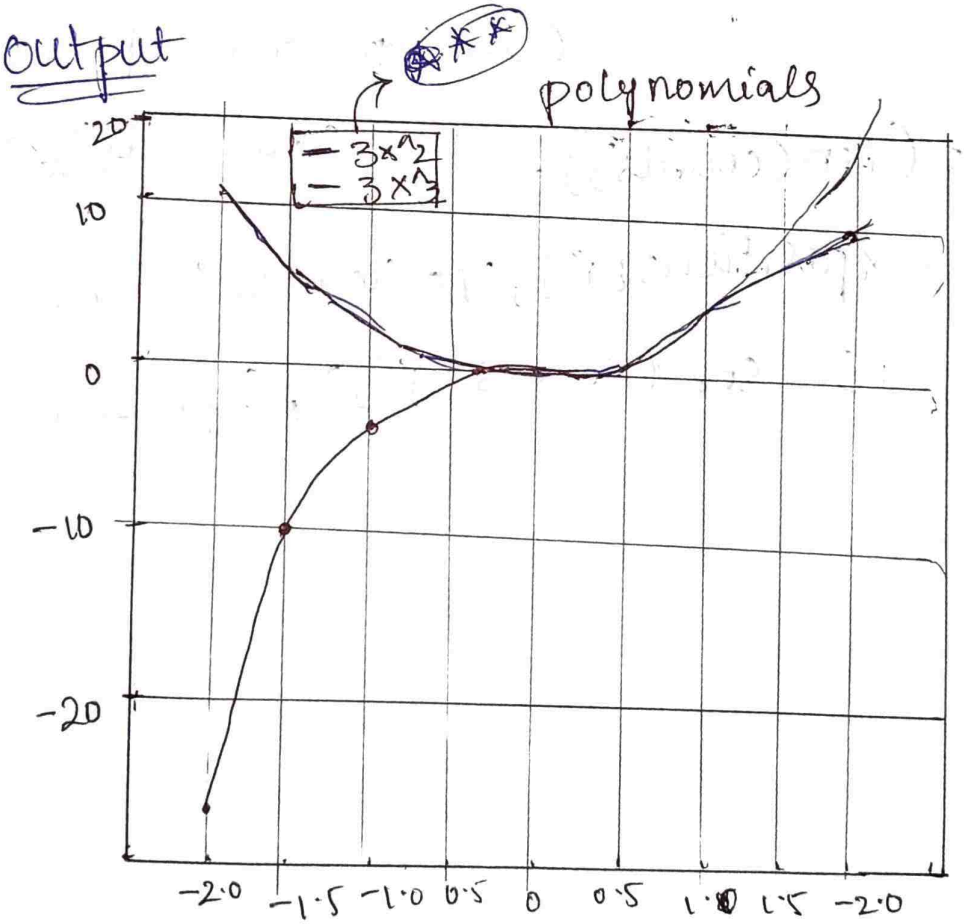
```
import numpy as np.
import matplotlib.pyplot as plt.

def func(x):
    return 3 * x**2 # return 3x^2.

def func1(x):
    return 3 * x**3 # return 3x^3

# After function, we should need values of x
  so for creating it, so take 50 points.

no.of.points = 50.

~~xArray1 = np.zeros(num.of.points).~~
~~yArray2 = np.zeros(num.of.points)~~
~~xArray = np.zeroes(num.of.points)~~

xArray = np.linespace(-2, 2, num.of.points)
# creates Array of elements containing num.of
                                    points.
yArray1 = np.zeroes(no.of.points)
yArray2 = np.zeroes(num.of.points)
for i in range(numof points):
    yArray1[i] = def func(xArray[i])
    yArray2[i] = def func(xArray[i])
```

Similarly in C language, here also Arrays elements stored in same way

for i in range(x): / means loop continues
- for i=0, to i= x.

plt.title('polynomials') # title of graph
plt.xlabel('x') # naming for x axis
plt.ylabel('y') # naming for y axis
plt.grid(True)
plt.plot(xArray, yArray1) } # These
plt.plot(xArray, yArray2) } plt.plot(x, y) for used for plotting
plt.legend(['3x^2', '3x^3']) # for showing into
** → to get this.
plt.show() # to show the plotted graph.

Output



Ex programme to visualise data in piechart and barchart

programme

```
import matlib.pyplot as plt.
# bar chat.
xPosition = [1, 2, 3, 4, 5]
charcters = ['a', 'e', 'i', 'o', 'u']
Counts = [10, 20, 30, 15, 8]
barwidth = 0.3.
Plt.bar(xposition, Counts, width = barwidth)
# Text below each other.
plt.xticks([0.7+r+barwidth for r in range
                    (len(xpositions))],
for i in range (len(counts)):          charcharts).
  plt.text (x = xpositions[i], y = counts[i]+0.2,
            S = str (counts[i]), size=7)
```

Binary Search

Its like func or algorithm to find element in array or list. this useful to find a word / while processing with large data

Ex programe

```
def binarySearch (key, arr):
    arr = sort.arr
    ui = len(arr) -1
    li = 0
    while ui <= li # This used because loop continues until it returns true.
    mi = ui + li //2
         2
# (// is used to get Integer vaule).
    if arr[mi] == key:
        print ("key word found").
        return true.
    elif:
        if arr[mi] > key:
            ui = mi-1
        else arr[mi] < key:
            li = mi + 1
    else : print ("keyword not found").
```

Recursion

- Recursion is a concept, where we call the function in its same defined function. (same)

Eg program:

```
def get():
    print("hello")
    get()
```

When runs

```
Hello
:
:
Hello
```

→ praticaly it should print ∞ times.

* (:. but recursion in python was limited to 1000 times beyond that we should use some special custumisazation).

* The process custumisze to extend the recursion as many times we want.

Ex program to make recursion for 2000 times

```
import sys
sys.setrecursionlimit(2000)
# above these 2 lines of code make recursion
# 2000 times to happen
```

- finding factorial of n using recursion

```
def factiral(n):
    if n == 1 or n == 0:
        return 1
```

```
else:
    return n*(n-1)*factiral(n-1)
    # we called again func in it self
def factiral(5):
output result = factiral(5) # because we
print(result) # had returned the vaule
```

so that value is assigned to "result" variable and wrote a code to print "result" so we can see the value

~~Simple Sorting~~

Break, Continue, pass Statements in python

Break, this is used for breaking the continuity of break.

Eg program

```
for i in range(5):
    if i==2:
        break
    print("this is", i)
```

Out put

```
# This is 0      # when we use "break"
  This is 1        the loop continuity
                   breaks.
```

Continue :- This used for, when given condition statisfied, at that instance in loop shifts and continue further loop without executing at that instance.

Ex programme

```
i = int(input("enter i: "))
x = 0
while x < i:
    if x == 2:
        continue
    print("Hello", x)
    x += 1
```

Output

```
enter i: 4
Hello 0
Hello 1
Hello 3
```

Here when x == 2 statisfied the loop skiped to next without excuting.

Pass : this used for, when we donot have clarity at cretain func or, loop we use pass or class we use pass statement so we can write code later.

Ex program

```
x = 0
if x > 5:
    x += 1
print(x)
```

```
def func():
    pass
```

Output

```
1
2
3
4
5
```

here we do not get any Syntax error at def func(): line because we used "pass" statement so it says to compiler that we can make alterations anytime.

Patterns in Python :-

→ for printing

```
# # # #
# # # #
# # # #
# # # #
```

program

```
for i in range(4): # i-strats 0 eg end at 3
    for k in range(4):
        print("*#", end=" ")  # to print # # # #
    print() # → to print in new line
```

→ for printing

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

```
for i in range(5):
    for k in range(4):
        print("* *" * k)
    print()
```

Array in python

* Advantage of Arrays in python is, the size of array is dynamic.
* Array Similar to list but class-type is fixed for respective array..

→ Using array.

Eg: program

```
import array as arr
# (from array import *) → This means we can use all functions in array module.
from array import *
# creating array
vals = array('i', [5,9,8,4,2])
# variable = array('typecode', [ ])
print(vals).                    values to be entred
print(vals.buffer_info())# out shows
```

Output (address of array, len of array)

```
[5,9,8,4,2]
(87142,5).
```

* User input array :-

programme

```
from array import *
arr = array ['i', [ ]]
n = int (input ("enter len of array "))
for i in range(n):
        x = int (input ("enter the vaule"))
        arr. append (x)
print (arr)
```

output

```
enter len of array 4
enter the value 16
"  "  "  21
"  "  "  23
"  "  "  10.
array ('i', [16, 21, 23, 10])
```

Python - Object oriented programming

* In object oriented programming "functions" are called as "methods"

* Concepts related objected oriented programming are
  i) * objects
  (ii) encapsulation
  (iii) class
  (iv) Abstraction
  v) polymorphism.

Class in python

creating class / defining class.

Eg program

```
class variablename:
```

Attributes → Variables
Behaviour → methods (functions).

} General structure of Class.

Programmes on Class

①
```
class computer:
    def config(self):
        print("i5, 16gb, 1TB")

a = '8'
print(type(a))
com1 = computer()
print(type(com1))
```

output → This is inbuilt class/object

<class 'str'>

<class '__main__.computer'> → This is defined one. class/object

as com1 is a object of class.

2) class computer:

```
    def config(self):
        print("i5, 16gb, 1TB")

com1 = computer() # created object

computer.config(com1) #
com1.config() #
```

→ These are 2 ways to call methods in object but mostly 2nd one com1.config() is used in syntax.

output

i5, 16gb, 1TB

i5, 16gb, TB

3) Special type of variable in class

When we use special variable, it calls itself without calling it.

Ex programme

```
class computer:
    def __init__(self): # __init__ special variable
        print("in init")
    def config(self):
        print("i5, 16gb, 1TB")
```

```
com1 = computer()   # created com1 and
com2 = computer()     com2 objects.

com1.config()
com2.config()   # here we are calling
                  config method but not
                  __init__ method.
```

Output

```
in init
init init
i5, 16gb, 1TB
i5, 16gb, 1TB
```

→ # this output is not caused by our code, it caused due to special variable.

→ # This output caused by com1.config()
com2.config().

Creating objects and giving attributes to class (OBJS) in.

* The main thing is while assigning attribute to method (function) in class we have to write "self.attributename" to access the ~~attri~~ attribute in class.

Eg programme is to 2 objects with attribute (i5, 16gb ram) and (Ryzen 3, 16gb ram).

Sol

programme

```
class computer:
    def __init__(self, cpu, ram):
                # cpu and ram are attribute
        self.cpu = cpu
        self.ram = ram
```

to self.attribute name is written to accesses the attributes

```
def config(self):
    print("config is", self.cpu, self.ram)
# Objects creations
obj1 = computer('i5', '16gbram')
obj2 = computer('Ryzen3', '16gbram')
obj1 = computer('i5', '16gbrams')
obj2 = computer('Ryzen3', '16gbrams')
# calling objects
obj1.config()
obj2.config()
```

output

config is i5 16gbram
config is Ryzen3 16gbram

* "id function" is a function to find the address of attribute in class (or) variable.

Ex programe

```
class computer:
    pass          # here pass statement, because we have not defined anything, if we donot use pass statement without creati defining method, we get error.

c1 = computer()
print(id(c1))
```

Output

5119240

* Changing attribute Vaules/Data in Objects.

Ex programme

```
class computer:
    def __init__(self):
        self.name = "Navin"
        self.age = 28
c1 = computer()
c2 = computer()
```

here c1 and c2 Objects are created.

here c1 and c2 object same data. but stored in different addresses.

```
c1.name = "Rashi"
```

when use c1.name it accesses name attribute in c1 object and changes to "Rashi"

print(c1.name, c1.age)
print(c2.name, c2.age)

Output

Rashi 28 # here only name attribute
Navin 28 change.

*types of variables in Oppis

in Oppis 2 variables < Instance Variable
Class Variable

*Programme to differenide b/w instance and class variable

Ex programme

```
class car:
    wheels = 4 # -> here class variable.
    def __init__(self, c, m)
        self.colour = c
        self.model = m } # These are Instance variable.

car1 = car(Red, 5.7)
car2 = car(White, 2.5)
print(c1.colour, c1.wheels)
print(c2.c, c2.wheels)
```

Output Red 4
White 4
} # here we can use class (or) object name to call the class variable

* programme to change vaule of class variables:-

```
class car:
    wheels = 4
    def __init__(self):
        self.colour = "Red"
c1 = Car()
c2.car.wheels = 3 #(class.class variable to access the
```

from now in class vaule).

vaule of wheels changes from 4 to 3.

```
print(c1.colour, c1.wheels, car.wheels)
```

Output Red 3 3

here as mentioned in back page

* Different types of methods in class

→ Instance methods → deals with instance variables
→ class methods → deals with class variables
→ Static methods → deals variables other than class and instance.

* Ex programe

→ next page

```
class Student:
    def __init__(self, m1, m2, m3)
        self.m1 = m1
        self.m2 = m2
        self.m3 = m3.
def avg(self):
    return (self.m1 + self.m2 + self.m3)/3
```

this is Instance ~~variable~~ method, whole defing method no need use any special variable.

```
@classmethod # decorator to access for cls method.
def getschool(self):
    return cls.school # calling class variable.
@staticmethod # decorator for static method
def info():
    print("This student class -- in abc molude").
S1 = Student(34, 47, 32)
S2 = Student(89, 32, 12)
print(S1.avg()) # for calling class func
```

{Object.function name()}

```
print(Student.getschool())
```

for callin method ⇒

Inner Class

Creating a class inside already defined class.

→ ex programme

```
class Student:
    def __init__(self, name, rollno):
        self.name = name
        self.rollno = rollno
        self.lap = Laptop()
        # creating inner class, for access to we should created Instance variable, Shift the data of class to Instance variable.
    class Laptop:  # inner class
        def __init__(self):
            self.ram = "4Gb"
            self.cpu = "i5"
# to print the cpu value of inner class
s1 = Student()
print(s1.lap.cpu)
```

output

i5

Inheritance :-

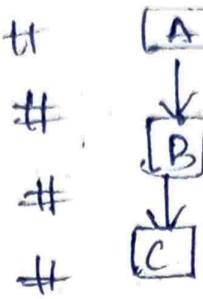
→ This was used to import / Inherit the features & from one class to another. like parent habits to children.

Ex program

```
Class A:
    def feature1(self):
        print("Feature 1 is working")
    def feature 2 (self):
        print("Feature 2 is working")

class B(A): # here Class B is Child (of A) that means
            # ans B it had access to all
            # variables and methods.
    def feautre 3 (self):
        print("feautre 3 is working")

class C (B): → # parent class of C.
    def feautre 4 (self):
        print("feaurtre 4 is working")
# here from class C we get access to all
instant and variables (methods) of A and B.
#if here
```

```
print(
c1 = C()
c1.feature1()
c1.feature3()
```

for understanding.



Output

feature 1 is working # output instance of Class A

feature 3 is working # ouput instance of class B.

* multiple inheritance

When we want to access instances of two different classes, those 2 classes are not related. to a different class

Ex programme

```
class A:
    def f1(self):
        print("f1 is working")
class B:
    def f2(self):
        print("f2 is working").
```

multiple inheritance

```
class C(A, B):
    def f3(self):
```

print ("f3 is working").

#so here class C is inherited with A and B classes.

C1 = C() # object creation

B1 = B()

A1 = A().

C1.f2() # calling the instances.

B1.f2()

C1.f1()

A1.f1()

output

f2 is working # called from C object

f2 is working # called " B object

f1 is working # " " C object

f1 is working # " " A object

* ~~POLYMORPHISM~~

*Constructors In Inheritance

when we inherit from parent classes and create a object for sub class then we call the object of sub class it checks whether "__init__()" is in sub class or, if not there, then it moves to super class (parent class) and excutes

that function

Eg programme **

```
class A:
    def __init__(self):
        self.obj = 45
        print(obj)

class B:
    def f1(self):
        print("f1 is working")

class C(A,B):
    def f2(self):
        print("f2 is working").
```

@ c1 = C() :-

~~print~~ (c1.

c1.__init__()

Output

45 # Obj instance in class A.

** Accessing method of Super class by creating method in Sub class

(∴ Super class = parent class)

(∴ Sub class = inherited class).

Eg programme:

```
Class A:
        def --init-- (self):
            print ("A is accessed")

Class B (A):
        def --init-- (self):
            Super().--init--().
    # Super() in constructor to access the
    inster methods of Supper class.
            print ("B is accessed").

b1 = B() # Object is created.
```

Output:

A is accessed
B is accessed

} # here to access to init method, no need to call the --init-- method specially, when we run the code, it calls by itself.

* MRO → Method Resolution Oder.

```
class A:
    def __init__(self):
        print("A is accessed").

Class B:
    def __init__(self):
        print("B is accessed").

Class C(B,A):
    def __init__(self):
        Super().__init__().
        print("C is accessed").
```

c = C()

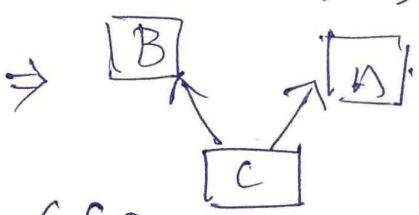
Output

B is accessed.
C is accessed

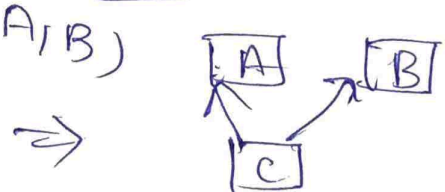
here it ~~bias~~ is not baised to 'class B, follows.

MRO, where class selection (Super class) for goes / strat from left to right,

here class C(B,A):



class C(A,B)



POLYMORPHISM

* Duck Typing
* Operator Overloading
* method Overloading
* method Overridding.

Duck Typing :-

Eg programme

```
class Pycharm:
    def excute (self):
        print ("compiling")
        print ("Running")
class Myeditor:
    def excute (self):
        print ("spell check")
        print ("convention check")

class Laptop:
    def code (self, ide):
        ide.excute()
```

case(i)

```
ide = PyCharm()
lap1 = Laptop()
```

Output

compiling
Running

case(ii)

```
ide = Myeditor
lap1 = Laptop()
```

output Spell check
convention check.

* here the data of method/x can be transferred to instance (variable) (class).
as we did like.

```
ide = pycharm()
or ide = MyEditor()
```

Operator Overloading

* we can not add two variables of same class, so to perform any operation (operation) we should define a method and through method we can do.

Eg
```
a = 5
b = 5
print(a+b) /output 10. but at back there is predefined method
```

we have to remember that 'int' is pre defined class.

Eg programme to understand:

```
Class Student:
    def __init__(self, m1, m2):
        self.m1 = m1
        self.m2 = m2
# creating method to add instances of same class.
# here __add__() → method to add.
```

```
def __add__(self, other):
    m1 = self.m1 + other.m1
    m2 = self.m2 + other.m2
# converted (Instances → variables).
    s3 = Student(m1, m2).
    return s3.

s1 = Student(58, 69)
s2 = Student(60, 45).
s3 = s1 + s2.
print(s3.m1)# ⇒ s3.m1 = s1.m1 + s2.m2.
```

output

118.

Method Overloading

→ This does not a feature of python

* The meaning of Method Overloading, 2 methods having same name and but having different parameters/arguments. in class is called method Overloading

Eg

```
class Student:
Student/Class:
    def avg(a, b):
        pass.
    def avg(a, b, c):
```

this is called method Overloading but not present in python.

→ Alternative method for Method Overloading way

Eg programme

```
class Student:
    def __init__(self, m1, m2):
        self.m1 = m1
        self.m2 = m2
    def sum(self, a=None, b=None, c=None):
        s = 0
        if a != None and b != None and
                                c != None:
            s = a+b+c
        else if a != None and b != None:
            s = a+b
        else:
            s = a
        return s
s1 = Student(59, 69)
print(s1.sum(2, 5))
print(s1.sum(5, 7, 8))
```

Output

7

20

Method Overriding

when we Inherit the methods class. then we, if both of Super class and Sub class contain methods of same name then, when we call the method from the object of Sub class, then it checks first in Sub class, if it is there it executes, if not Checks Super class.

Eg programme

```
Class A:
    def show(self):
        print("A")

Class B(A):
    def show(self):
        print(B)

S1 = B()
print(S1.show())
```

Output

B

another case :

```
Class A:
    def show(self):
        print("A")
Class B:
    def func(self):
        print("B")
S1 = B()
print(S1.show())
```

Output

A.

Iterator In python

Eg programme:

```
num = [7, 8, 9, 5]
```

Creating List into iterator (iterator is like converting i++ in C/C++).

Advantage of converting list to iterator is, to access elements in list no need of index.

```
it = iter(num) # List converted iterator named as it.
```

print(it) # gives output of object of iterator address.

print(it.next())

print(it.__next__()) # prints num[0]

__next__() is in built method/ function to print element in iterator

print(it.__next__()) # prints num[1].

Output

<list-iterator object at 0x7f37dff8b5e8>
↳ address

7 } num[0]

8 } num[1].

Another case **

Eg programme:- # if no. statement > no. of elements

num = [5, 6]

x = iter(num)

print(x.__next__())

print(x.__next__())

print(x.__next__())

output

5

6 # and gives also error "StopIteration." at line 5.

print(it) # gives output of object of iterator address.

print(it.next())

print(it.__next__()) # prints num[0]

__next__() is in built method/ function to print element in iterator

print(it.__next__()) # prints num[1].

Output

<list-iterator object at 0x7f37dff8b588>
↳ address

7 } num[0]

8 } num[1].

Another case **

Eg programme:- # if no. statement > no. of elements

num = [5, 6]

x = iter(num)

print(x.__next__())

print(x.__next__())

print(x.__next__())

output

5

6 # and gives also error "StopIteration." at line 5.

* Creating Own iterator

```
class Top:
    def __init__(self):
        self.num = 1
    def __iter__(self):
        val = self.num
        self.num += 1
        return self

values = Top()

    def __next__(self):
        val = self.num
        self.num += 1
        return val.

values = Ten()
print(values.__next__()) or (next(values))
print(values.__next__()) or (next(values))
```

Output

1
2

Genrators

* Genrators returns iterator in function s and methods.

Eg programme

~~def topten():~~

```
def fun-name (arguments):
    Statements
    Yeild variable
```

This function becomes genrator.

Eg problem to print Squares of first 4 numbers using genration.

```
def top4():
    ~~n=0~~ n=1
    While n<=4
        sq = n*n
        Yeild sq

values = top4() # all values will be
                  Shifted to values

for i in values:
    print(i)
```

Output

1 16
4
9

```
print(next(values))
print(values.__next__())
```

Output

Address of object

1 # Gives first Ouput.

Expection Handlings [Errorhandling]

- types of errors
  - * Syntax error
  - * logical error.
  - * Run time error.

Syntax error:- error would occur ~~at the~~ due to incorrect Syntax in code.

logical error:- The code doesnot have any Syntax error and runs but the desired output iis not given.

Eg when we A.p to "2+2"
but the ~~deepd~~ ouput 3;

These.

Run time Error: There would no be no Syntax error and logical error but error iis raised due to improper input

```
Eg  a = int(input())
    b = int(input())
    print(a/b).
```

output

2
0

Run time error
(undefined error).

Handling With Runtime error

Eg programme to handle with basic error

```
a = 5
b = 0
try:
    print(a/b)
except Exception as e:    # Exception is Keyword to address all types of error
    print("Hey, you cannot divide number by 0")
print("Bye").
```

↳ capital 'E'

Output: Hey, you cannot divide number by 0
Bye

division by zero

Output due to e.

Another runtime error handling programme to more Specify errors and output

```
a = 5
input b = int(input("b:"))
try:
    print(a/b)
except errorname1 as variable1:
    # statement
```

except error name2 as varbile2:
    // Statements.
except exception as varbile3:
    // Statements.

finally:
    // Statements that need to executed wheather runtime Error occurs or not.
Eg Closing data base Connection b/w client and database.

Multi threading

* Threads are used to perform/handle with different tasks/programmes. Eg while gamming we, other oppoients and system need perform different actions so each threads are assigned to dif perform different programmes at a time.

* To deal with threads we need to threading package.

Eg programme to run different task on 2 diff threads

from threading import * # calls / imports all packages

```
class Hello(Thread):
    def excute(self):
        for i in range(25):
            print("Hello").

class Hi(Thread):
    def excute(self):
        for i in range(500):
            print("Hi").

t1 = Hello()
t2 = Hi()

t1.Excute()
t2.Excute()
```

Output

```
Hello
Hello
 :25 times
Hi
Hi
Hello
 :
Hi
Hi
 :
Hello Hi  → This is one case
 :
```

usually while running this programme we running 'class Hello' on one thread and 'class Hi' on other thread, which are excuted parallelly

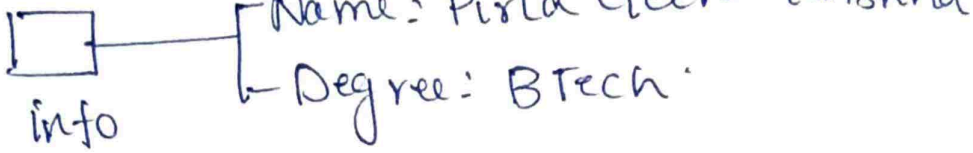
'cause is collision

* collision is a case where both threads execute the programme at a time.

* Actually while we coding for industrial purpouse, then the code of a class is so huge that, the execution of threads go alternatively, In that case 'collision' does not take place.

File Handling

* Let think a file 'info'



to read the file 'information'.

Programme-1

```
f = open('info','r')
    # here 'r' is arugment for
                reading.

print(f.reading()).
```

Output

```
Name: PirlaGeetaKrishna.
Degree: BTech.
```

programme -2 (to access line by line).

```
f = open('info','r')
print(f.readingline()) # first line of file
print(f.readingline()) # Second line of file.
```

Output

```
Name: Pirla Geeta krishna
Degree: BTech.
```

* Appending / adding data permanently to file.

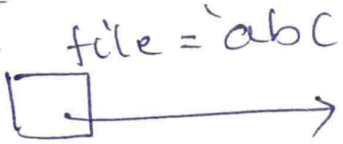
Ex programme

```
f1 = open('abc','a').
        # (filename, a → for appending).
```

if there is no file 'abc', then 'abc' named file will be created.

```
f1.writeline("laptop")
f1.writeline("\n")
f1.writeline("mobile")
```

Output



laptop
mobile.