Keyword / operator	uses
class	to create class
def	to define method
self	default Parameter of f^n of class
is	compair id of objects at which variable is pointing, ~~id~~ return (True or False)
+, -, *, /	Basic Arthmatic operations
// (flor division)	9//4 = 2, 11//4 = 2
% (reminder)	9%4 = 1, 11%4 = 3
** (Exponant)	7**2 = 49, 5**3 = 125
>, >=, <=, <, ==, !=	Basic comparison operator
OR (|)	5 | 2 = 7
AND (&)	5 & 2 = 0
XOR (^)	5 ^ 2 = 7
left shift (<<<)	5 <<< 1 = 10
right shift (>>>)	5 >>> 1 = 2
if, elif, else	Basic if-else conditions
for [loop]	to implement for loop.
while "	" while "
in	True if sub-string found in string
@ static (decorator)	to define static method in class
global	to specify global variable inside class methods
lambda	to define lambda function
non local	to define non local variable inside nested function
from	} used to import module in Python
import	
* (astrik)	to import all module
@ [function name]	to implement closure, after this statement ~~return~~ written in will go as input to it
@ property	to Access private data of class
del	to delete object / entity
try	put instruction (which can give error) in it
Except	to handle exception
as	Except as e, so synonym
Exception	generalize error of exception

method / f^n	uses
__init__(self)	constructor of class
type (x)	display datatype of x
print (x)	print value of x
id (x)	return id of object at which x is Pointi
x = input()	to take input from user, value stored in x
range ()	to generate no within given range range (stop), range (start, stop) range (start, stop, steps to take) : stop is excluded in range
* Strings methods	(20 methods)
len (string)	return length of string
ord (char)	gives ASCII value of charactor
chr (int)	convert ASCII value to charactor
upper ()	convert to uppercase
lower ()	convert to lower case
swapcase ()	swap case lower ⇄ upper
replace ("this", "with that")	replace character in string
startswith ()	True if string start with Parameter
endswith ()	True if string end with Parameter
count ()	count no of apperence of Parameter in stri
find(), index ()	give index of string if found
rfind(), rindex	give "−" but search from reverse
format ()	for string formatting
isdigit, isalpha	to check for digit, alphabate
isalnum	to check for alphanumeric
lstrip, rstrip, strip	to strip space from, left, right, Both
MRO ()	to show method resolution order.
lambda A : A+10	if function call, only once, then use it.
filter (f^n, list)	to filter element in list, on some condition
map (f^n, list)	to map element in list, on some f^n
getter ()	to print privat data
setter ()	to store private data
property (getter, setter)	to access getter, setter by same name
super ()	to call constructor of Parent class

* Python → object orientation, scripting, functional

* object orientation :

object : Anything & everything in universe is object
→ object are not isolated
→ they will interact with each other.

* Steps to be followed while using object to solve problem
(i) identify the object needed to solve the problem.
(ii) classify the object based on their type
(iii) define what the object has (property) & what it does (behaviour).

eg Ball → color, size, price
→ roll, Bounce
FAN → Price, color, no of Blade
→ rotate, spin

→ properties, behaviour

⇒ properties are known as variable
Behaviour are known as method / function

Class FAN :
→ color
→ Brand
→ price
} Variable

→ def on() :
→ ~~~
→ def off() :
→ ~~~
} method

* cohesion

```
Class Demo :
    a=10
    b=20

    def x():
        — —
    def y():
```

this is Incohesive
or
Not cohesive

Because name of variable & method doesn't make any sense

* Rules for defining class :-

(i) Name of class must be written in capital letter
(ii) variable name ~~must~~ must be Noun
(iii) method / f^n name ~~must~~ must be verbs.
(iv) other then class name everything is in lowercase

```
Class FAN :
    color = " Red "
    price = 1000
    Brand = " usha "

    def on ( ) :
        print ( " — " )
    def rotate ( ) :
        print ( " — " )
```

FAN ( ) ← object creation

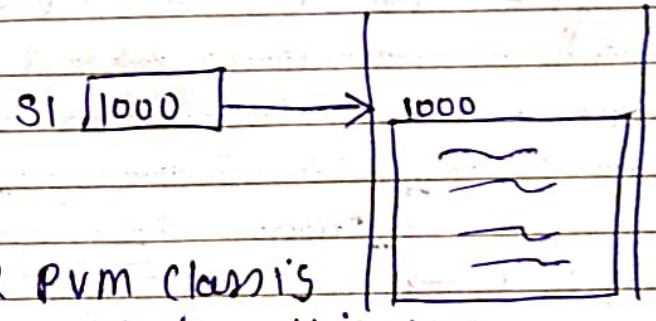
=> without object no memory is allocated to class
=> when we create object , class get memory.

9/12/21

* what happen when we create object

```
class Student :
    name = "mj"
    age = 24

    def study ():
        print (" studying")
    def play ():
        print (" Playing")
S1 = Student()
```

So In RAM



note :

Student () ⇒ look PVM class is ready, create a new object of this class

So, PVM give memory to object & return address of object, which we store in S1

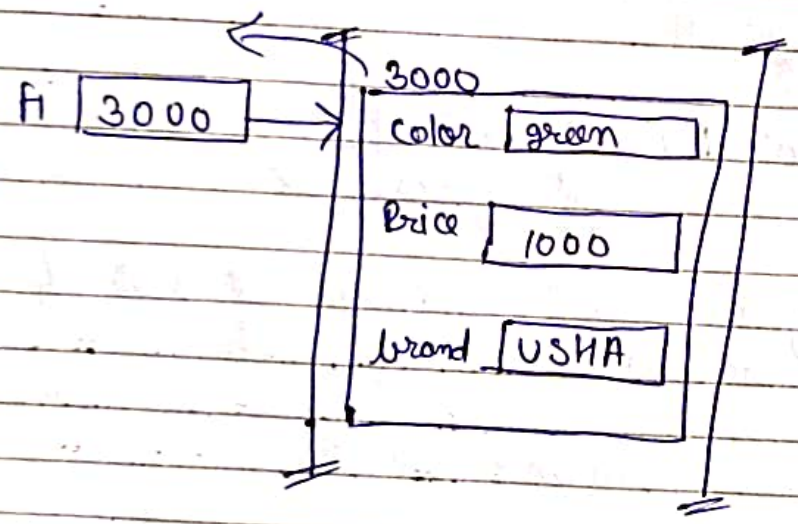
* Steps involved in the creation of object.

(i) PVM will allocate memory on the RAM.
(ii) PVM will look for __init__() inside the class
(iii) the address of object created will be returned

So our updated class

```
class FAN:
    def __init__(self):
        self.color = "green"
        self.price = 1000
        self.brand = "usha"

    def on(self):
        print("fan on")

f1 = FAN()          # returning address.
```

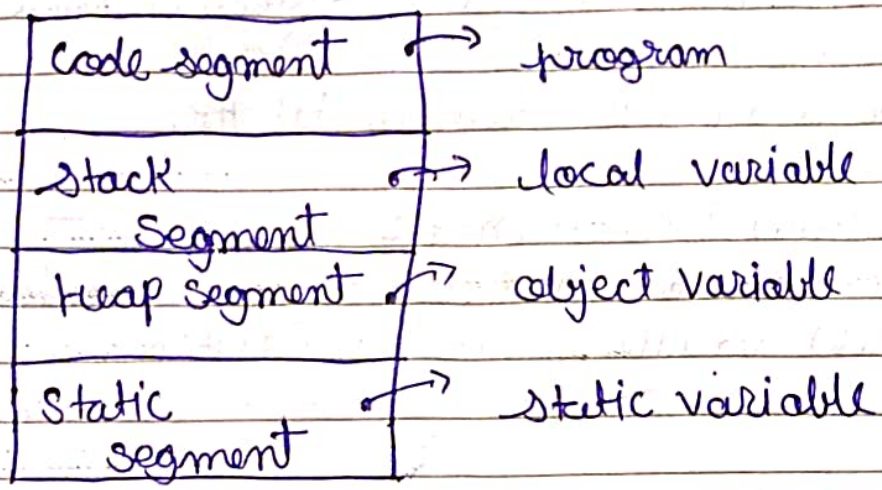
In RAM



(i) memory is given to object & it has some address (3000)
(ii) PVM will look for __init__() method
(iii) In init() method, 1st line
self.color → 3000.color
so PVM will go to address 3000 & look for color.
if object doesn't have color variable then PVM will create new

(iv) PVM will return address

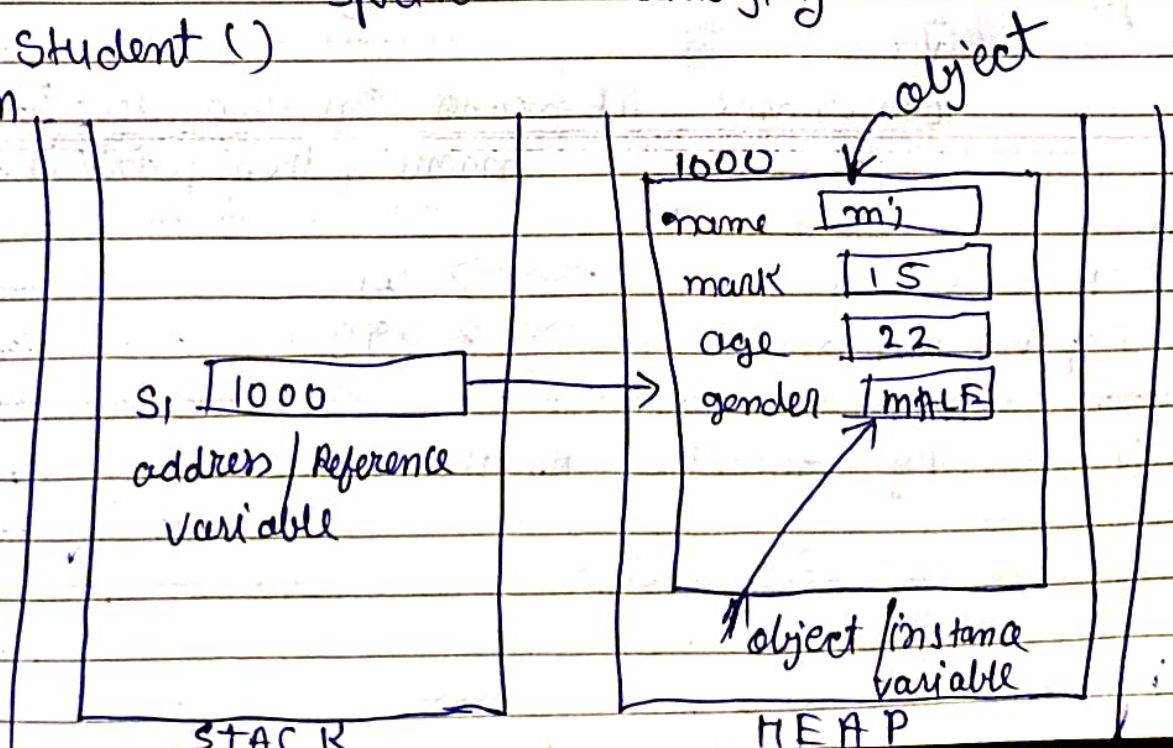
* Segments of RAM :



* constructor : In Python we have constructor __init__(self) method.

ex
```
class student
    def __init__(self):
        self.name = "mj"
        self.mark = 15
        self.age = 22
        self.gender = "MALE"

    def study(self):
        print("Studying")
s1 = Student()
```

In RAM



when we create object, firstly constructor is called

⇒ variable of constructor is called object/instance variable.

⇒ object / instance variable stored in HEAP.

⇒ address of object is returned & stored in variable, ie. called as address / reference variable

⇒ Reference / address variable is stored in STACK

* printing class values :

```
class student :
    def __init__ (self) :
        self.name = "mj"
        self.age = 21
        self.marks = 99
    def study ( self ) :
        print ( " studying " )

S1 = student ()

print ( S1.name )  ~~~> mj
```

1000.name it means in 1000, look for name, then print it.

simi.

```
print (S1.age)     ~~~> 21
print (S1.marks)   ~~~> 99
```

if we copy object to another then

$S_2 = S_1$

$S_3 = S_1$

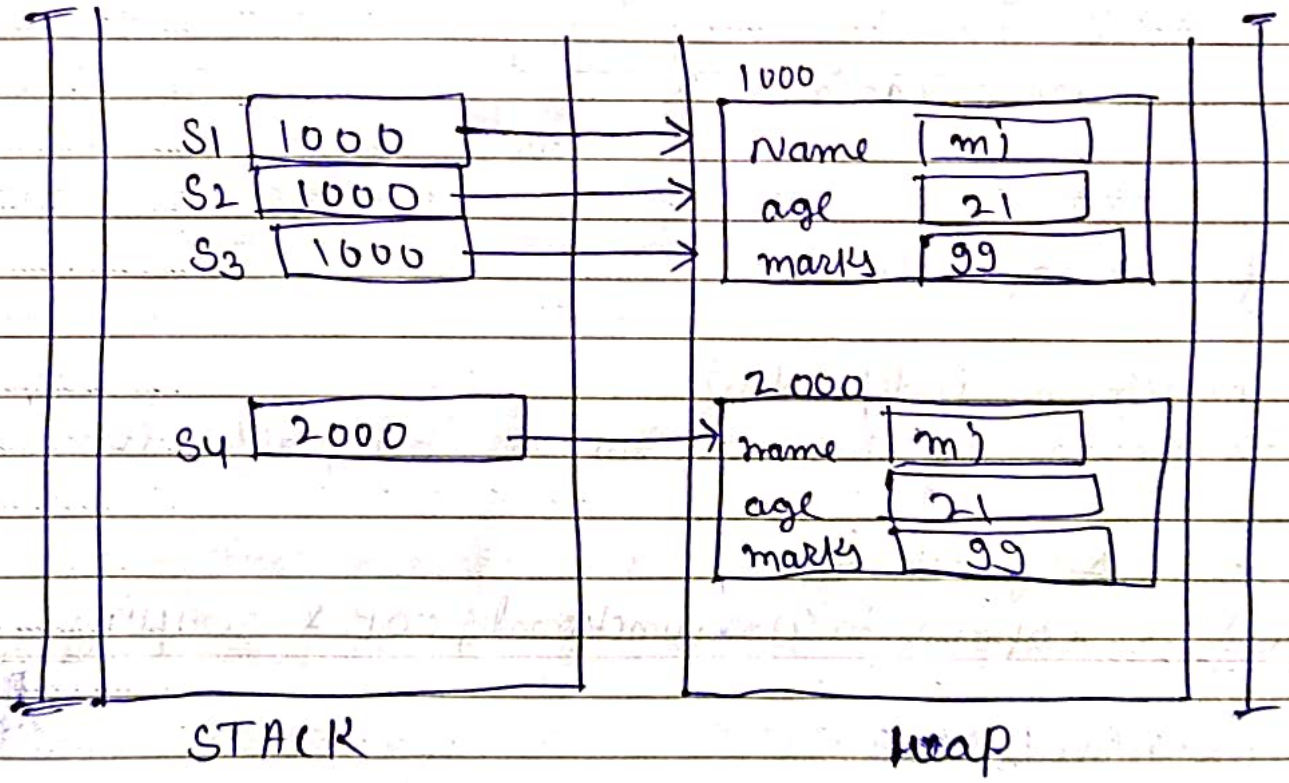
print (S2. name) ⇝ mj
print (S2. age) ⇝ 21

print (S3. name) ⇝ mj
print (S3. age) ⇝ 21

S4 = Student # new object

S1. age = 25 # value updated
print (S1. age)
print (S2. age) } ⇝ 25
print (S3. age)
print (S4. aage) ⇝ 21

In RAM



S1 = Student ()
S2 = S1
S3 = S1
} one object is created
3 ~~single~~ reference variable

S1 = Student ()
S2 = Student ()
} 2 object is created
2 reference variable

10/12/21

* features of Python

(i) Simple & Easy to code

Pythodn code to add two no

```
a,b = 10,20
c = a+b
print (" sum is ", c)
```

(ii) High level language

any programing language which uses
A-Z, a-z, 0-9 & all special symbol
is High level language

$$\text{ease of coding} \propto \frac{1}{\text{level of language}}$$

(iii) free & open source

(iv) plateform independent
→ we can use same bit code for all plateform (OS).

(v) OOP & functional
it is capable to do functional, OOP & scripting

(vi) Extensive library
it has huge library & support.

(vii) Dynamically typed

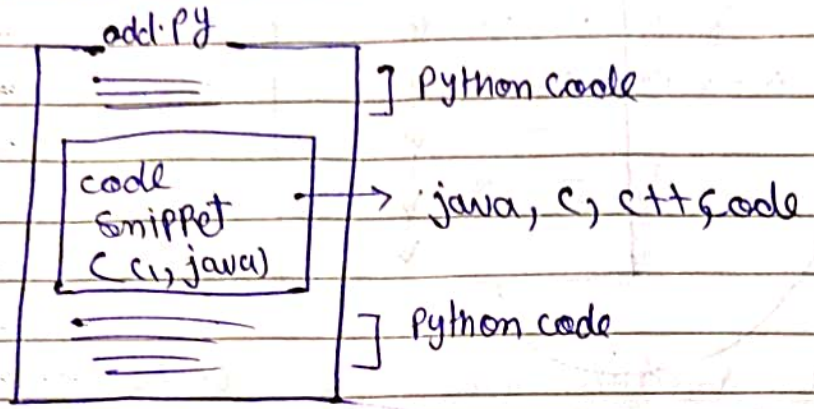
```
a = 10
b = 20
c = a+b
```

it check itself for data type.

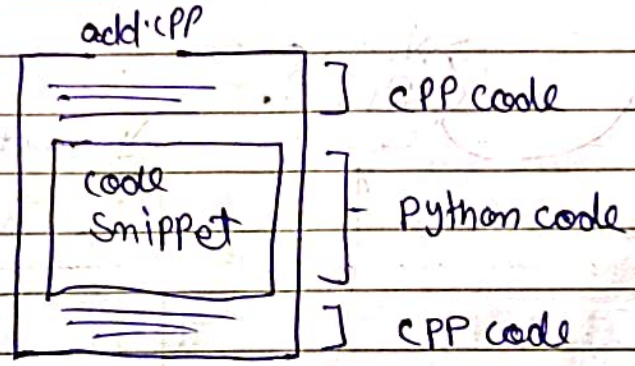
⇒ PVM during execution will assign a data type to variable Based on the value, which is stored in variable

these type of language is called dynamically typed language

(viii & ix) Extensible



Embeded



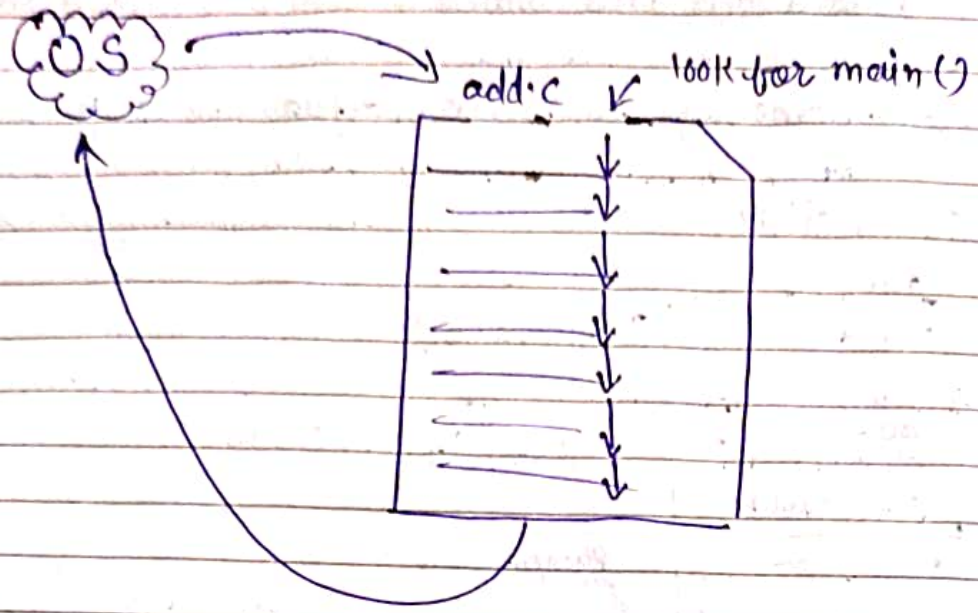
(X) Interpreted & compiled

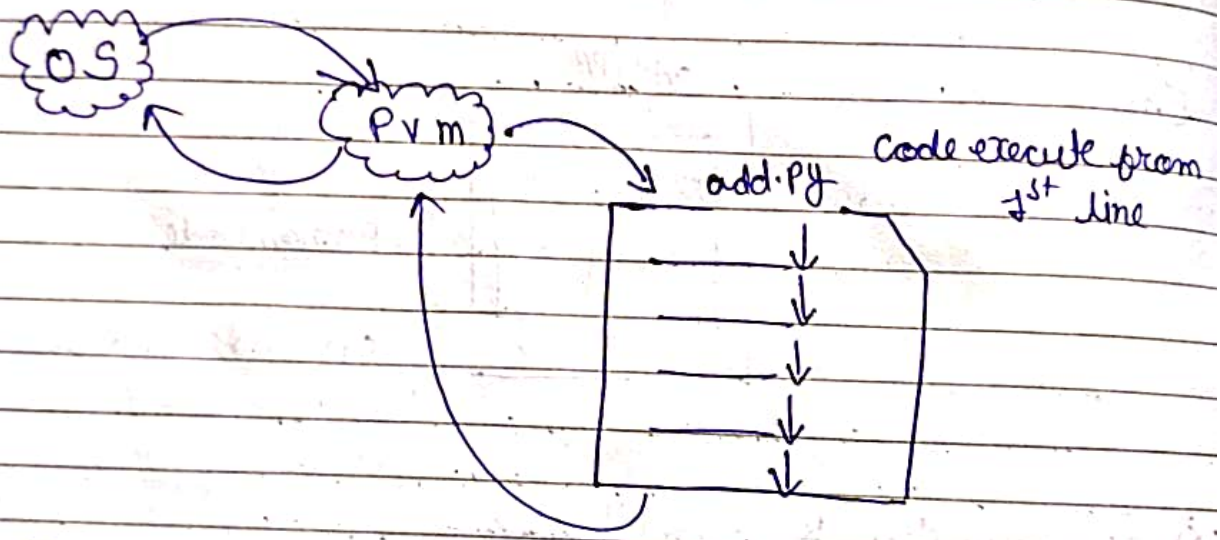
In Python code firstly compiled & generate Byte code then it is given to interpreter

that' why

~~it is~~ we have compiler & interpreter in Python.

* controll flow

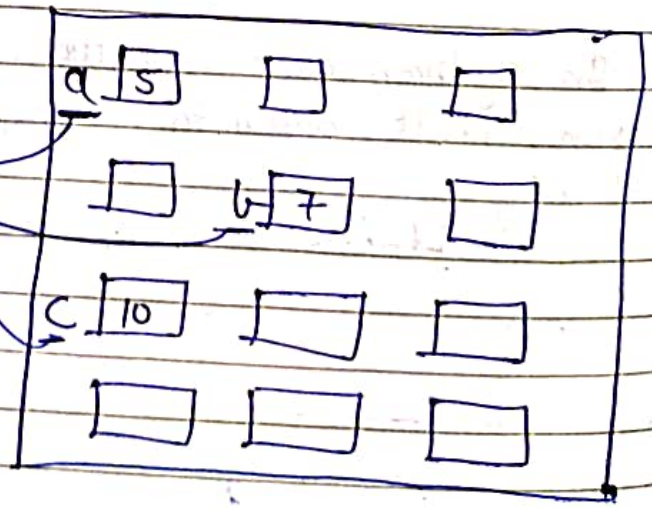




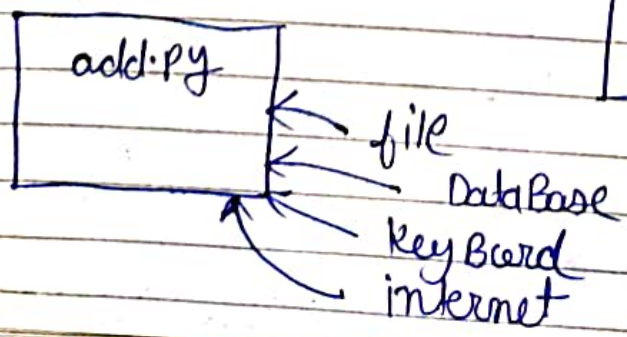
(*)

a = 5
b = 7
c = 10

so In RAM



(*) Input/Data source



13/12/21

IDLE => integrated development & learning Environment

By default when we open IDLE, IDLE open in interactive mode

* interactive mode :

```
>>> 6+4
    10
>>> 5*4
    20
>>> print (" welcome ")
    welcome
```

so, In interactive mode, we write one statement & we get output for one statement

* Scripting mode : when we open new file, new file open in scripting mode (By default).

-> we can write multiple line & get output for all,
-> mainly we work in scripting mode

Code 1 name : add_two_no

```
a = 10
b = 20
c = a+b
print (" sum is ", c)
```

Code 2 name : class_object

```
class student :
    def __init__(self) :
        self.name = "mi"
        self.age = 22
        self.marks = 99
        self.location = "Bhilwara"
```

object variable

address variable

```
def study(self):
    print("studying")
```

```
S1 = student
print(S1)
```
→ address of object

```
print(S1.name)
print(S1.age)
print(S1.marks)
```
→ mj
→ 22
→ 99

```
S1.study()
```
method is called, & code of method is excerted

↳ studying

```
S2 = S1
```
copying object
so No new object created
address/reference variable created

```
print(S1)
print(S2)
```
→ same address

```
S3 = S1
S4 = student()
```

```
print(S1)
print(S2)
print(S3)
```
→ same address

```
print(S4)
```
→ different address

```
S1.name = "manoj"
```
→ # name changed

```
print(S1.name)
print(S2.name)
print(S3.name)
```
→ manoj

```
print(S4.name)
```
→ mj

so, if we change any value, using any one address variable then the change will Happend to other address variable also, which are pointing to same addre

* <u>type method</u> :-

type( ) : this will return data type of variable

ex

Code

name: data-type

```
a=10
b=5.5
c="mj"

print (type(a))  ~~> <class 'int'>
print (type(b))  ~~> <class 'float'>
print (type(c))  ~~> <class 'str'>
```

<u>note</u>: In Python declaration without Assignment is not Allowed

ie

int c;	a	✗] → not allowe
int a=10;	b=10	✓
int b=20;	c=20	✓
c=a+b;	a=b+c	✓
printf("c);	print (a)	✓

⇒ so in Python only declaration is not allow, we must give value to variable, we we declare it.

By program data-type we know

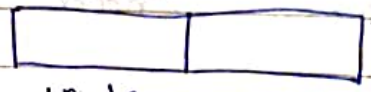
-> all datatype in Python are classes
-> variable are pointing to object of these classes.

* Size of variable :

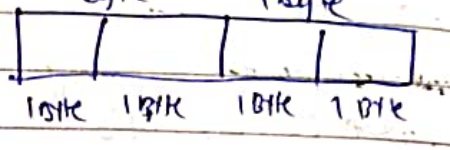
In language other than Python, a previously determined amount of memory space is allocated to data type

ex

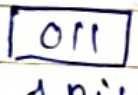
int a ⇒ 2 Byte



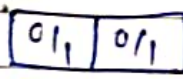
float B ⇒ 4 Byte



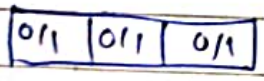
so How we calculate range



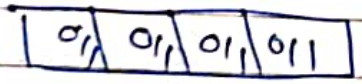
range is 0 - 1 (2) $0 - 2^1 - 1$



range is 0 - 3 (4) $0 - 2^2 - 1$



range is 0 - 7 (8) $0 - 2^3 - 1$



range is 0 - 15 (16) $0 - 2^4 - 1$

so for n Bit

range is $0 - 2^n - 1$

* n is no of bit

so for

8 Bit range is 0 - 255
16 Bit range is 0 - 65535

But we have 2 issues in this kind of allocation

(i) if we need to store data beyond range
then (a) we need to ↑ memory space
(b) we need to change data type

(ii) if we have less data to store
then (a) we have many memory waste

But In Python :-

& for particular data type, amount of memory allocation is not fixed, rather it depending upon size of data, ie How much memory data needed

so

Python has optimize & dynamic memory allocation

Code

```
name : variable - size
a = 1000000 0000 00000000 0000
print (a)
print ( type (a))
```

* memory mapping :

a = 10
b = 5.5
c = "mj"

so memory mapping

a - [10]
b [5.5]
c [mj]

print (type(a)) → class int
print (type(b)) → class float
print (type(c)) → class str

so, variable, internally in Python are managed as object.

→ object are Pointing to data

so

correct memory mapping is

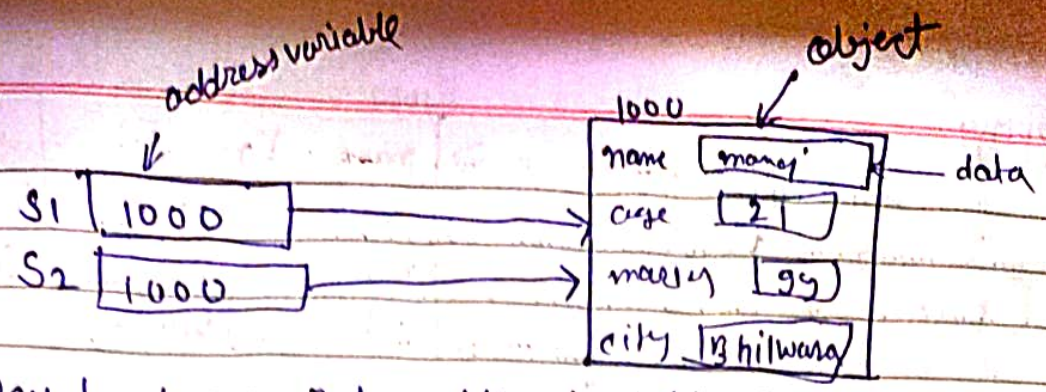
a → [10]
b → [5.5]
c → [mj]

[code] name : id of object

```
Class Student :
    def __init__ (self) :
        self. name = "manoj"
        Self. age = 21
        Self. marks = 99
        Self. city = "Bhilwara"

S1 = Student()
S2 = S1
# 1 way to prove Both ~~object~~ address variable Pointing to same object

print (S1)
print (S2)
```

→ 1000
→ 1000] same

mapping



2 way to prove Both address Variable Pointing to Same object

print(S1.name) → manoj
print (S2. name) → manoj
print (S1. age) → 21
print (S2. age) → 21

3 way to prove Both address variable Pointing to Same object

→ Each & Every object have address & UID (unique int I
→ UID will not change under any circumstance
→ address may change during execution by various reason.

→ implement of Heap
→ garbage collection
→ allocation & deallocation of memory

So Same UID can't be given to two object.

print (id (S1)) →] same id
print (id (S2)) →

4 way to prove Both address variable Pointing to Same object.

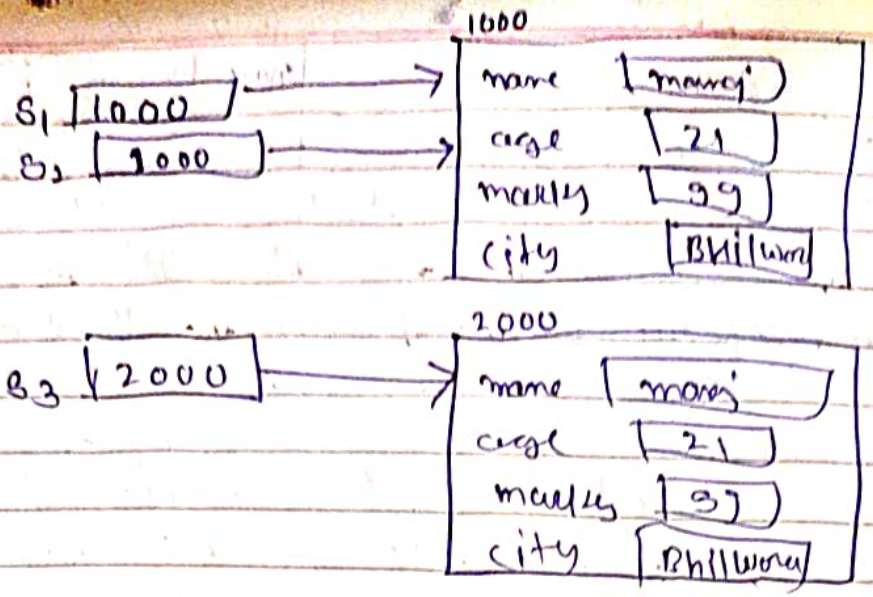
print (S1 is S2) # is operator, it compair id of object, which address variable is Pointing

if = = true
else ! = false

Now, we create one more object

S3 = Student ( )
print (S3) → 2000

mapping



#1 way to prove S1, S2 & S3 pointing to different object

print (S1)] — same
print (S2)
print (S3) → diff

#2

print (id (S1))] — same
print (id(S2))
print (id (S3)) → diff

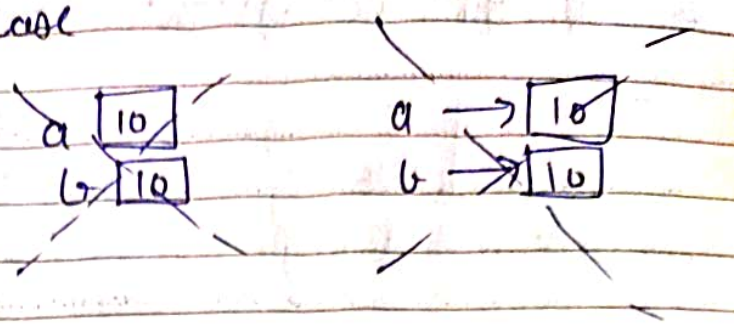
#3

print (S1 is S2) → true
print (S1 is S3) → false
print (S2 is S3 → false

[code] data variable object

let's look at one more case

a = 10
b = a



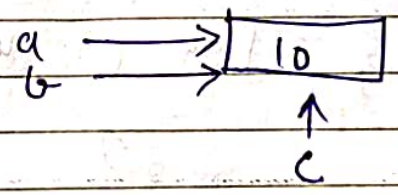
print (a) → 10
print (b) → 10
print (id(a))
print (id(b)) } Same id it means both Pointing to same object

print (a is b) → true it confirm that Both Pointing to same object

Now,
c = b
print (id(c)) → same id

print (a is c)
print (b is c) } true so our previous memory mapping is wrong

memory mapping

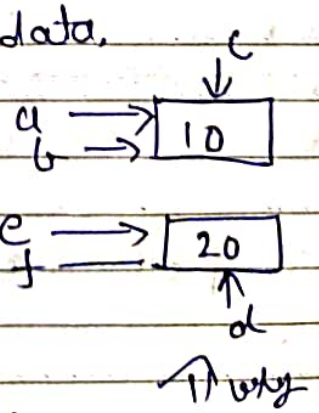


note :
so { when assignment is done from one variable to another, copy of new data is NOT genrated & Both variable will Point to same object containing data.

a = 10
b = 10
c = 10
d = 20
e = 20
f = 20

print (id(a))
print (id(b))
print (id(c)) } same

print (id(d))
print (id(e))
print (id(f)) } same



Because all have same id

15/12/21

* Arthmetic operator

2+3 = 5 , 17 + 7 = 24
10-5 = 5 , 7 - 3 = 4
6*6 = 36 , 7*5 = 35
10/2 = 5 , 9/4 = 2.25 , 11/4 = 2.75

// floor devision

9//4 = 2 , 11//4 = 2

```
4 ) 9 ( 2
    8
    1

4 ) 11 ( 2
    8
    3
```

% Reminder

9% 4 = 1, 999%3 = 0 , 11 %4 = 3

```
4 ) 9 ( 2
    8
    1

3 ) 999 ( 333
    999
    0

4 ) 11 ( 2
    8
    3
```

** Exponant operator

7**2 ⇒ 7^2 = 49
11**2 ⇒ 11^2 = 121
5**3 ⇒ 5^3 = 125

* comparison operator

> < ==
>= <= !=

1	1	1	1	1	1	1	1
128	64	32	16	8	4	2	1

* Bitwise operator

=> OR (|)

a = 5
b = 2
c = a|b

a	0	0	0	0	0	1	0	1
b	0	0	0	0	0	0	1	0
c = 7 =	0	0	0	0	0	1	1	1

=> AND (&)

a = 5
b = 2
c = a&b

a	0	0	0	0	0	1	0	1
b	0	0	0	0	0	0	1	0
c 0 =	0	0	0	0	0	0	0	0

=> XOR (^)

0	0	0
0	1	1
1	0	1
1	1	0

a = 5
b = 2
c = a^b

a	0	0	0	0	0	1	0	1
b	0	0	0	0	0	0	1	0
c	0	0	0	0	0	1	1	1

=> 7

=> left shift (<<<)

a = 5
c = a << 1

	0	0	0	0	0	1	0	1
10 =	0	0	0	0	1	0	1	0
					8		2	

=> Right shift (>>>)

a = 5
c = a >> 1

	0	0	0	0	0	1	0	1
=	0	0	0	0	0	0	1	0
	2							

* Indentation in Python

Indentation : no of space that you leave before you write code.

→ In Python we do not have { }

→ for start we have : But for end we have nothing.

#

#

```
void main()
{
  int a,b,c;
  for ( )
  {
     if ( )
       {
       }
     else {
       }
  }
}
```

Block

Block

Block

Block

But in Python

```
a=10
for ( ) :
    if ( ) :
    else ( ) :
print (a)
```

Block

Block

Block

* for loop :-

Syntax for variablename in itemlist

=> firstly looping variable assigned some memory
=> during first iteration variable give value

#1
```
for i in 1, 2, 3, 4, 5 :
    print (i)
```
→ 1 2 3 4 5

#2
```
for x in 1.1, 2.3, 4.1, 5.7, 99.8 :
    print (x)
```
→ 1.1 2.3 4.1 5.7 99.8

#3
```
for i in "ramu", "rani", "reta", "rohan" :
    print (i)
```
→ ramu rani reta rohan

#4
```
for i in "ramu", 1, 7.77, "rita" :
    print (i)
```
→ ramu 1 7.77 rita

=> In Python loop can work on diffirent data set on same time

#1 working
```
for i in 1,2,3,4,5
    print 1
    print 2
    print 3
    print 4
    print 5
```
i → ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5

* range ( ) : it is used to genrate no within a given range, it can be used in 3 ways
→ it can't genrate string & non integer

#1 range (stop)

ex range (5) : 0, 1, 2, 3, 4
it will genrate 5 no.

#2 range (Start, Stop) :
ex

range (20, 25) ⇒ 20, 21, 22, 23, 24

start included, stop excluded

{ if stop is n then range genrated is n-1 }

#3 range (Start, Stop, steps to take) ← increment

range (0, 10, 2) → 0, 2, 4, 6, 8

range (1, 10, 2) → 1, 3, 5, 7, 9

(*) #

range ("ramu", "rani") ✗

range (1.4, 23.7) ✗
we get error in Both cases

range only works for integer no., it can't genrate string & non integer

17/12/24

for printing 1 to 1000

```
for i in range (1, 1001):
    print (i)
```

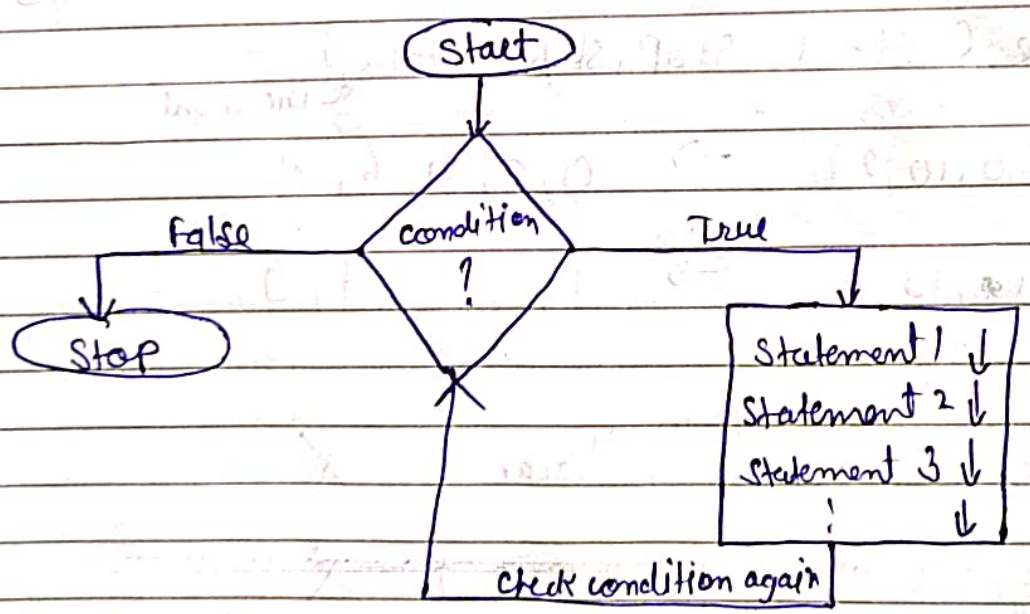
Because in range start is included but end/stop is excluded
so range (1, n)
will generate 1, 2, 3, --- n-1.

for generating no 1, 2, 3, 4, 5

```
for i in range (1, 6)
    print (i)
```

* while loop :

⇒ flow chart of while loop



⇒ Psudo code

```
i = 0                          # initilization
as long as  i < 10             # condition
            print ("Hii")
while
            i = i+1            # increment
```

syantax of while

```
i = 0                    # inilization
while (i< 10) :          # condition
    print ("Hii")         # statements
    i = i+1              # increment / decrement
```

loop tracing

1 iteration	i $\boxed{0}$	✓ $\boxed{0<10}$	→ Hii	i $\boxed{1}$
2 iteration	i $\boxed{1}$	✓ $\boxed{1<10}$	→ Hii	i $\boxed{2}$
3 iteration	i $\boxed{2}$	✓ $\boxed{2<10}$	→ Hii	i $\boxed{3}$
⋮				
11 iteration	i $\boxed{10}$	✗ $\boxed{10<10}$	→ exit	

note : we must increment value because if we don't increment value the it goes in infinite loop

note in Python pre / post increment / decrement are not valid
ie i++ / ++i --i / i-- ✗
↑ invalid

But shorthand are valid

```
i += 1   ,   i = i+1
i -= 1   ,   i = i-1
i *= 3   ,   i = i*3
i /= 2   ,   i = i/2
```

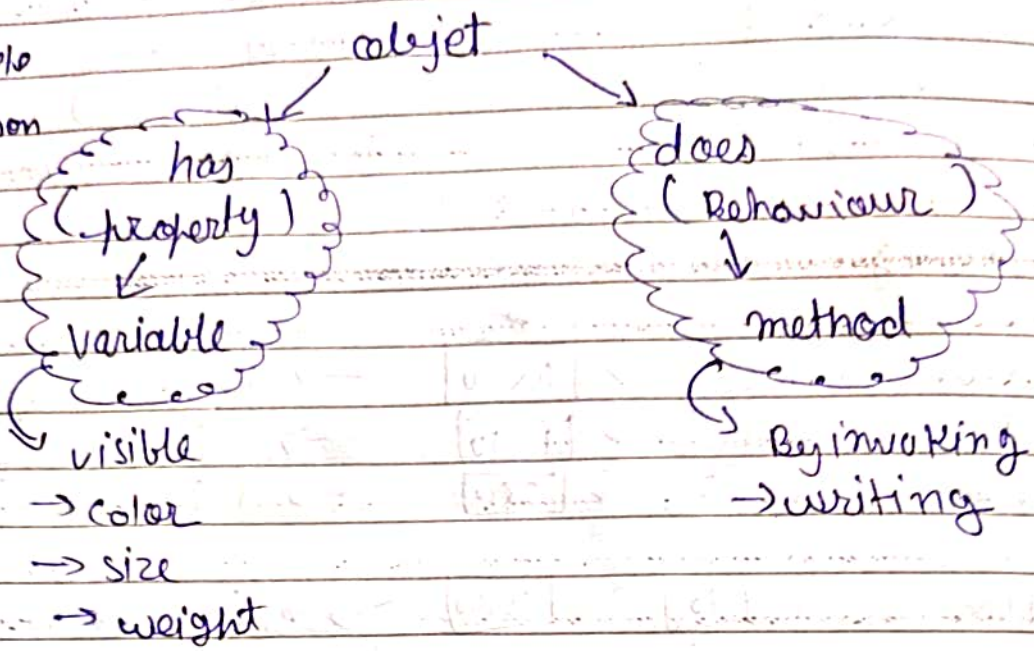
valid ✓

* method / f^n :

In class we have object

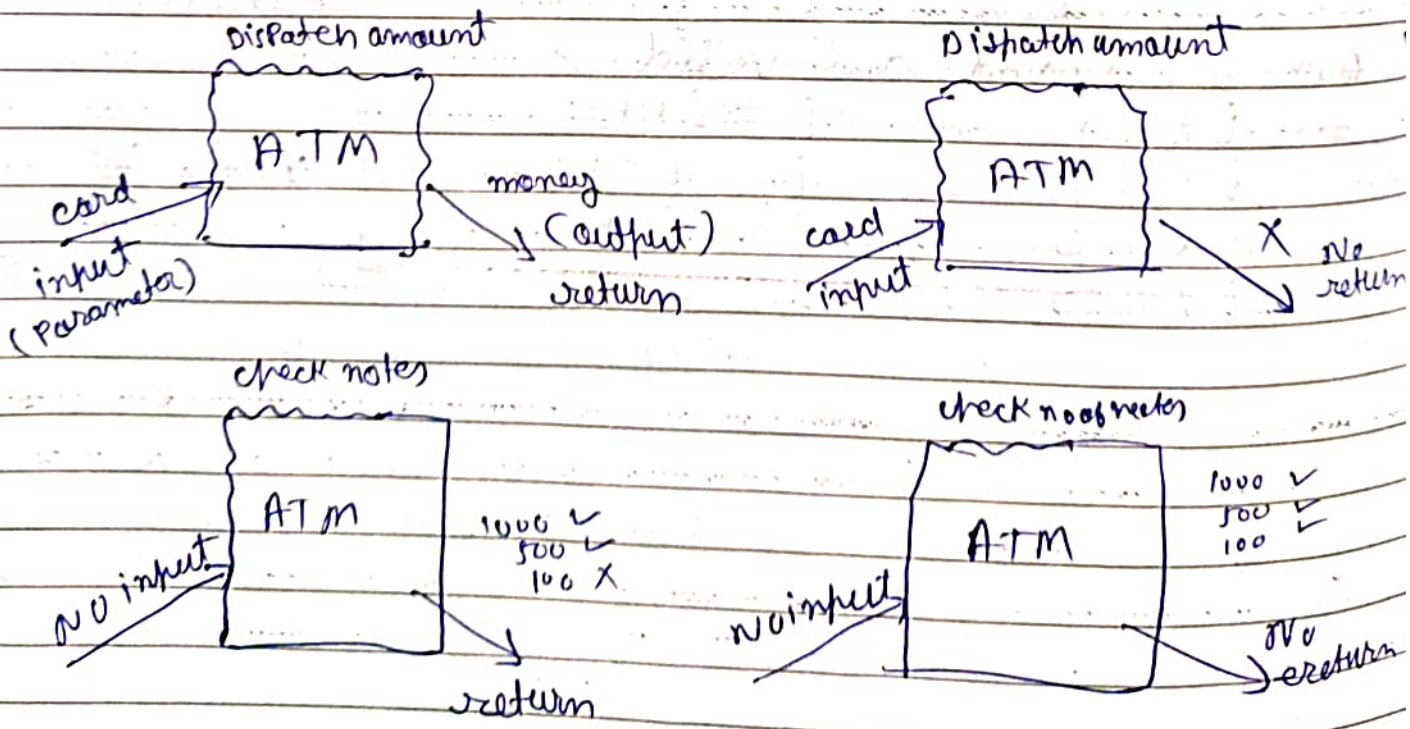
Penexample

slap a person



* type of method

(i) with Parameter , with return value
(ii) with Parameter, without return value
(iii) without Parameter, with return value
(iv) without Parameter, without return value



* Syantax of method

keyword

```
def method name ( Parameter ) :
    ____________
    ____________
    ____________
```

indentation

(i) with Parameter with return value

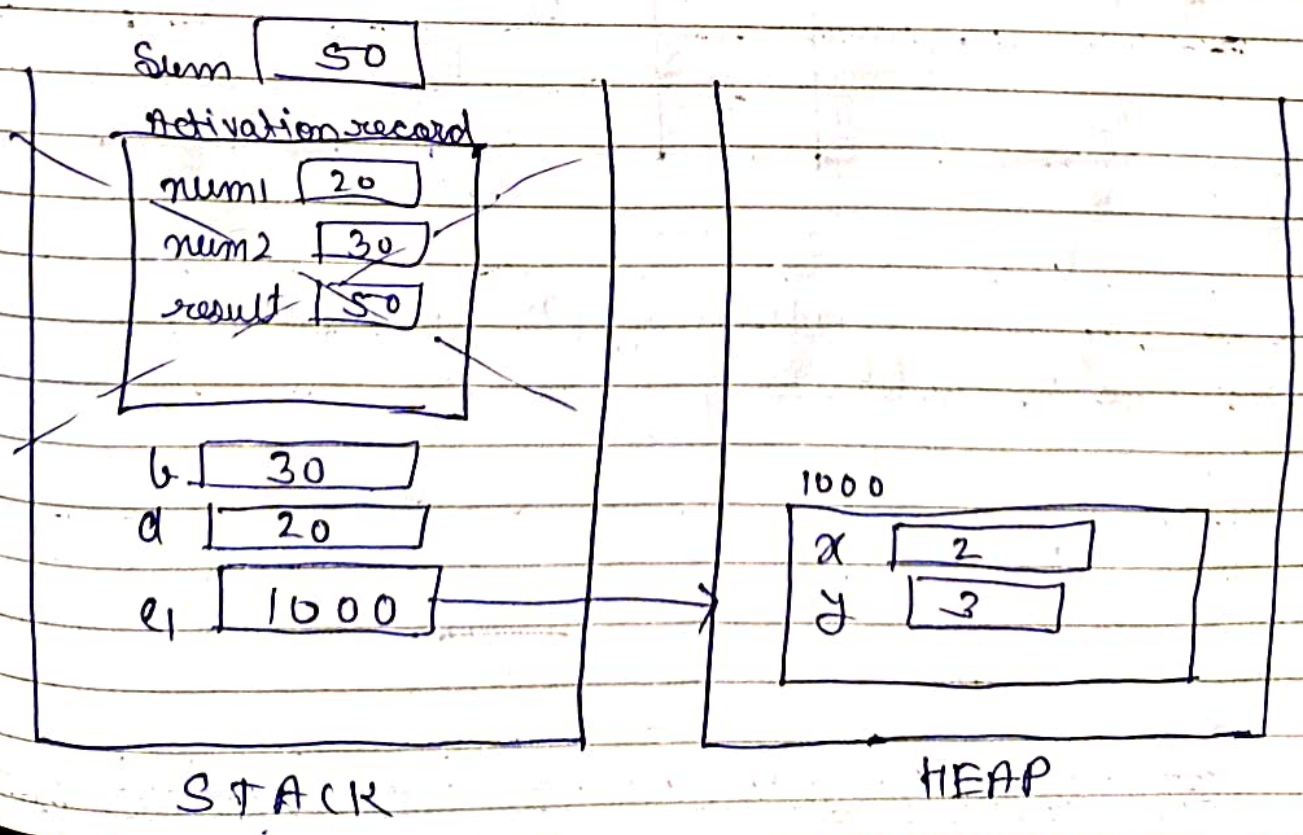
```
Class calculator :
    def __init__(self) :
        self.x = 2
        self.y = 3

    def add ( self, num1, num2 ) :
        result = num1 + num2
        return result

c1 = Calculator()
a = 20
b = 30
sum = c1.add ( a, b )
print ( "sum is : ", sum )
```

with parameter 20, 30

created activation Record

activation record deleted

with return value 50



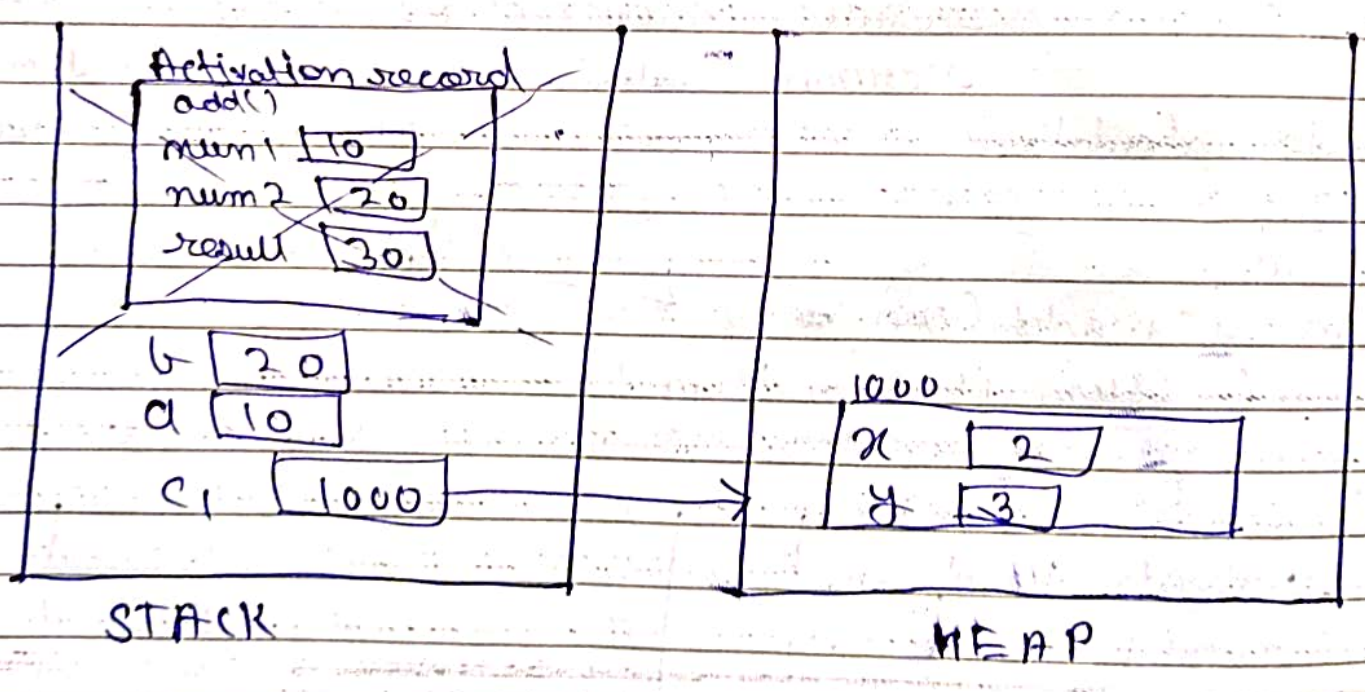
(ii) with Parameter, without return value

```
Class calculator :
    def __init__(self):
        self.x = 2
        self.y = 3
    def add ( self, num1, num2 ):
        result = num1 + num2
        print ("sum is : ", result )

c1 = calculator
a = 10
b = 20
c1.add (a, b)
```

with Parameter 10, 20

No return value

Activation record created → Activation record deleted



note : after execution of f^n Activation record is deleted

(iii) without Parameter, with return value

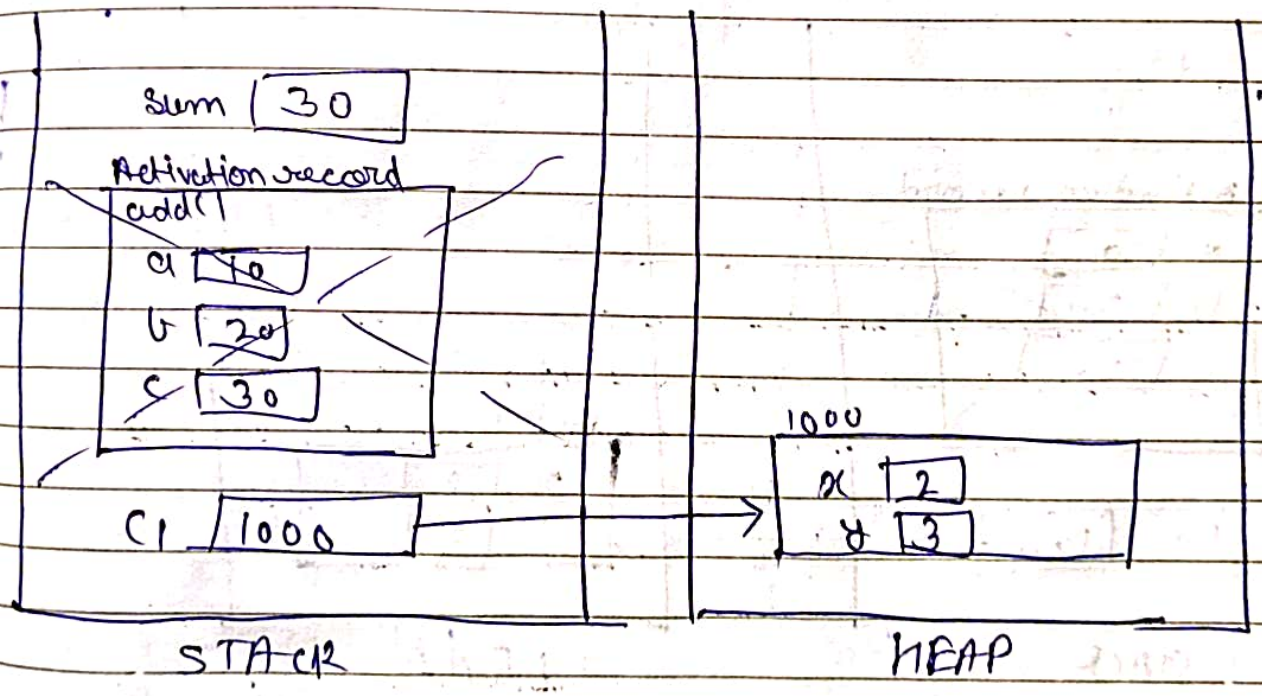
```
class calculator :
    def __init__(self):
        self.x = 2
        self.y = 3
    def add(self):
        a = 10
        b = 20
        c = a+b
        return c
c1 = calculator()
sum = c1.add()
print("sum is : ", sum)
```

no parameter

Activation record created

Activation record deleted

return value 30



when method return a value, then value must be catched by or store by another variable for further use.

(iv) without Parameter without return value

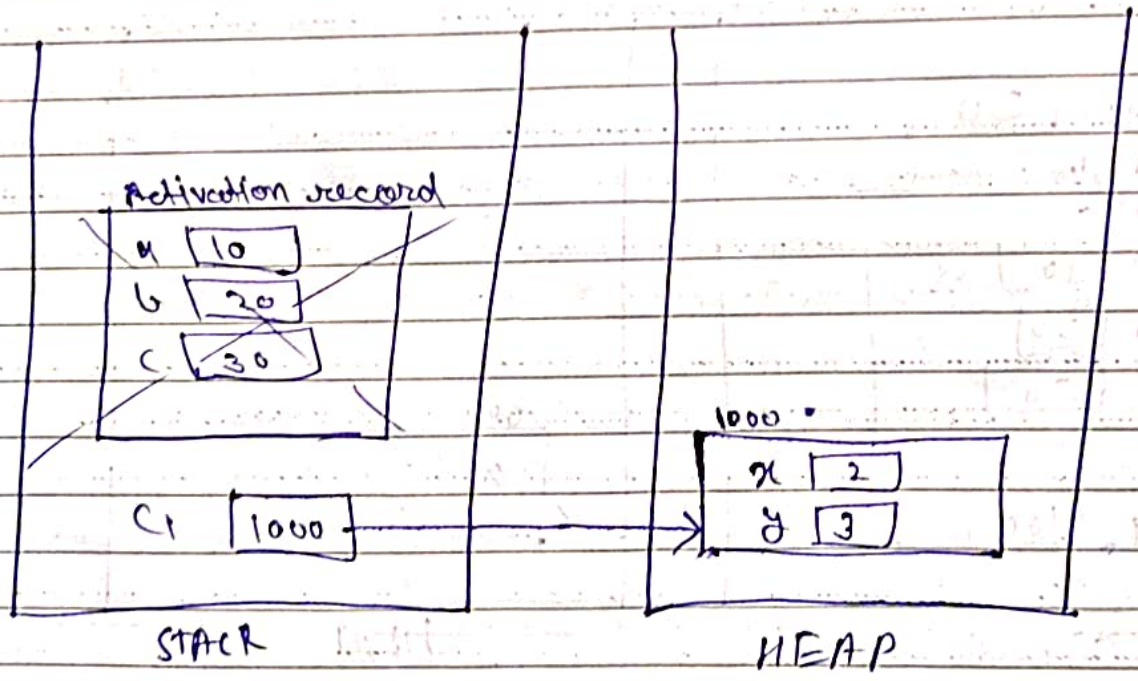
```
class calculator:
    def __init__(self):
        self.x = 2
        self.y = 3

    def add(self):
        a = 10
        b = 20
        c = a+b
        print("sum is ", c)

c1 = calculator
c1.add()
```

No Parameter

No return value

Created Activation record

Activation record deleted



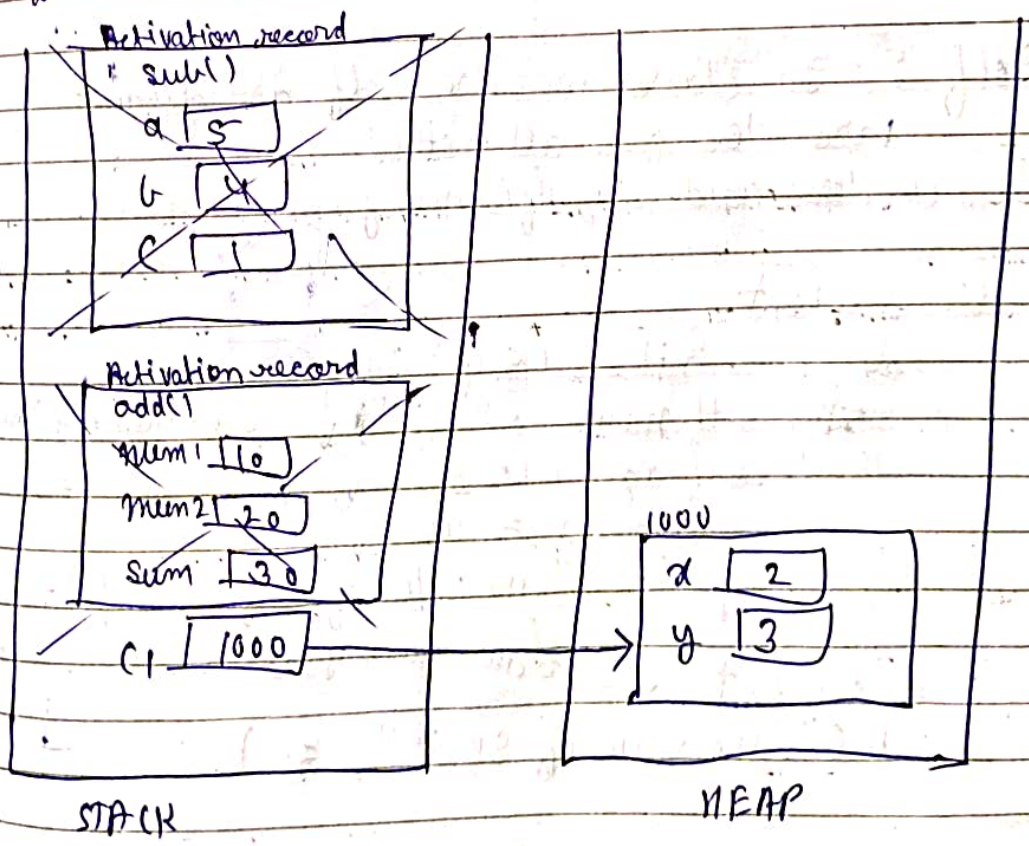
when we come out from class object also deallocated

```
* class calculator :
        def __init__ self
object variable { self.x = 2
                  self.y = 3

        def add (Self):
Activation record created
        local variable [ num1 = 10
                         num2 = 20
                         sum = num1 + num2
                         print (sum)
Activation record deleted
Activation record created
        def sub ( self):
        local variable [ a = 5
                         b = 4
                         c = a - b
                         print (c)
Activation record deleted
```

constructor calling

address of obj

```
C1 = calculator
C1.add
C1.sub
```



* difference between local & object variable

instance/object variable	local variable
(i) variable of constructor __init__(self)	variable of other method
(ii) memory allocated when object is created	memory allocated when method is invoke / called
(iii) memory deallocated when object become garbage	memory deallocated when control reach to last statement of method, ie after execution of method
(iv) these stored in HEAP	these stored in STACK

21/12/21

* Self : In classes we use self as default Parameter for all method,

let's understand significance of self.

```
class student:
    def __init__(self):
        self.name = "mj"      ↓ 1000.name = "mj"
        self.age = 23         ↓ 1000.age = 23
        self.marks = 40       ↓
        print(" the value of self ", self)
S1 = student  ← return address of obj
print(" the value of S1 is ", S1)
```

same

* 1000.name = manoj , so PVM will go to 1000 & 100K for name, if name not present then create name, Put value "manoj".

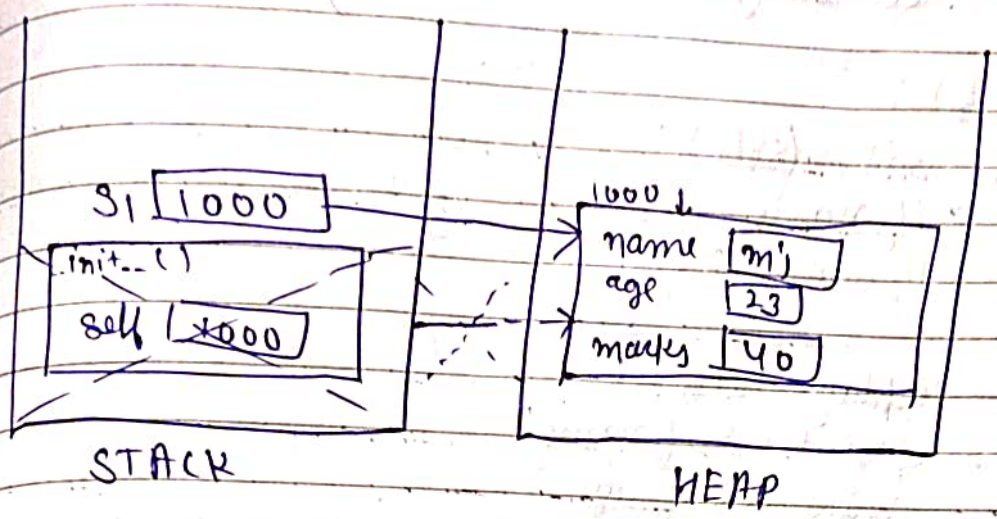
In memory

PVM —invoke→ __init__(self)

call → __init__(1000)

so self [1000]

Now PVM will execute __init__(1000) method (self)



-) when we create object PVM will look for __init__(self)

-) PVM will call constructor as __init__(1000) (address of obj)

so Now self have ~~ob~~ address of newly created object
now statements of __init__() method will execute

-) self.name => 1000.name so ~~PVM~~ controll will go to 1000 & look for name if name not present then PVM will create name & put value in it.
similarly with
self.age => 1000.age ; & self.marks => 1000.marks

at last __init__() return newly created object address which is stored in S1

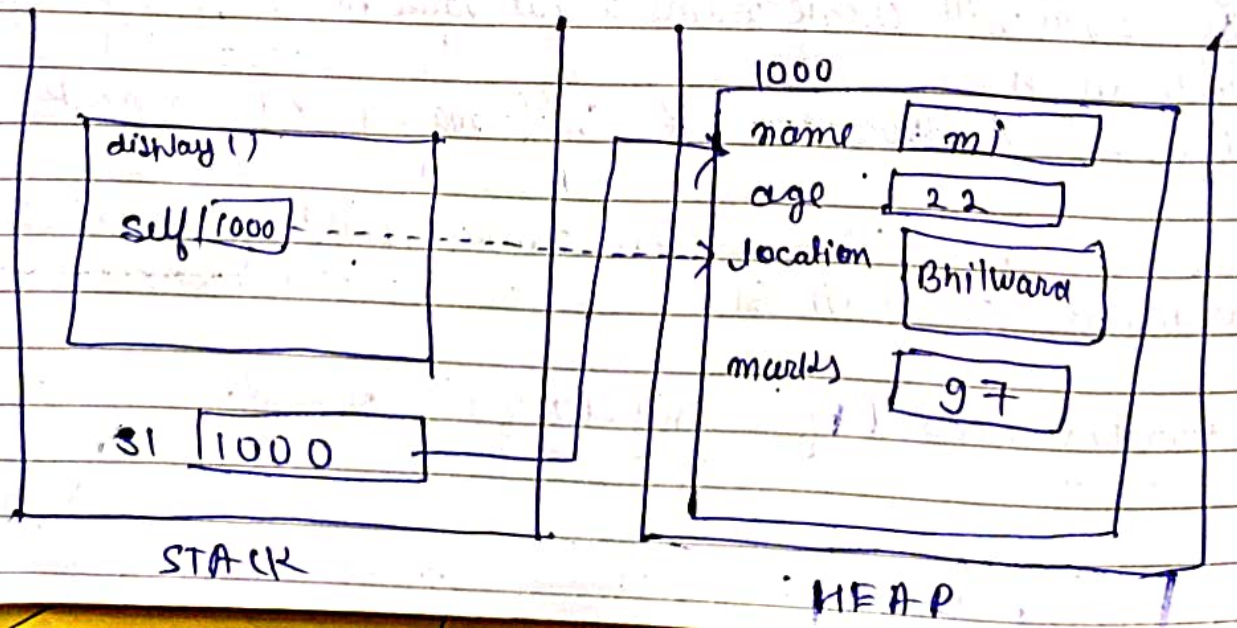
& activation record of __init__(self) is deleted.

⇒ In Python name of every constructor is __init__

⇒ it is a standard to use self to catch address of object in __init__

* ways to declearing instance/object variable

```
class student:
    def __init__(self):
        self.name = "mj"        ] # declear 1
        self.age = 22

    def display(self):
        print(self.name)
        print(self.age)
        self.marks = 97         ] # declear 3
        print(self.marks)
        print(self.location)

S1=student()
print(S1.name)
print(S1.age)
S1.location = "Bhilwara"        ] # declear 2
print(S1.location)
S1.display()
```



7 we can use # 2 & # 3 to declear object / instance variable But this is not a standard way

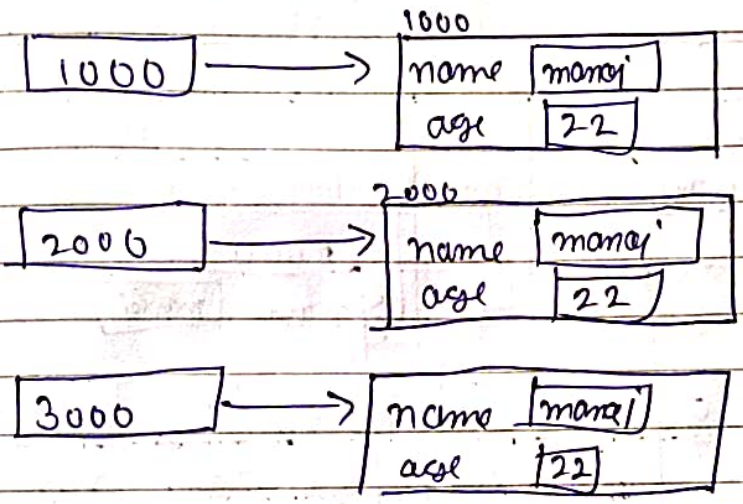
* standard way of writing classes :-

```
class student :
    def __init__(self):
        self.name = "manoj"
        self.age = 22
```

] #1 error

```
S1 = student()
S2 = student()
S3 = student()
```

] # 2 error



#1 we can't define name in constructor as all student doesn't have same name & age

#2 we need to send name and age with student object.

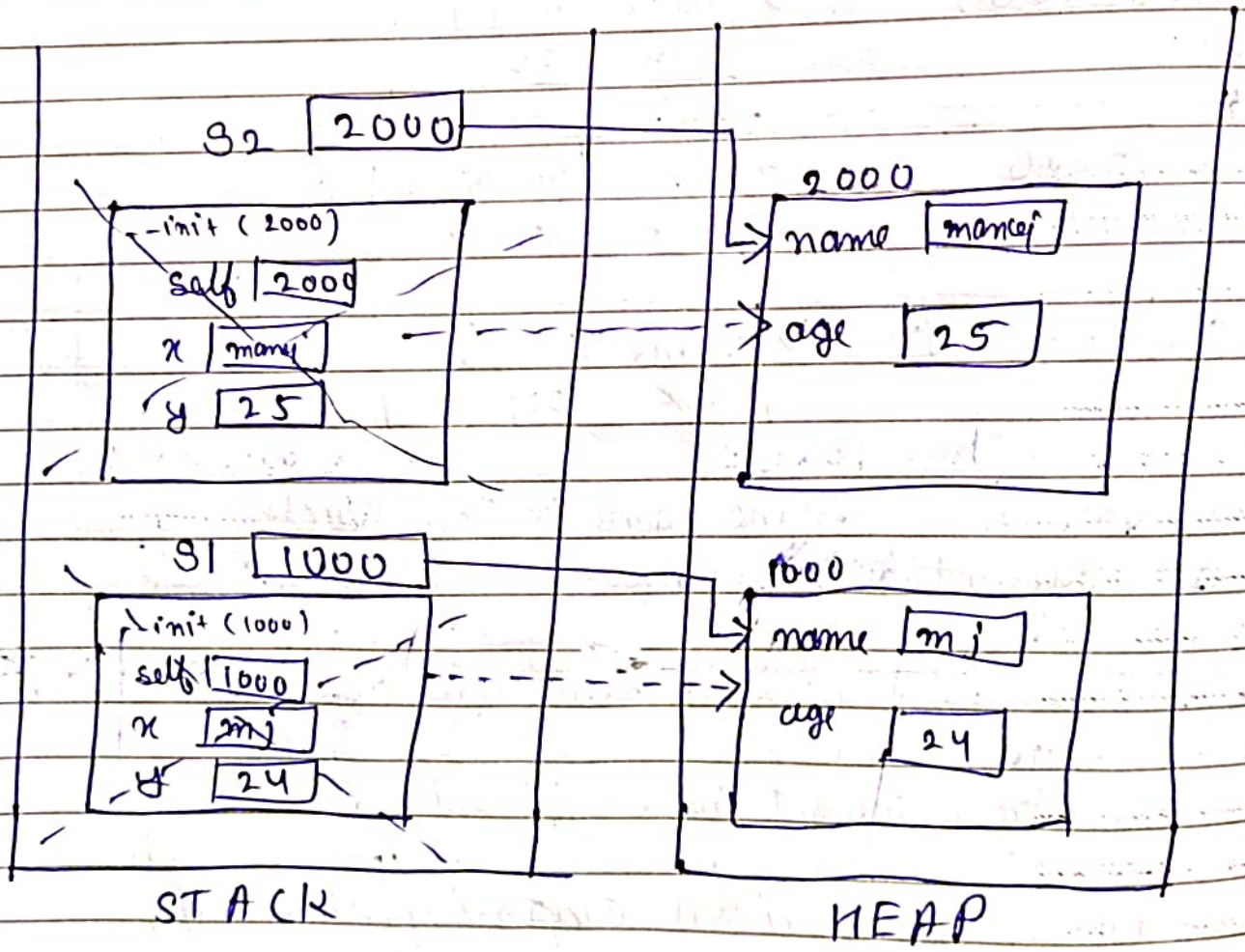
so that each student have different name & age

so the above written class is wrong

So standard way of writing class

```
class student :
    def __init__ (self, x, y) :
        self.name = x
        self.age = y

S1 = student ("mj", 24)
S2 = student ("manoj", 25)
```

PVM —invoke→ __init__ (1000, "mj", 24)
→ __init__ (2000, "manoj", 25)

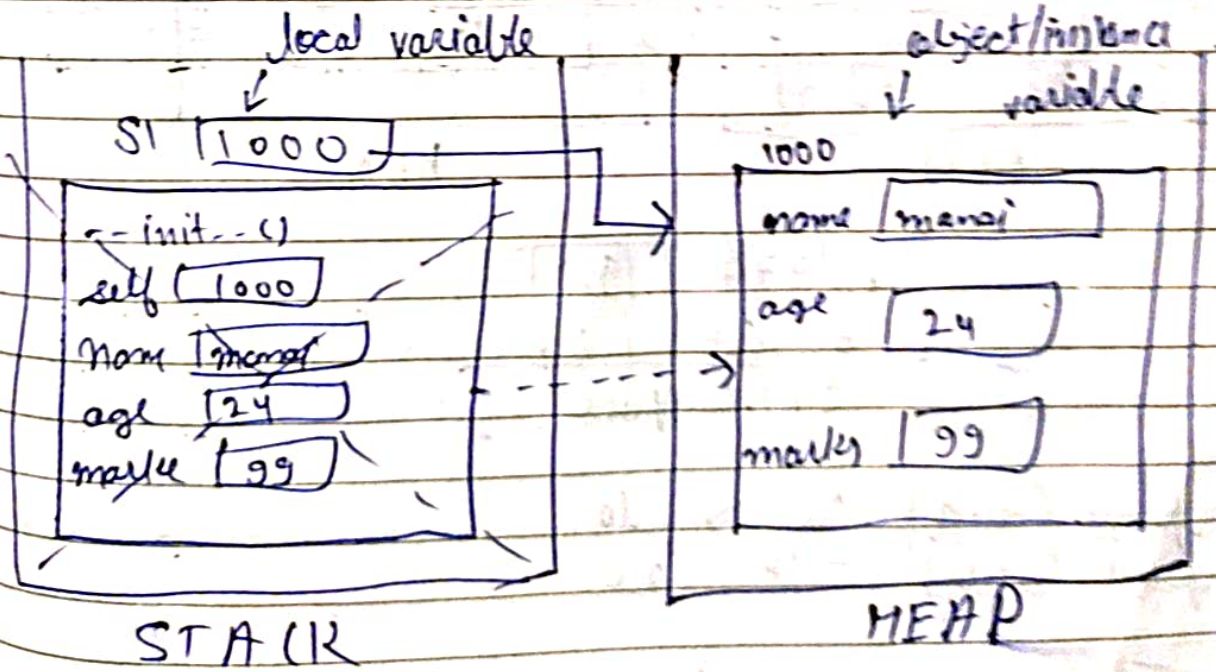


so the standard way is

```
class student :
    def __init__ ( self , name , age , marks )
        self.name = name
        self.age = age
        self.marks = marks

s1 = Student ( " manoj " , 22 , 99 )
```

Q what is diff b/w self.name & name

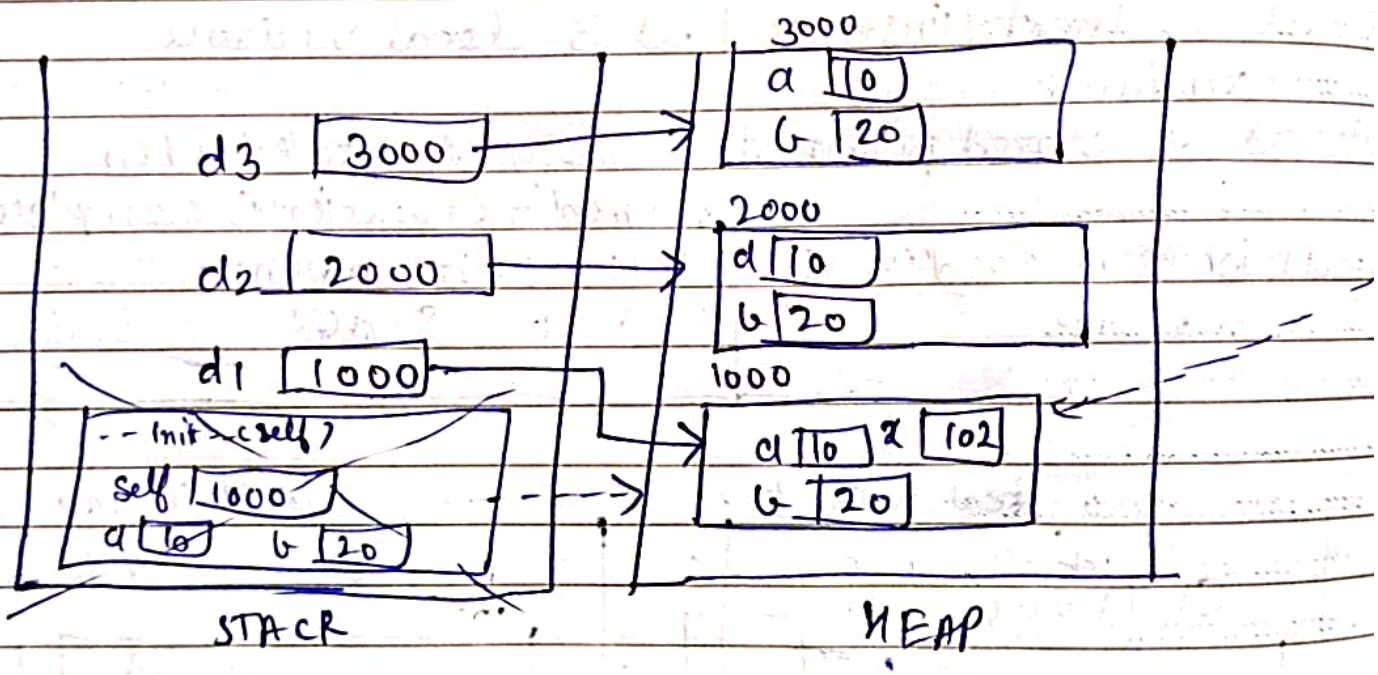
self.name	name
(i) it store in object	it store in activation record
(ii) it is object / instance variable	it is local variable
(iii) it is stored permanent	memory deallocated when method's execution is complete
(iv) this is stored in Heap	this is stored in STACK



22/12/21

* Static variable : these are declear inside class But outside __init__() or any other method

```
class Demo:
    x = 777
    y = 666
    def __init__(self):
        self.a = 10
        self.b = 20

d1 = Demo()
d2 = Demo()
d3 = Demo()
```



STATIC Space

print (d1.a) → 10
print (d1.b) → 20

print (d2.a) → 10
print (d2.b) → 20

to print static variable

print (Demo.x) → 777
print (Demo.y) → 666

note: In most of cases we can't access static variable by address variable (d1, d2, d3)

to change value of static variable

Demo.x = 999
Demo.y = 888
print (Demo.x) → 999
print (Demo.y) → 888

# if we try
d1.x = 102 then

we want to modify static variable but we just created new variable x in object d1 is Pointing

— x — x — x — x — x — x — x —

```
class Demo :
        x = 777        # for case 3 & 4

        def __init__ ( self ) :
                self. a = 10
                self. b = 20

d1 = Demo()
```

1 case :
print (d1.x) → error
because we don't have x

#2 case

```
d1.x = 100
print (d1.x) ——→ 100
```

We created new variable x for object a which d1 is Pointing & then we print it

#3 case: Now we create a static variable

```
x = 777

print (d1.x) ~~→ 777   # this time we get
d2 = Demo()            output because we
                       have x
print (d2.x) ~~→ 777
```

#4 case:

```
d1.x = 100
d2 = Demo
print (d1.x) —→ 100   # it print its
print (d2.x) —→ 777     own variable
                  # it printing static
                    variable
```

$\boxed{mj}$

it means if we try to print static variable by address variable then firstly address variable search variable in object, if he didnot find it then he search outside object

⇒ we can print static variable by address/object variable, But we can't modify static variable by address/object variable.

⇒ We can print static variable if and only if there is no variable present with same name as object / instance variable

d1,d2,d3 = address / reference variable
a1b = object / instance variable

⇒ We can modify static variable By class name only. there is no other method to modify static variable.

x — x — x — x — x — x — x

* Application of static variable

#1 count no of object created for Particular program
#2 28/12/2021

logic : no of object = no of time init() invoked

without using static variable

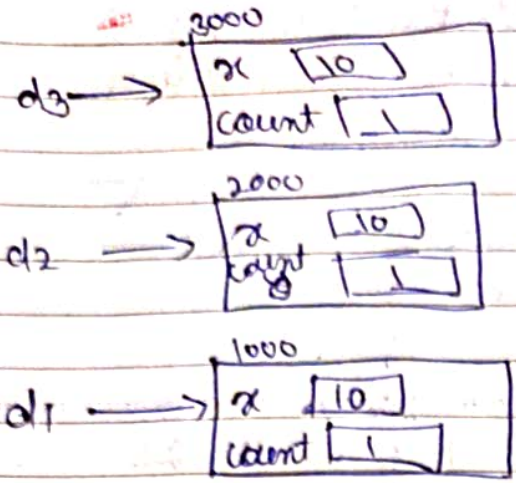
```
Class Demo :
        def __init__(self):
                self.x=10
                self.count = 0
                self.count = count+1
d1=Demo()
d2 = Demo()
d3 = Demo
```

X

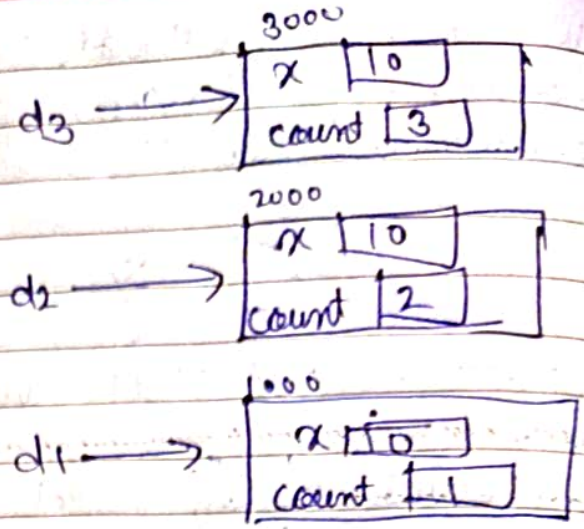
with static variable [code]

```
Class Demo :
        count = 0          # static variable
        def __init__(self):
                self.x=10
                Demo.count = Demo.count +1   # using static variable

d1 = Demo()
d2 = Demo()
d3 = Demo()
print(" no of object in this class ", Demo.count)   # printing static variable
```

without static variable ✗



with static variable count ✓



* strings : A collection of characters

☆ In Python strings are Immutable, it means we cannot change value of string after assignment

* How to declare string,

#1 'Ramu'
#2 "Ramu"
#3 """Ramu"""
#4 Ramu

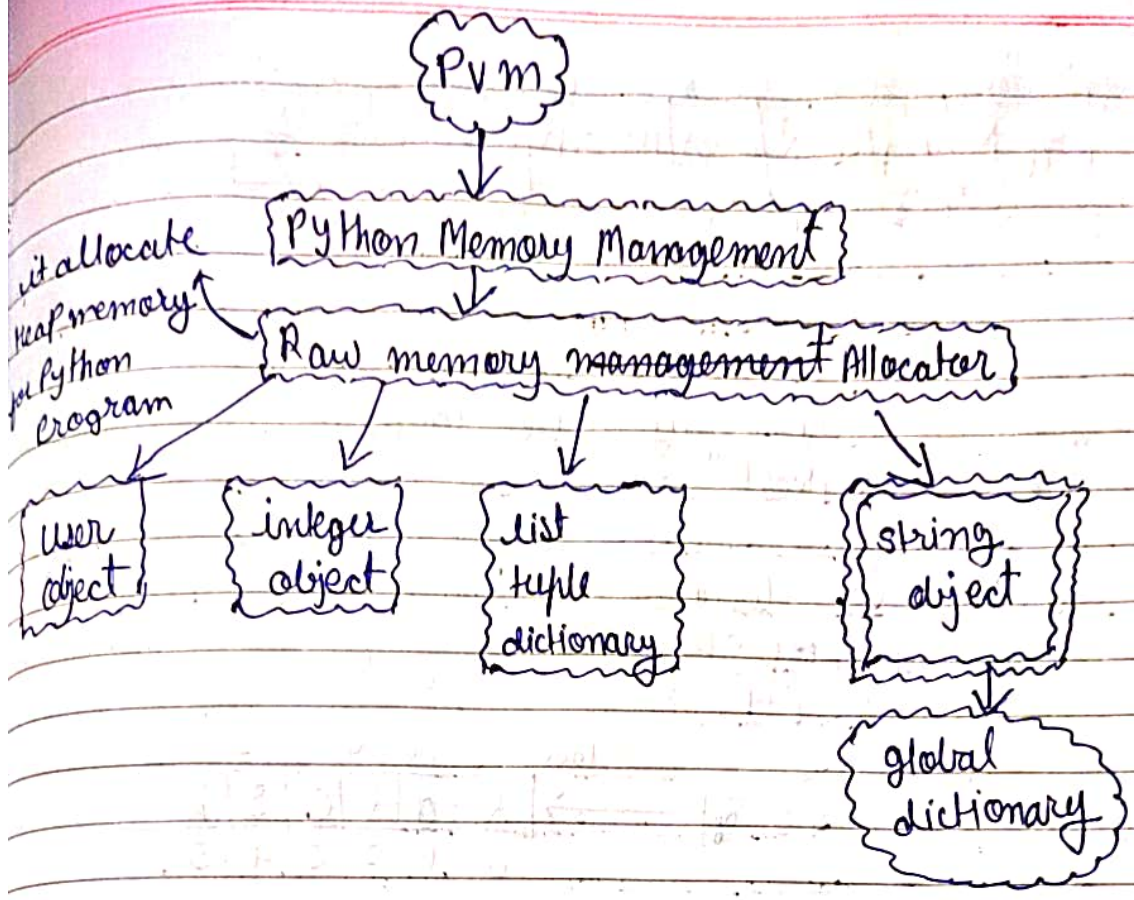
```
a = " Ramu "
print (type(a)) ⟶ <class 'str'>
```

#3

```
str = " Rakesh "
print (str) ⟶ Rakesh

for char in str :
        print (char) → "R","a","k","e","s","h"
```

it will be interpreted as

```
for char in "R","a","k","e","s","h"
```



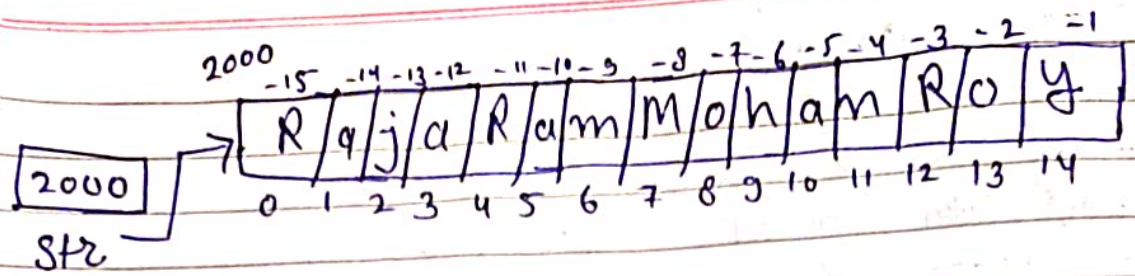
#1

```
str = " Ramu "
print (str) ⟶ Ramu
print (str[0]) ⟶ R
print (str[1]) ⟶ a
print (str[2]) ⟶ m
print (str[3]) ⟶ u
```

#2

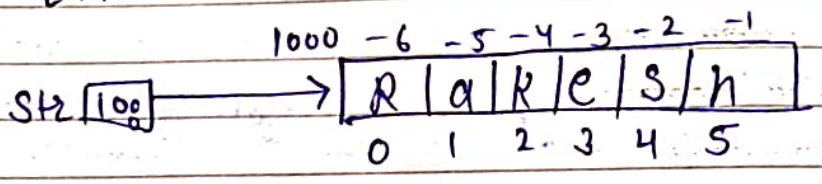
```
Str = " Raja Ram Mohan Roy "
print (Str) ⟶ Raja Ram Mohan Roy
print (Str[0] ⟶ R
print ( str[-15] ⟶ R

print ( str[14] ) ⟶ y
print ( str[-1] ) ⟶ y

print ( len (str) ) ⟶ 15
```

{ length = last index + 1 }



#4 printing string by while loop

```
str = "Rakesh"
i = 0
while ( i <= len(str)-1 ) :
        print ( str [i])
        i = i+1
```



I_1

```
i = 0
while ( 0 <= 5 ) ✓
        print ( str[0]) → R
        i = 1
```

I_2 i = 1

```
while ( 1 <= 5 ) ✓
        print ( str[1] → a
        i = 2
```

at

I_6 i = 6

while (6 <= 5) X condition false loop end

* unpacking of strings :

a, b, c = 10, 20, 30

so, a = 10
b = 20
c = 30

if a = b = c = 10, 20, 30
then

a = (10, 20, 30)
b = (10, 20, 30)
c = (10, 20, 30)

} tuples

so In this

1 way of unpacking of string

c1, c2, c3, c4 = "R", "a", "m", "u"

print (c1) → R , print (c2) → a
print (c3) → m , print (c4) → u

2 way

str = "Balu"
c1, c2, c3, c4 = str

print (c1) → B
print (c2) → a
print (c3) → l
print (c4) → u

* Arthmatic operation on string

+ , *

#1 concatenation

```
str1 = " Apple "
str2 = " laptop "
str3 = str1 + str2      # concatenation
print (str3) ⟶ Apple laptop
```

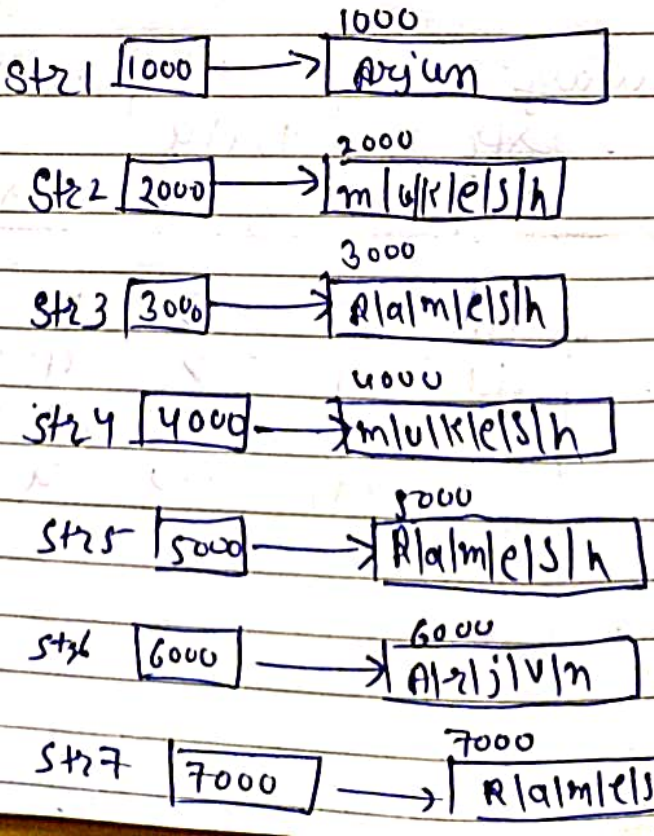
#2 multiplication

```
str1 = " Apple "
str2 = str1 * 3
print(str2) ⟶ Apple Apple Apple
```

* String Interning :

```
str1 = " Arjun "
str2 = " mukesh "
str3 = " Ramesh "
str4 = " mukesh "
str5 = " Ramesh "
str6 = " Arjun "
str7 = " Ramesh "
```

so memory mapping for this as per our current understanding



☆ But this memory mapping is wrong

String uses global dictionary for string management

How it works !

⇒ str1 comes first we check in global dictionary for "Arjun" we doesnot have any "arjun" string so we allocate memory to it

⇒ str 2 comes, we check in global dictionary for "mukesh", we doesnot have any "mukesh" string so we allocate memory to it.

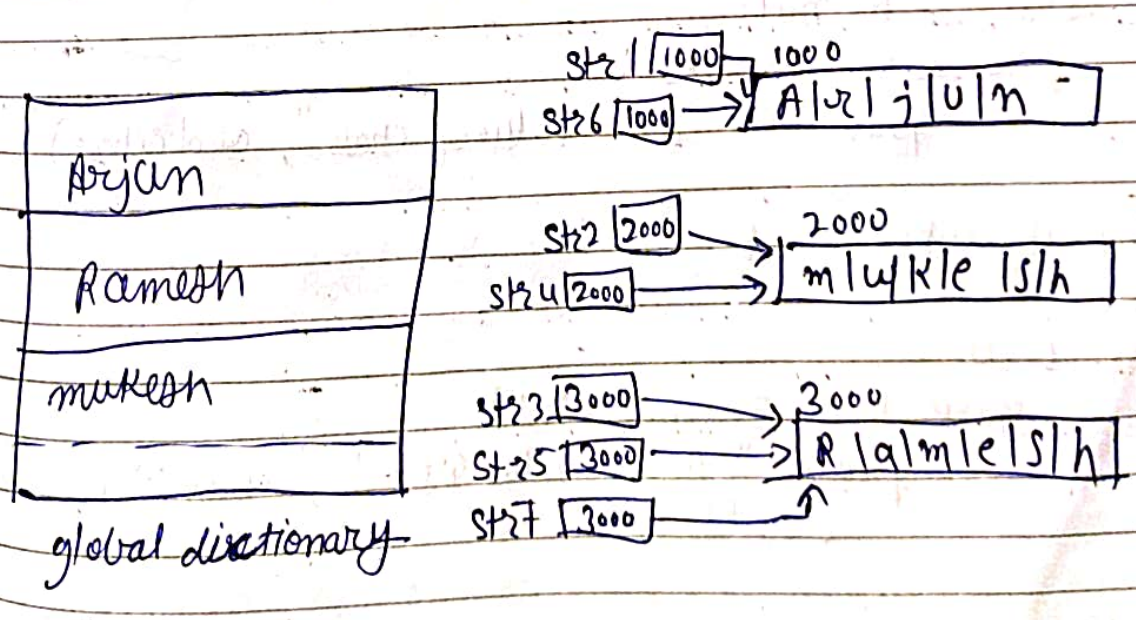
⇒ Str 3 comes, we check in global directory for "Ramesh", we don't have it so we allocate memory to it,

Now, Str 4 comes we already have "mukesh" So we return address of "mukesh" to Str 4.

Simil. str 5 we already have "Ramesh" so we return address of "Ramesh" to Str 5.

so

correct memory mapping for it.



So, firstly data will be checked in global dictionary

if data not present
then allocate memory to data & return address

if data present
then return address of existing data
we did not allocate memory to new data
] ie know as string interning

* ASCII code : Standard value of char

a - z A - Z
97 122 65 90

diff b/w A - a = 32

ASCII -> American standard for information interchange

[code] print ASCII value of characters

#1 str = "Ramu"
for char in str :
print (" ASCII value of char ", ord (char))

#2 By while loop

i = 0
while (i <= len(str) - 1)
print (" ASCII value is : ", ord(str[i]))
i = i+1

I1 i = 0 ✓ 0 <= 3 R -> 82 i = 1

I2 i = 1 ✓ 1 <= 3 a -> 97 i = 2

I_3 $i=2$ ✓ $2<=3$ $m \rightarrow 109$ $i=3$

I_4 $i=3$ ✓ $3<=3$ $u \rightarrow 117$ $i=4$

I_5 $i=4$ ✗ $4<=3$ loop end

* String comparision

str1 = " Ramu "
str2 = " Ramu "

	R	a	m	u
Str1	82	97	109	117
Str2	82	97	109	117

Algo :

we will do subtraction of ASCII value
if we get zero as result ie same ASCII value, ie same character
if we get non zero as result ie different ASCII value ie different character
so at last we get zero as return ie
strings are equal
else
strings are not equal

ex 1 Str1 = " Ramu " Str2 = " Rama
82 97 109 117 82 97 109 97

sub

R	82 - 82	= 0 ✓	move ahead
a	97 - 97	= 0 ✓	move ahead
m	109 - 109	= 0 ✓	move ahead
u - a	117 - 97	= +20 (+ve nonzero)	stop

(i) Str1 & Str2 are not equal
(ii) Str1 > Str2 because +ve nonzero is returned

u > a
117 > 97

27/12/21

ex 2 Str1 = "Ramu"
Str2 = "Rani"

Str1	R	a	m	u
	82	97	109	117
Str2	R	a	n	i
	82	97	110	105
	0	0	-1	

✓ ✓ ✗ stop here

-1 is returned

so

Str1 < Str , -ve nonzero is returned

∵ m < n

∵ 109 < 110

[code] WAP to compair strings

```
Str1 = input (" enter string 1 : ")
Str2 = input (" enter String 2 : ")
if ( Str1 == Str2 ):
    print ( Str1, "is equal to", Str2 )
else if ( Str1 > Str2 ):
    print ( Str1 , " is grather than ", Str2)
else:
    print ( Str1 , " is less than", Str2)
```

execut y:

Str1 = Ramu
Str2 = Ramu

R	a	m	u	R	a	m	u
~~82~~	~~97~~	~~109~~	~~117~~	~~82~~	~~97~~	~~109~~	~~117~~

0 0 0 0 → equal

ex 2: Str1 = Balu
Str2 = Bala

B	a	l	u		B	a	l	a
66	97	108	117		66	97	108	97

0 0 0 +20 so Bala > Balu

ex3: Str1 = Raja
Str2 = Rani

R	a	j	a		R	a	n	i
82	97	106	97		82	97	111	97

0 0 -5 so ~~Ra~~ Raja < Rani

ex 4: Ramu Ramesh

R	a	m	u	>	R	a	m	e	sh
82	97	109	117		82	97	109	101	105

0 0 0 +16
u > e so Ramu > Ramesh

[code] WAP to change case of string. (L->U)

Psudo code:
→ Accept the string
→ i = 0
→ newstr = " "
as long as i <= len(Str)-1
data = fetch the character at ith index
Ascii = find Ascii value of data
if (Ascii >= 65 && ascii <= 90)
add data to newstr
else
new ascii = ascii - 32

→ convert Ascii value to charactor & Store in data
→ add data to new Str
→ i = i+1
}
print value of new Str

{ chr() → it convert Ascii value to charactor }

Code

```
str = input(" Enter string : ")
i = 0
New Str = " "
while ( i <= len(Str)-1 )
        data = Str[i]
#ord method     Ascii = ord (data)
                if (ascii >= 65 and ascii <= 90):
# concatenation     newStr = newStr + data
                else
                        ascii = ascii - 32
# char()                data = char (ascii)
# concatination         new Str = newStr + data

print (new Str)
```

Code Ury using Built in f^n /method

```
str = input(" Enter String ")
Str1 = Str.lower()          # lower case
print ( Str1 )
Str2 = Str.upper()          # upperase
print ( Str2 )
str3 = Str.swapcase()       # swapcase
print ( Str3 )
```

28/12/21

#2 Application of static variable

=> when we have common value of variable for all object then we can make variable static, so that we have single copy of variable across all object

ex

```
class farmer :
        rate = 5    # static variable

    def __init__(self, principle, time):
        self.principle = principle
        self.time      = time
        self.rate      = 5

    def calcsi (self):
        # si = self.principle * self.time * self.rate /100

    si = self.principle * self.time * farmer.rate /100

        print(si)

f1 = farmer(10000,2)
f2 = farmer (20000,3)
f3 = farmer (30,000,10)
```

using static variable by class name only

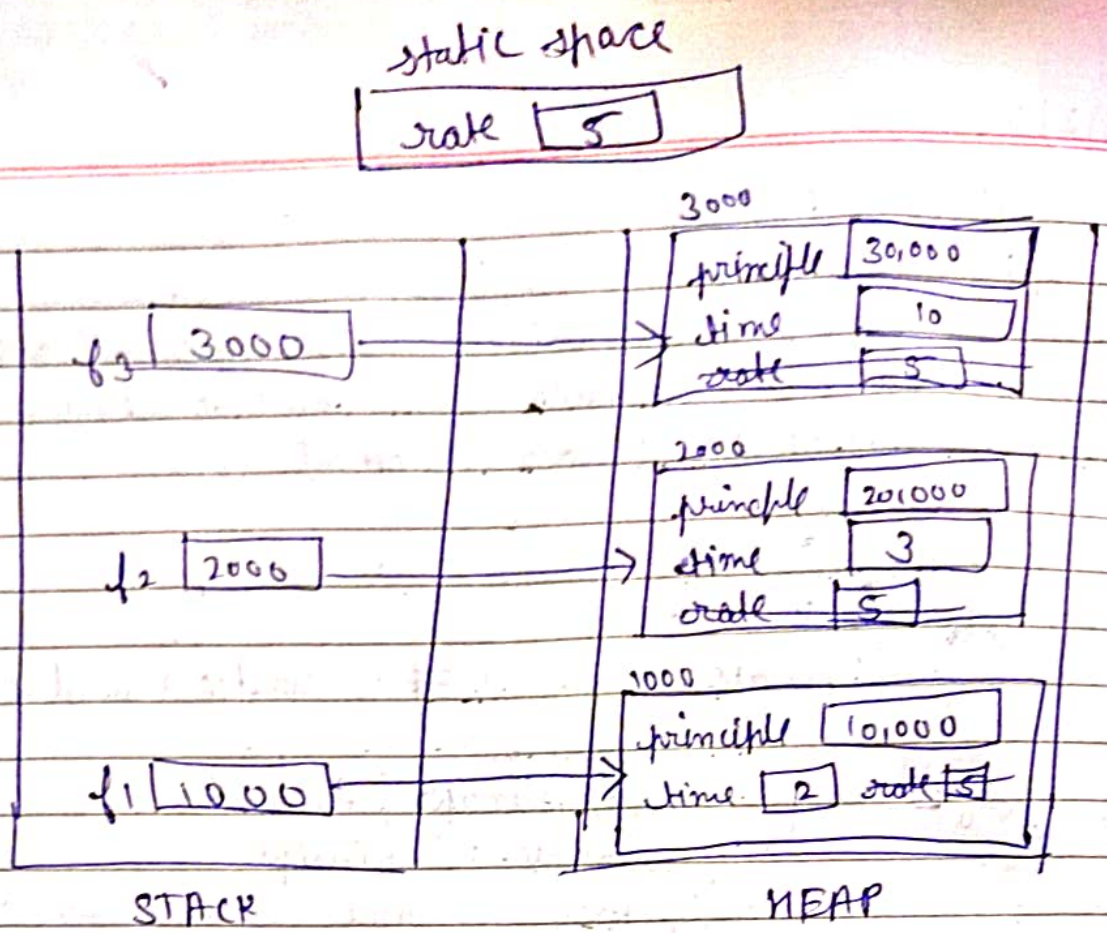
In above mention code we have rate = 5 which is common & fix value for all object, because we already fix it.

Now, we create object then.

we create 3 object

so all 3 object has principle, time, $\boxed{\text{rate}}$ as object variable

so for 3 object we have 3 copies of each variable



But if we create 5 lakh object then each obj variable has 5 lakh copy of itself.

so

we have 5 lakh cop of rate also

let 1 copy take 3 Byte to store

so 5 lakh copy will take 15 lakh Byte

so huge amount of storage is waste to store comon & fix value variable

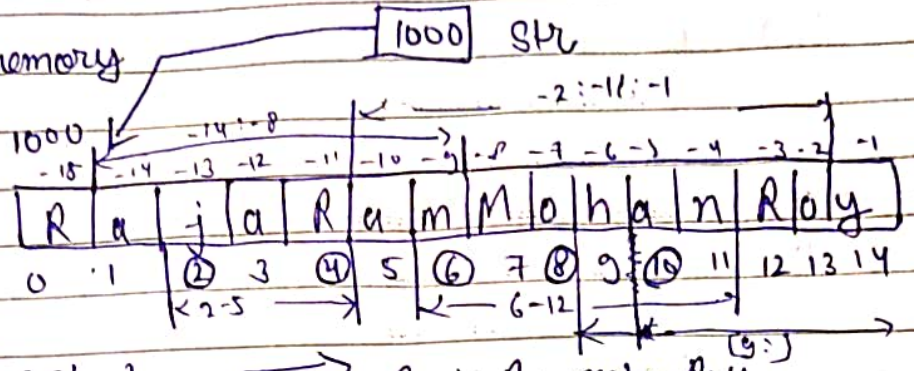
so

we make rate as static, so that only one copy of rate is share among all object, & we save our memory.

* String slicing :-

str = " Raja Ram MohanRoy

So In memory



print (Str) → Raja Ram mohan Roy
print (str[0]) → R
print (str[8]) → o
print (str[2:5]) → jaR
print (str[6:12]) → mMohan
print (str[9:]) → hanRoy
print (str[:]) → Raja RamMohanRoy
print (str[2:11:2]) → jRmoa

{ print (str[start: stop : steps to take]) }

print (str[6:1]) → No output 1 ← 6 X

print (str[6:1:-1]) → maRaj (6 inclusive 1 exclusive)

-1 for reverse

reverse dir^n ↗ 1 ↗ 1 step

print (Str [-14 : -8] → a jaRam
print (str [-2:-11: -1] → oRnahoMma

print (str [-2 : -11] → No output, no error
← -ve dir^n not mentioned

print (str [-11:-4:2]) → R m o a
print (str [: :]) → Raja Ram Mohan Roy
↓ ↓ ↓
← default

print (str [Start at Begining : end at last : +1])

reverse a string

print (str [: : -1] yoRnahaminaRajaR

value error (steps never zero)
print (str [8:12:0]) → value error
↓ steps is zero

* strings method

#1 replace ("this ", "with this ")

str1 = " Python is cool "
str2 = str1.replace (" is ", "will be ")
print (str2) → Python will be cool

#2 starts with (" string "), end with (" string).

str = " if you think you can, you will "
print(str.Starts with ("if")) → True
print. (str. starts with ("you")) → False
print (str. ends with ("will")) → True
print (str. ends with ("think")) → False

#3 count (" string")

str = " if you think you can you will"
print (str. count ("you") → 3
print (str. count ("if") → 1

#4 in operator : it check wheather string 1 is present in another string or not.
it gives true or false

```
str = " if you think you can you will "
str1 = " you "
str2 = " rabbit "
print (str1 in str) → True
print (str2 in str) → False
```

#5 find, rfind, index, rindex

```
                                  22 ←
str = " if you think you can you will "
        0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
        →
print (str.find("you")) → 3 →
print (str.rfind("you")) → 22 ← reverse
print (str.index("you")) → 3 →
print (str.rindex("you")) → 22 ← reverse
```

find / index & rfind / rindex work similarly when string is present But if string not present

★ find → will return -1
index will return error, value error.

29/12/21

6 format ( ) / string formatting

ex 1

```
name = input ("Enter name")
print ("Hi {} ".format(name))
```

ex 2

```
name = input ("Enter name:")
age = input ("Enter age:")
marks = input ("Enter marks:")
```

way 1

```
print ("Name = {} ".format(name))
print ("age = {} ".format(age))
print ("marks = {} ".format(marks))
```

way 2

```
print ("name={0} \n age ={1} \n mark=
{2} ".format(name, age, marks)
```

0 1 2 indexing

way 3

```
print ("Name = {2} \n Age = {1} \n
marks = {0} ".format(marks, age, name
```

0 1 2

O/P

```
Enter name : manoj
Enter age : 22
Enter marks : 99

Name = manoj
age = 22
marks = 99
```

7 isdigit() / isalpha() / isalnum()

```
str = input(" Enter the string ")
if (str.isalpha()):
        print(" String is Alphabate only ")
if (str.isdigit()):
        print(" string is digit only ")
if (str.isalnum()):
        print("string contain alphabate & digit ")
```

I/P/O/P

ramu -> string is alphabate only
12345 -> String is digit only
MJ5284 -> string contain alphabate & digit

8 strip(), lstrip(), rstrip()

⇒ to remove spaces from right, left, Both side

```
str = "          Python is cool          "
print(" start ", str, " End ")
str1 = str.lstrip()          # left strip
print(" start ", str1, " End ")
str2 = str.rstrip()          # right strip
print(" start ", str2, " End ")
str3 = str.strip()           # Both side strip
print(" start ", str3, " End ")
```

O/P

```
Start          Python is cool          End
Start Python is cool          End
Start          Python is cool End
start Python is cool End
```

WAP to remove all space in a string

Psudo code

Accept string
new str = " "
i = 0
as long as i is not equal to length of string - 1 :
Extract value at i^{th} index & store it in data
if (the character at i^{th} index is space :
move to next character
else :
add the character to the index of newstr
move to next character
print new.str

code

```
str = input (" Enter the string ")
new str = " "
i=0
while ( i<= len(str)-1)
        data = str[i]
        if (data == " "):
                i = i+1
        else:
# String concatenation  new str = newstr + data
                i = i+1
print (new str)
```

I/P O/P

	P	y	t	h		o	n						
0	1	2	3	4	5	6	7	8	9	10	11	12	13

"p" "y" "t" "h" "o" "n"

data

PYthON

newStr

Strings are Immutable

=> In strings, operation other than accessing & printing will done on copy of string ★

=> Actual string is immutable

=> we can't modify string after assignment

=> modifying string is ~~m~~ not allowed in Python.

ex#1 str = " Rakesh "
a = str [3]
print (a) → a

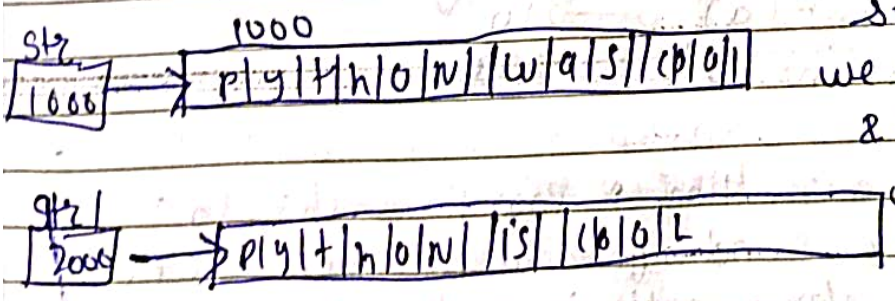
★ str[5] = Z → type error
print (str) # 'str' object doesnot support item assignment.

To prove strings are immutable

[code]

```
str = " Python was cool "
str1 = str.replace (" was ", " is ")
print (str1) → Python is cool
print (str) → Python was cool

print (id (str))
print (id (str1))
```

⇒ different id ie different object
ie str & str1 are different string
we create copy of str & then modify it then store it in str1



str1 is a copy of str, str doesnot modified at all

* concept of Actual & formal Parameter

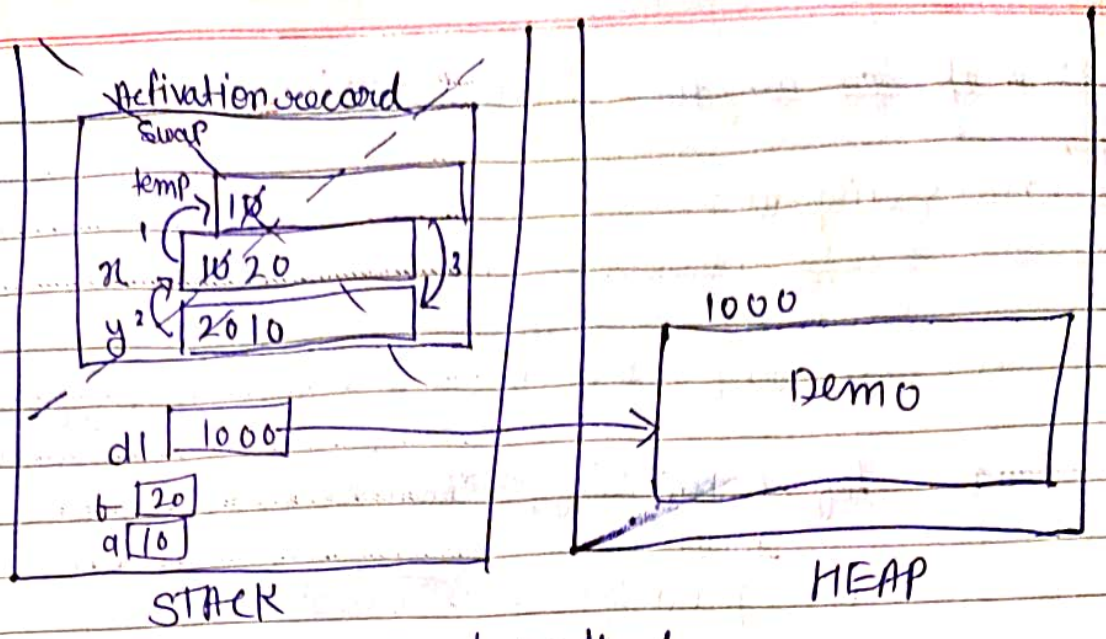
```
a = 10
b = 20
print ("befor swapping")

class Demo:
    def __init__(self):
        ____________
activation record created →  def swap(self, x, y):   # 10, 20
                                temp = x            # x & y are formal parameter
                                x = y
                                y = temp
                                print ("value after swapping")
                                print(x) → 20
activation record deleted       print(y) → 10

a = 10
b = 20   ] actual Parameter
print (befor swapping)
print(a) → 10
print(b) → 20
d1 = Demo()
d1.swap(a, b)
print ("after swapping")
print(a) → 10
print(b) → 20
```

10, 20

No return value

When we invoke a method & Pass Parameter to it then we send formal Parameter to it.

* formal Parameter works in method only so modification on formal Parameter doesn't reflect on Actual Parameter.



So, we send 10, 20 to method
In method we swap values, But when we come outside from method our modification does't work,
→ the modification we did inside method, will not reflect outside the method

Because,
x & y are copies of a, b so, whatever change we made on x & y will not reflect on a, b
also,
x & y are deleted with Activation record
so, changes done inside the Activation record, will not reflect outside the Activation record

x & y ⇒ formal Parameter
a & b ⇒ actual Parameter

⇒ We ~~send~~ use formal Parameter inside method, change in formal parameter will not reflect on Actual Parameter.

* Concept of default Argument :

```
class display :
    def __init__(self):

#1  def disp ( self, x, y, z )   # without default argument
                                 # with.
#2  def disp ( self, x=666, y=777, z=888)
        print (x)
        print(y)
        print(z)
```

O/P	#1,2	#1	#2	②	④	⑤	⑥	⑦	⑧
x	10	10	10	666	10	100	666	666	10
y	20	20	20	777	777	200	777	20	20
z	30	error	888	888	888	300	30	30	30

```
a = 10, b = 20, c = 30
d1 = display ()
① d1. disp( a,b,c )
② d1. disp (a,b)      # Here #1 we get error, because
③ d1. disp ( )          we did not send all Parameter
④ d1.disp (a)           But #2 we give output as
⑤ d1.disp( 100,200,300)   it has default Parameter.
⑥ d1.disp (z=c)
⑦ d1.disp ( y=b, z=c )
⑧ d1.disp( x=a, y=b, z=c )   # Keyword argument.
```

↓

Keyword argument

* calculator

```
class calculator :
    def add (self, x, y) :
        res = x+y
        print (res)      -> 30

    def sub (self, x, y) :
        res = x-y
        print (res)      -> 10

    def mul (self, x, y) :
        res = x*y
        print (res)      -> 200

    def div (self, x, y) :
        res = x/y
        print (res)      -> 0.5

c1 = calculator()
a, b = 10, 20
c1.add (a, b)
c1.sub (a, b)
c1.mul (a, b)
c1.div (a, b)
```

3/11/22

* Returning multiple values, How to catch it.

```
class calculator:
    def arthops (self, x, y):
        res1 = x+y
        res2 = x-y
        res3 = x*y
        res4 = x/y
        return res1, res2, res3, res4
```

returning multiple values

```
c1 = calculator()
a, b = 10, 20          # assignement multiple variable

x1, x2, x3, x4 = c1.arthops(a, b)
```

catching multiple returns

```
print(x1)  -> 30   (+)
print(x2)  -> -10  (-)
print(x3)  -> 200  (*)
print(x4)  -> 0.5  (/)
```

* Passing variable Parameter to method

```
class calculator:
    def add (self, *value):
        i = 0
        sum = 0
        while (i <= len(value)-1)
            sum = sum + value[i]
            i = i+1
        return sum
```

it act as array

c1 = calculator # object creation

a = 10
b = 20
c = 30
d = 40
e = 50

} multiple variable declaration & assignment

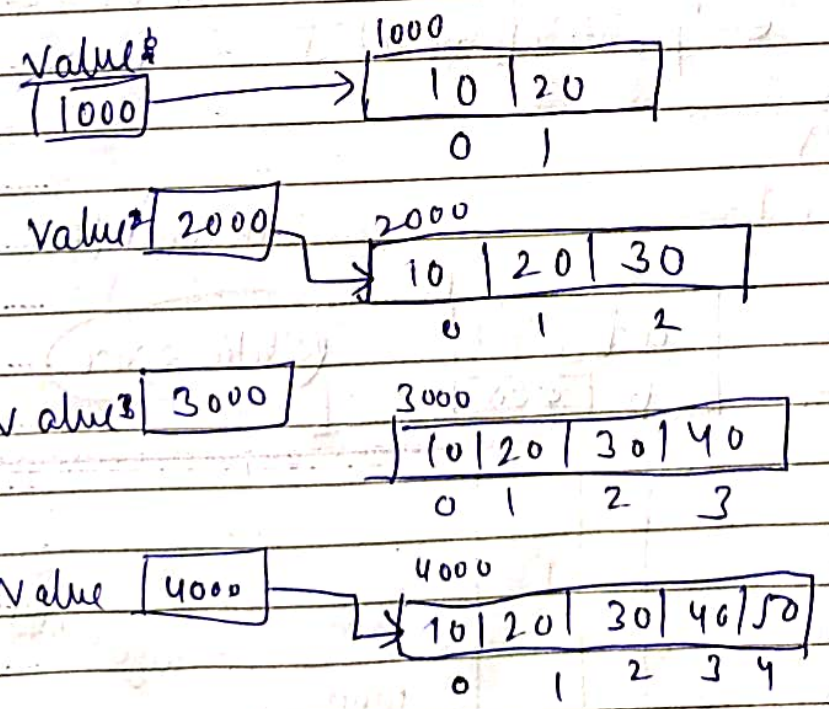
sum1 = c1.add(a,b)
sum2 = c1.add(a,b,c)
sum3 = c1.add(a,b,c,d)
sum4 = c1.add(a,b,c,d,e)

} calling method for diff. Parameter

print(sum1) → 30
print(sum2) → 60
print(sum3) → 100
print(sum4) → 150

] if we do not create *value then method will give error of missing Parameter

memory mapping



* diff b/w local variable, instance variable, static variable

```
class demo:
    a = 777
    b = 888
```
} static variable declear

```
    def __init(self):
        self.x = 20
        self.y = 10
```
} instance variable

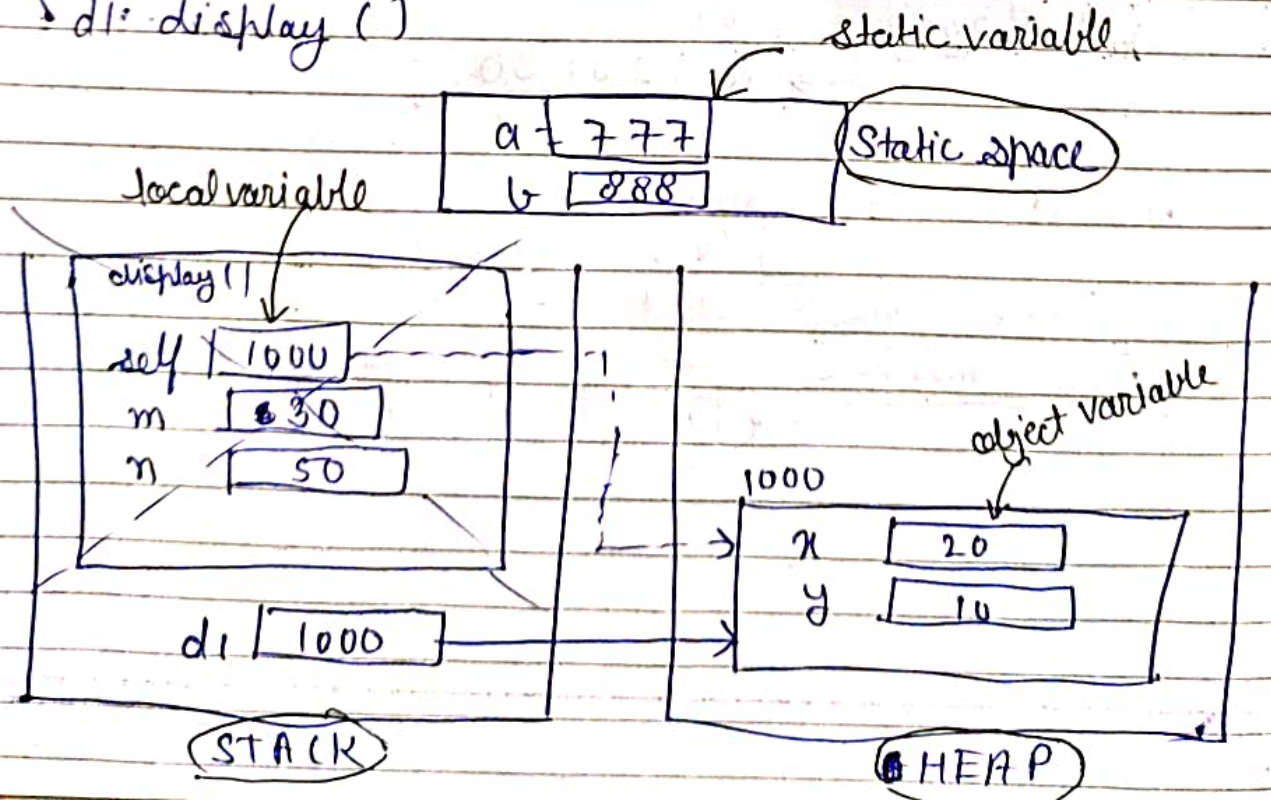
```
    def display():
        m = 30
        n = 50
```
} local variable

```
30 ← print(m)
50 ← print(n)
```
} local variable printing

```
20 ← print(self.x)
10 ← print(self.y)
```
} instance variable printi

```
777 ← print(Demo.a)
888 ← print(Demo.b)
```
} static varable printing

```
d1 = Demo()
d1.display()
```



Static variable	instance variable	local variable
(i) declear inside class but outside constructor or any other method	declear inside constructor & other method (not standard)	define inside class methods other than constructor
(ii) stored in static space	Stored in Heap	stored in Stack
(iii) memory allocated when class is created when program load on code segment	memory allocated when object is created	memory allocated when method is called (Activation record)
(iv) Accessing values by class name ie Demo.a Demo.b	Accessing values by object (outside class) by self (inside class)	Accessing value inside method only. After execution of method Activation record deleted
(v) when program is terminated, when memory deallocated	when object become garbage, memory deallocate	memory deallocated when method execution finish

* Static method :

```
class Demo :
    x = 99                    # static variable

    def __init__(self):
        self.a = 10           } instance variable
        self.b = 20

    def instancemethod (self):  } instance method
        print (self.a)
        print (self.b)
#decorator
    @ static method
    def staticmethod ( ) :      } static method
        print ( Demo.x)

d1 = Demo()
d1. instancemethod() # calling instance method
Demo. staticmethod() # calling static method
```

O/P

```
10
20
99
```

note : in static method we can't use instance variable as there is no self for address.
we can access static variable only

In instance method we can access static variable & instance variable (Both).

ex simple interest program .

application_static_2

* function :

method outside a class is known as function

syantax :

```
def function name ( Parameter ):
    ____________
    ____________   } Body
    ____________
```

* type of function

```
def add (x,y) :
    sum = x+y
    return sum
```

} with Parameter with <u>return</u>

```
def sub ( x,y)
    diff = x-y
    print (" diff is ", diff)
```

} with Parameter, without return

```
def mul ( ) :
    a = 10
    b = 20
    product = a*b
    return product
```

} without Parameter, with <u>return</u>.

```
def div ( ) :
    a = 20
    b = 10
    divi = a/b
    print (" division is ", divi)
```

} with out Parameter, without return

```
m, n = 10, 20
sum1 = add (m,n)
sub (m,n)
prod1 = mul()
div( )
```

} # calling method

* default ~~arg~~ argument:

```
def function1 (x= 999, y=888, z=777)
    print (x)
    print (y)
    print (z)
```

↳ deafault argument

		x	y	z
function1(10, 20, 30)	→	10	20	30
a, b, c = 10, 20, 30	→	10	20	30
function1 (a, b, c)	→	10	20	777
function1 (a, b)	→	10	888	777
function1 (a)	→	999	888	777
function1 ( )	→	30	888	777
function1 (c)	→	999	888	30
function1 (z = c)	→	10	20	30
function1 (x=a, z=c, y=b)	→			

↳ keyword argument

*

```
def add (*value):
    i, sum = 0,0
    while (i<= len (value)-1)
        sum = sum + value[i]
        i=i+1
    return sum

sum1 = add (10,20)
sum2 = add (10,20,30)
sum3 = add (10,20,30,40)

print (sum1)  ⟶  30
print (sum2)  ⟶  60
print (sum3)  ⟶  100
```

4/1/22

* global variable :

```
a = 999        # global variable

def fun1():
    c = 30
    print(c)
    print(a)
```

local variable → 30 → printing global variable → 999

fun1()

fun2()

print(a) → 999

```
def fun2():
    b = 20
    print(b)
    print(a)
```

local variable

b → 20

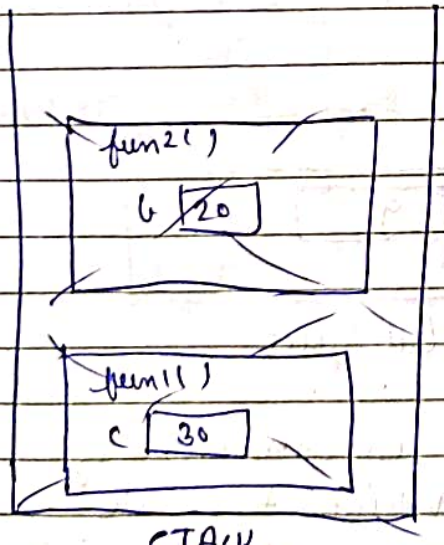
print(b) → 999

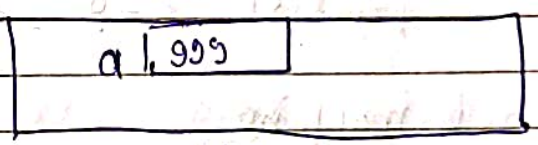
print(a) → printing global variable

so, any Part of program can access global variable.

→ Here in function we are not dealing with object (as we are now outside class) so we do not use HEAP.

→ for global variable we use global space.





Global space

⇒ In print(a) statement firstly PVM search a in Activation record. if (found) then print.
if not found then search in global space & print it
if not found then give error.

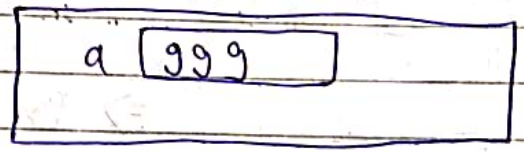
* Name clash :

if we use a variable, which is global variable inside function (without specifying it as global). we modify it, then firstly PVM search variable in activation record. if it is not found, then create a local copy of that global variable

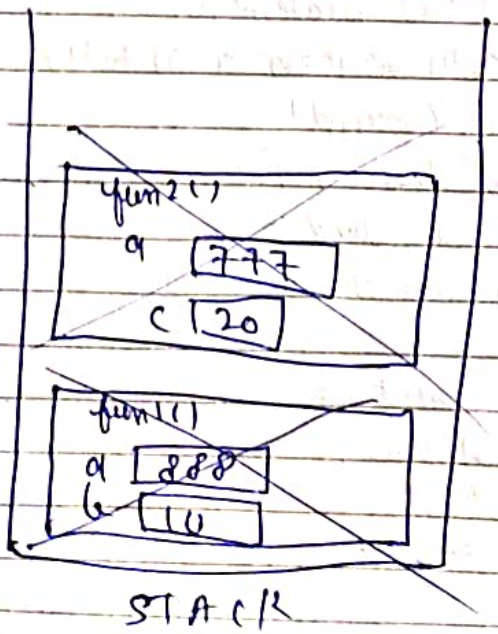
```
a = 999
def fun1():
        a = 888 # same name as global variable (But not specify)
        b = 10
        print(a) → 888
        print(b) → 10

def fun2():
        a = 777
        c = 20
        print(a) → 777
        print(c) → 20
```

```
fun1()
fun2()
print(a)
↓
999
```

* memory mapping



Global space

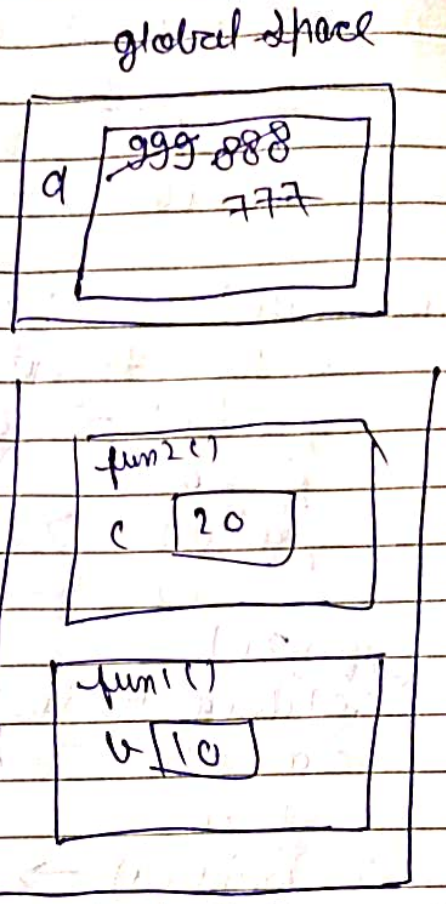


STACK

So, if we use same name of global variable & create new variable without specifying, then PVM create local copy of variable

* declear global variable inside function

```
a = 999
def fun1()
    global a      # → global variable specifing
    a = 888
    b = 10
    print(a) → 888
    print(b) → 10

def fun2()
    global a
    a = 777
    c = 20
    print(a) → 777
    print(b) → 20

fun1()
fun2()
print(a) → 777
```



note: global a means it is a global variable, PVM do not create its local copy.

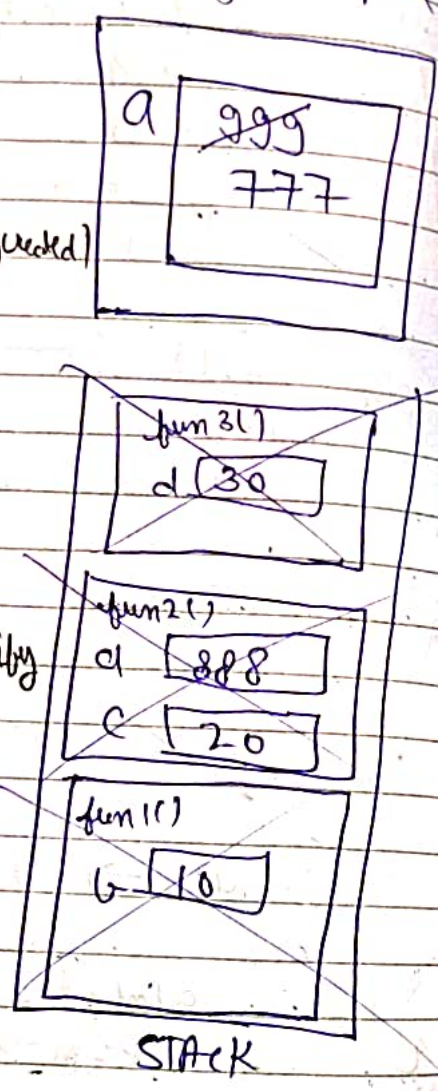
★

* Use of global ~~variable~~ Keyword

(i) to access & modify global variable inside function

(ii) if we use global Key word, then local copy of variable is not genrated.

```
def fun1()
    global a    # global variable declear
    a,b = 999,10  # assignment
    print(a) -> 999
    print(b) -> 10

def fun2()
    a=888   # Not specify (local copy created)
    print(a) -> 888
    c=20
    print(c) -> 20

def fun3()
    global a  # global copy
    a=777   # global copy of a modify
    d=30
    print(a) -> 777
    print(b) -> 30

fun1()   # calling fun1()
fun2()   # calling fun2()
fun3()   # calling fun3()
print(a) -> 777.
```



(**) Invoking function through variable

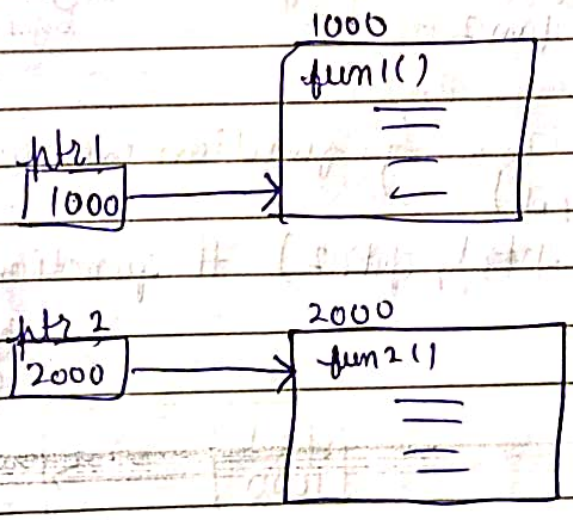
```
def fun1():
    print ("inside fun1")
```
} defining functions

```
def fun2(x,y):
    print (inside fun2: ", x+y)
```
} defining function 2

```
fun1()
a,b = 10,20
fun2(a,b)
```
} function call with function name

```
ptr1 = fun1
ptr2 = fun2
```
} function assigning to variable

```
ptr1()
c,d = 30,40
ptr2(c,d)
```
} function call with variable name



note: In Python, internally function are treated as object, so they have address & we can return & store their address in variable. so, variable can refer them.

* function as Parameter in a function call.

```
def fun1():
    print(" inside fun 1 ")

def fun2(x,y):
    print(" inside fun2 ", x+y)

def display(ptr3, ptr4):
    print(" In display ")
    ptr3()
    e,f = 30,40
    ptr4(e,f)
```

defining functions

```
fun1()
a,b,c,d = 10,20,30,40
fun2(a,b)
```

} function call with function name

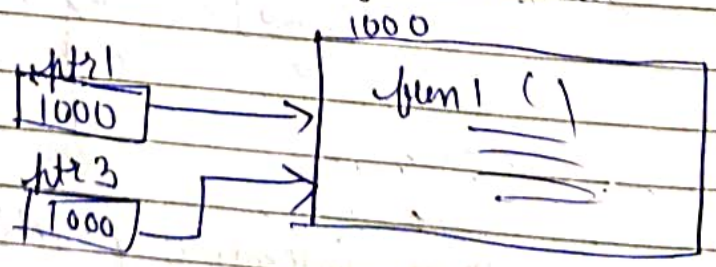
```
ptr1 = fun1
ptr2 = fun2
```

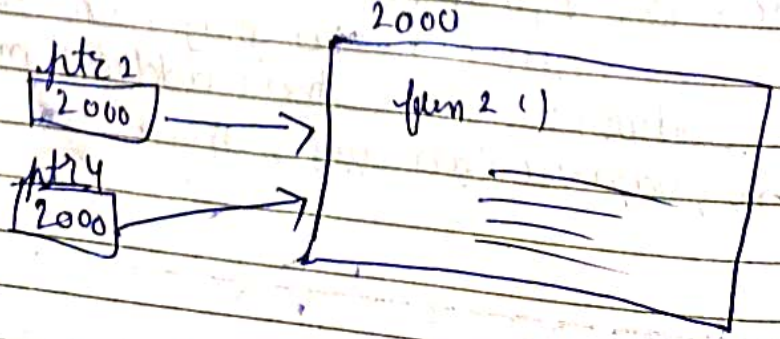
} function assigning to variable

```
ptr1()
ptr2(c,d)
display(ptr1, ptr2)
```

} function call with variable name

function as Parameter in function call





5/1/22

Higher order function: any function that takes another function as one of the Parameter is known as Higher order function

first class function: any function that Not takes another function as one of the Parameter is known as first class ~~Para~~ function.

why & when we use function: if we want to use repeatedly a set of instruction then we put them in function & call function again & again.

→ function save coding time & execution time

note:

```
def add()
    a=10
    b=20
    print (a+b)

add()
```

✓ ✓

```
add()
```

X

```
def add()
    a=10
    b=20
    print (a+b)
```

X

because Body of function always come before the function call.

* lambda function :

Syantax :

{ lambda Argument : Expression }

Keyword

(i) lambda function will be used when the f^n is called only once.
(ii) lambda f^n does not have function name
(iii) mainly it is used to implement Higher order f^n
(iv) In most inbuilt f^n that takes f^n as Parameter, in those case lambda f^n is used
(v) it is used to make code precise & efficient.

ex (i) def add (x,y) => lambda x,y : x+y
retur (x+y)

(ii) def square (num) => lambda num : num*num
return(num*num)

{ note : input ( ) By default accept char }

code

```
ptr1 = lambda x,y : x+y
ptr2 = lambda num : num*num

sum = ptr1( 10,6)
square = ptr2 ( 6)

print (sum) -> 16
print (square) -> 36
```

* filtering :- filter()

Input element in list & then print them

```
l=[]
i=0
while (i<=4):
    num= int (input(" Enter no"))
    l.insert (i,num)  # insert in list
    i=i+1
```

to input list

```
i=0
while (i<=4):
    print (l[i])
    i=i+1
```

to print list

filter even no from list

```
def even (x)
    if (x%2==0):
        return True
    else:
        return False
```

fn

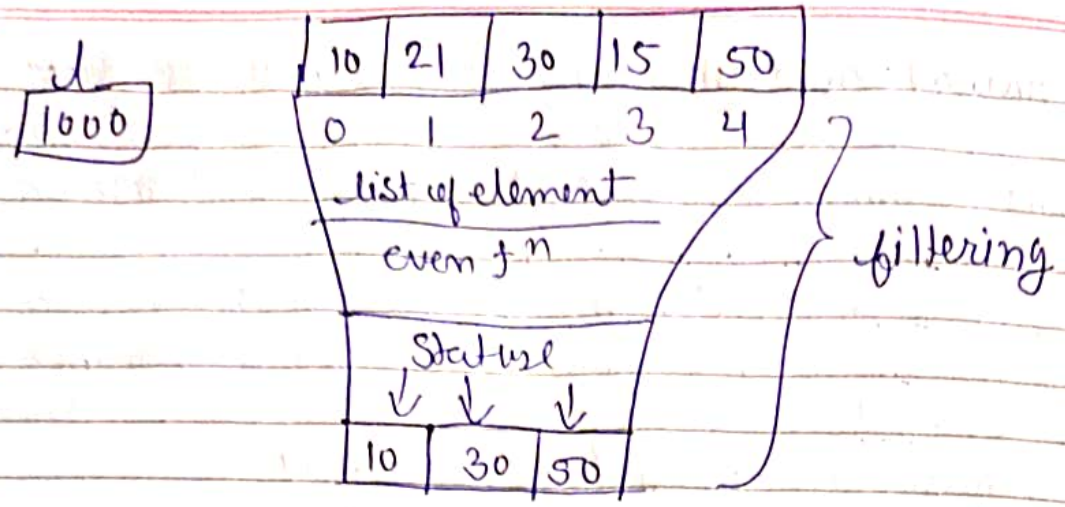
```
l=[]
i=0
while (i<=4)
    num= int(input("Enter no"))
    l.insert (l,num)
    i=i+1
```

to input list

```
i=0
while (i<=4)
    status = even(l[i])
    if(status==True)
        print(l[i])
    i=i+1
```

to filter list

so



ie on Basis of some condition, extracting only those element, who satisfy condition is known as filtering.

to do filtering in Python we have filter() f^n.
→ filter function takes two Parameter
(i) function
(ii) list of element

Syantax:

{ filter (function, list of element) }

so

Now we can replace filter section of code By this f^n

```
def even(x):
    if (x%2 == 0)
        return True
    else
        return False
```

f^n

```
l=[]
i=0
while (i<=4):
```

```
num = int (input ("Enter no: ")
l.insert (i, num)
i = i+1
```
} to input list

```
K= list (filter (even, l))
print (K)
```
} filter list

filter gives output as object so we convert this to list by list() & store it to K.

Now, we use even() f^n in whole program only once & by standard

★ { if we use a function in whole program only once, then function must be define as lambda f^n. }

So ★

input in list

```
l = [ ]
i = 0
while ( i <= 4):
        num = int (input ("Enter no "))
        l.insert ( i, num)
        i = i+1
print (l)    # printing list
```

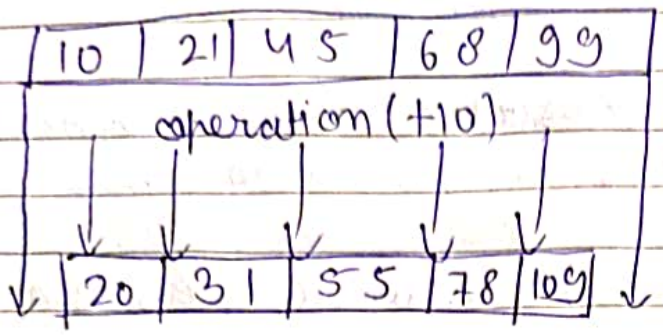
lambda fn used

```
# K = list (filter (lambda x : x % 2 == 0, l))
print (K)  # printing filterd list
```

* map function : when we perform an operation on all list of element of list, it is known as mapping

-> In this list is not reduced

ex



map function takes two Parameter
(i) function
(ii) list of element

syntax

```
{ map ( function, list ) }  # Higher order fn.
```

ex

```
def operation (num):
    return num+10
```

} fn

```
l=[]
i=0
while (i<=4):
    data= int(input ("Enterno"))
    l.insert (i, data)
    i=i+1
```

} input in list

```
i=0
while (i<=4):
    result = operation (l[i])
    print (result)
    i=i+1
```

} mapping & print mapped list

Now,
(i) we can use map f^n for mapping & printing mapped list.
(ii) operation f^n called only once, so it can be replaced by lambda f^n as standard

```
so
  i=0
  l=[]
  while (i<=4)
      data= int(input("Enter no"))
      l.insert (i, data)
      i=i+1
  K=list(map(lambda data : data+10, l))
  print(K)
```

input list

mapping & print

---

* Nested function :

if a function is call (invoke) inside its outer function only, then it must define inside outerfunction only.

=> the process of defining a function within another function is known as nested function.

ex

```
def fun1 ():
    def fun2()
        ...
    fun2()
fun1()
```

inner function

outer function

innerfunction calling

outerfunction calling

Code

```
def outerfunction():
        print(" inside outerfunction ")
    inner fn  def innerfunction():
                  print(" inside innerfunction ")
outer fn    innerfunction() #
            innerfunction() #
```

} calling innerfunction inside outerfunction

```
outerfunction()
outerfunction()
```

} outerfunction calling

```
innerfunction()
```

→ innerfunction call outside outer function

↳ error

so we can't access innerfunction outside the outerfunction, because it is out of scope

Code

```
def fun1()
    a=10
    def fun2():
        b=20
    10 ← print(a)
    20 ← print(b)
    fun2()
10 ← print(a)
error ← print(b) ×
```

} inner fn

} outer fn

```
fun1() # outer function, fun1() calling
```

→ we can't access (call) inner function directly outside the outer function

→ we can access indirectly

→ the variable of inner function is called local variable

→ the variable of outer function is called non local variable

& we know that local variable's scope only function in which it was declearad.
so we can't use local variable outside its scope ie outside its function...
if we try to access it will be out of scope & give error.

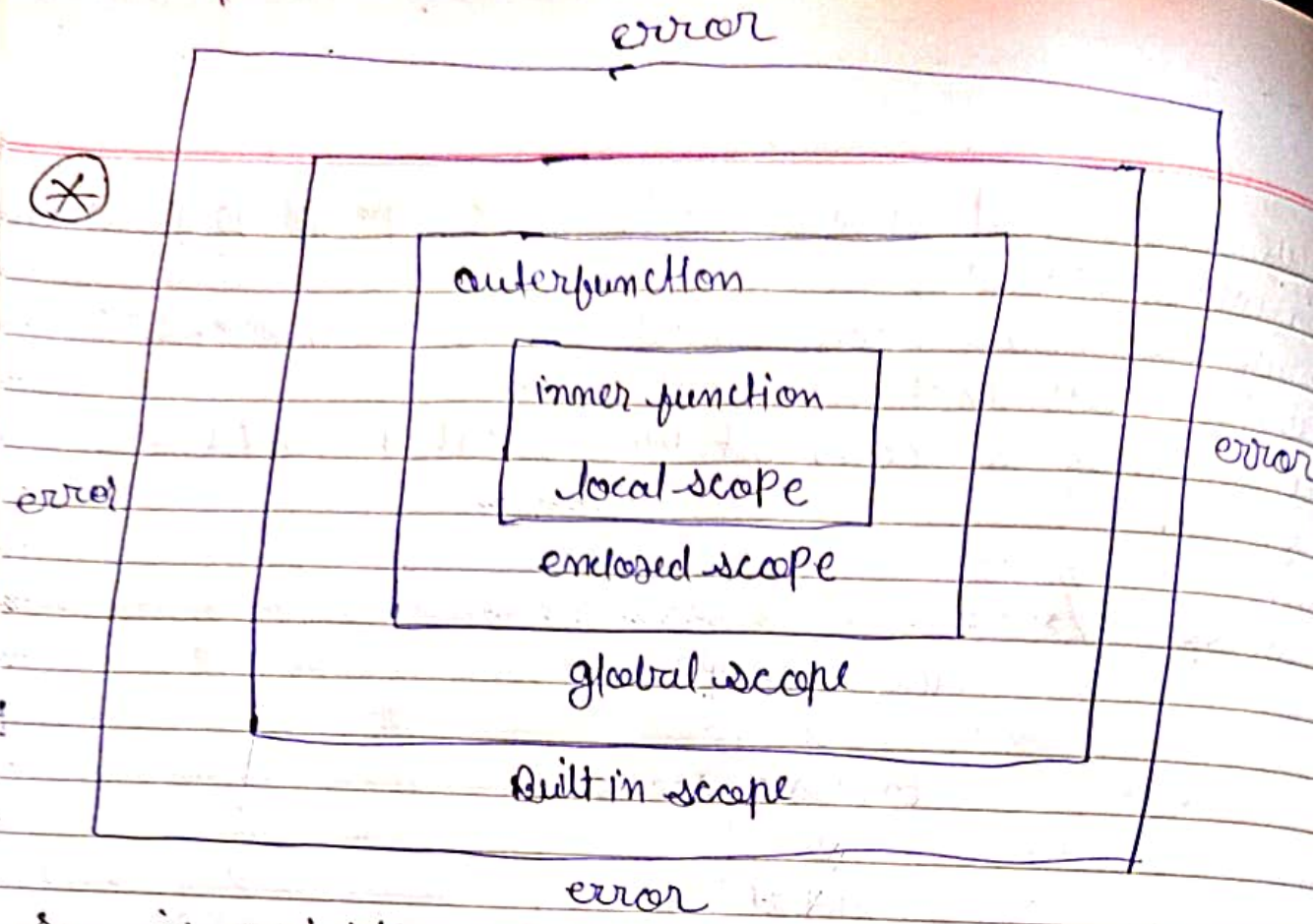
* Scope:

local scope
↓
Enclosed scope
↓
global scope
↓
Built in scope
↓
error

code

```
x=10
def fun1()
    x=99
    def fun2()
        x = 777
        print(x) -> 777
    fun2() # calling fun2

fun(1) # calling fun1
```

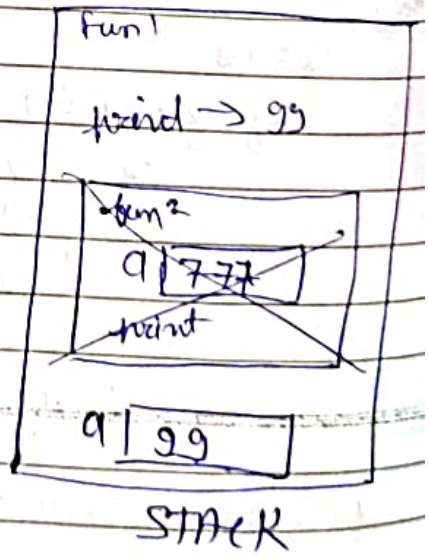
local scope
enclosed scope
global scope



So, if variable not found in Built in scope then Pvm will give error.

code How to use local variable out of scope

```
def fun1()
    a = 99
    def fun2()
        a = 777
        print(a) -> 777
    fun2()
    print(a) -> 99
fun1()
```



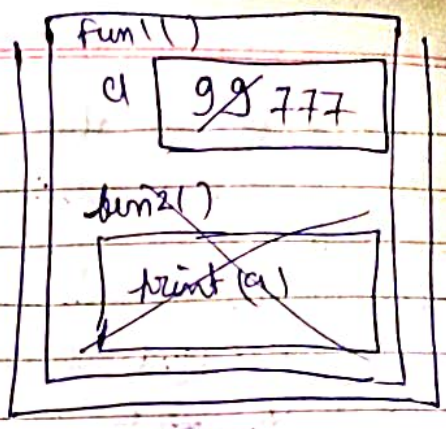
So a = 777 is local for fun2, so in fun2 when we print(a) we get 777

But when we are out from fun2, activation record of fun2 is deleted so Now a = 99 is local for fun1 Hence it print(a) => 99

Keyword: Nonlocal

same code

```
def fun1():
    a = 99
    def fun2()
        nonlocal a
        a = 777
        print(a) → 777
    fun2()
    print(a)  → 777
fun1()
```



STACK

Here nonlocal work similar to global variable when PVM see we created non local variable inside inner function, then it will not create a new variable & points to existing nonlocal variable (if any present).

* nested variable:

```
a = 666
def fun1():
    global a   # No New copy created
    a = 777
    b = 10
    print(a) -> 777
    print(b) -> 10
    def fun2():
        global a   # No New copy of a created
        nonlocal b   # No New copy of b created
        a = 888
        b = 20
        c = 30
        print(a) -> 888
        print(b) -> 20
        print(c)  30
    fun2()
fun1()
print(a) -> 888
```

Global scope
a = ~~666~~ ~~777~~ 888

nonlocal / enclosed scope
b = ~~10~~ 20

local scope
c = 30

* Modules & closures :-

=> Anything which extention is .Py is known as module

=> module provide facility of code reusability

* creating & importing module :

=> module are created as same as we write Python program

=> But we have 2 ways of importing them.

pgm1.Py

Ex1:
```
def fun1():
    print("I am inside fun1 of program1")
```

module created

Now, importing module

mypgm.Py
```
from pgm1 import fun1
fun1()  # module fn invoke
```
* from & import is Keyword

EX.2 if we have multiple fn in module then

Pgm2.Py
```
def fun1():
    print("inside fun1 of program 2")
def fun2():
    print("inside fun2 of program 2")
```

Now, importing

my pgm2.Py
```
from pgm2 import fun1, fun2
fun1()
fun2()
```

⇒ Here, we imported two module with function
But some module have thousands of fn
it is impossible ~~for~~ for us import them by their name.

Pgm3.Py

EX3

```
def fun1():
    print("Inside fun1 of program 3")
def fun2()
    print("inside fun2 of program 3")
    :
def fun100().
    print("inside fun100 of program 3")
```

Now, importing

MyPgm3.Py

```
from Pgm3 import *
fun1()                    # * import all fn
fun2()                      of module
  :
fun100()
```

* Second way of importing module

EX: 4

Pgm4.Py

```
def fun1():
    print("inside fun1 of pgm4")
def fun2():
    print("inside fun2 of pgm4")
```

MyPgm4

```
import Pgm4
Pgm4.fun1()
Pgm4.fun2()

# In this way we directly import whole
module, But when we call fn then
we must need to use module name
Pgm4.fun1()
```

module name ← Pgm4 ; fun1() → function name

mod1.py

Ex5:

```
def fun1():
    print(" inside fun1 of mod1")
def fun2():
    print(" inside fun2 of mod2")
```

mod2.py

```
def fun3():
    print(" inside fun3 of mod2")
def fun4():
    print(" inside fun4 of mod2")
```

my Pgm5.Py

```
from mod1 import *
from mod2 import *
fun1()                    # WAY 1
fun2()
fun3()
fun4()
```

my Pgm6.Py

```
import mod1
import mod2
mod1.fun1()               # WAY 2
mod1.fun2()
mod2.fun3()
mod2.fun4()
```

⇒ these are the 2 method two import modules.

* If two diffirent module have function with same name

mod1.py

```
def fun1()
    print(" inside fun1 of mod1 "
def fun2()
    print(" inside fun2 of mod1 ")
```

mod3.py

```
def fun1():
    print(" inside fun1 of mod3")
def fun2()
    print(" inside fun2 of mod3")
```

MyPgm7.py

```
import mod1
import mod3
mod1.fun1()        # No proble,
mod1.fun2()          No Ambigity
mod3.fun1()
mod3.fun2()
```

MyPgm8.py

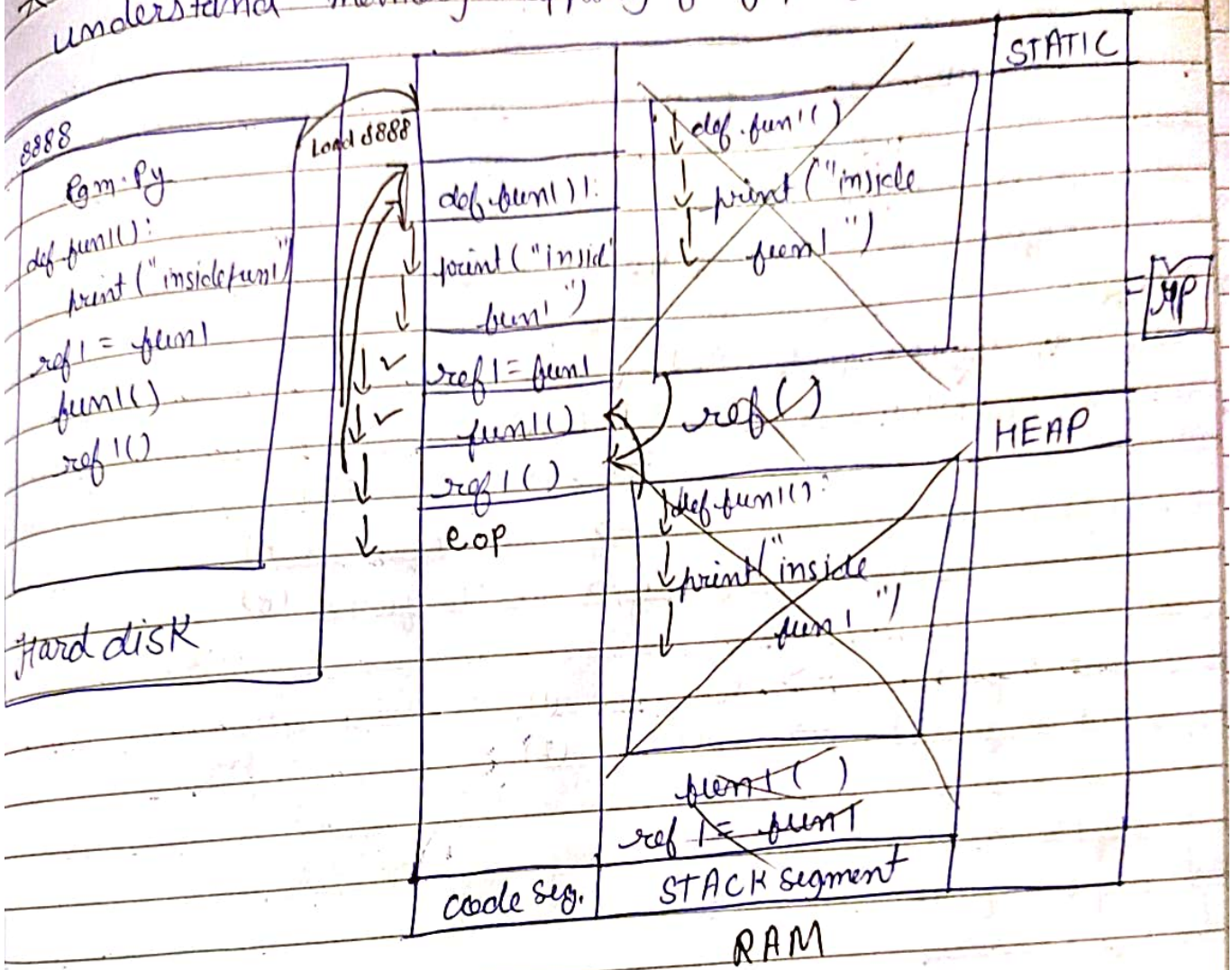
```
from mod1 import *
from mod3 import * ✓

fun1()   } # confusion, which module
fun2()   }                fun1, fun2
```

⇒ once you import modules into a program, inside module if function have same name, then the module which is imported at last, that's function are invoked (executed)

⇒ in 2 module, same f^n name, the last imported module function is executed.

* Clousers:

before understanding closure, firstly understand memory mapping of Py program



(i) code loaded in code segment

(ii) Now control scan whole code & find first LOC to execute & put it in stack

(iii) Control execute the statement & go to next statement

(iv) if we face function or method then control shift to fn

(v) when last statement of Program (ie EOP) is executed then program deallocate all memory.

11/1/22

recall function as Parameter

1000

```
def fun1():
    print(" inside fun1 ")
```

2000

```
def fun2():
    print(" inside fun2 ")
```

3000 ★

```
def display(ptr1, ptr2):
    print(ptr1)      → 1000        (5)
    print(ptr2)      → 2000        (6)
    ptr1()           → inside fun1 (7)
    ptr2()           → inside fun2 (8)
```

start

```
print(fun1)   → 1000   (1)
print(fun2)   → 2000   (2)

ref1 = fun1
ref2 = fun2

print(ref1)   → 1000   (3)
print(ref2)   → 2000   (4)
★
display1(fun1, fun2)

    fun1()    → inside fun1   (9)
    fun2()    → inside fun2   (10)
    ref1()    → inside fun1   (11)
    ref2()    → inside fun2   (12)
```

Now if

```
def outer ()
    print (" inside outer ")          → ①
    def inner ()
        print (" inside inner ")      → ②
    inner ()
```

out
outer()

But if:

```
8000
def outer ()
    print (" inside outer ")
9000 def inner ()
        print (" inside inner ")
    return inner        print (inner) {→ 9000}
```

```
ref1 = outer ()
ref1 ()
print (ref1) {→ 9000}
```

By returning address of inner f^n inner f^n called outside the outer f^n ie Beyond its Scope ie called as "Closure."

So, Closure: it is a function object that remembers value in enclosing scope even if they are not present in memory.

it allows the function to access those captured variable through the closures' copies of their value & references, even when ~~they are~~ function is invoked outside their Scope.

* when & why use closure

(i) closure are used as callback function, they provide some sort of data ~~hiding~~ hiding, this help us to reduce use of global variable.

(ii) when we have few f^n in our code, closure prove be an efficient way, But if we need to have many f^n the go for classes (OOP).

#

```
(i) def outer():                    # nested function
        print("inside outer")
        def inner():
            print("inside inner")
        inner()
    outer()
```

```
# def outer()
        print("inside outer")        # closure
        def inner():
            print("inside inner")
        return inner
ref1 = outer()
ref1()
```

```
# def outer(function):              # Decorator.
        print("inside outer")
        def inner():
            print("inside inner")
        return inner

ref1 = outer(function)
ref1()
```

* Decorator :

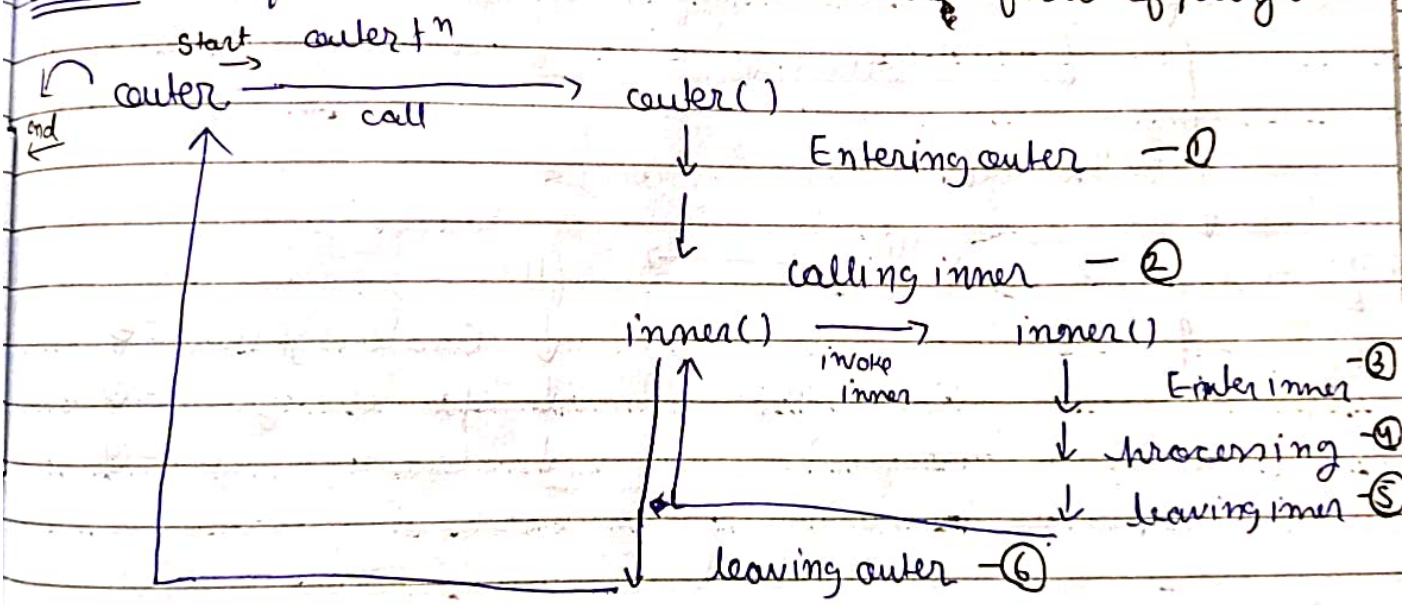
#1
```
def outer()
    print("Entering outer")
    def inner():
        print("Entering inner")
        print("processing")
        print("leaving inner")
    print("calling inner")
    inner()
    print("leaving outer")
```

Nested f^n

outer()

Execution Sequence :

control flow of program



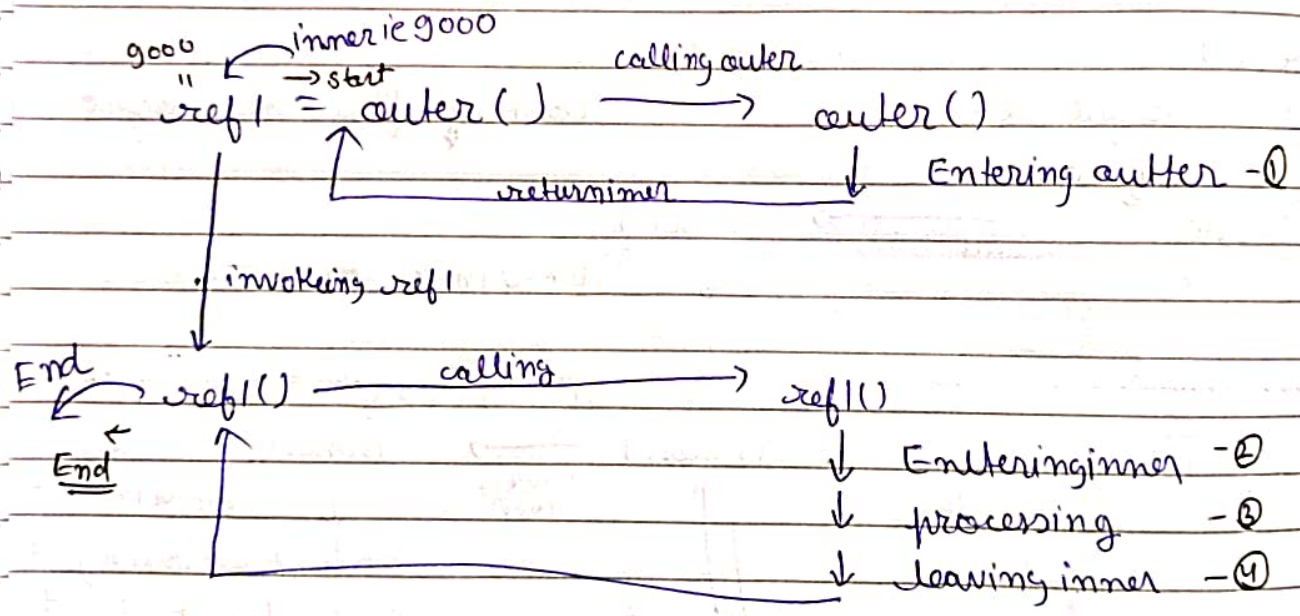
O/P
```
Entering outer
calling inner
Entering inner
processing
leaving inner
leaving outer.
```

Closure

#2 8000

```
def outer()
    print("Entering outer")
    def inner()
        print("Entering inner")
        print("processing")
        print("leaving inner")
    return inner
```

```
ref1 = outer()
ref1()
```



O/P
Entering outer
Entering inner
processing
leaving inner

Decorator

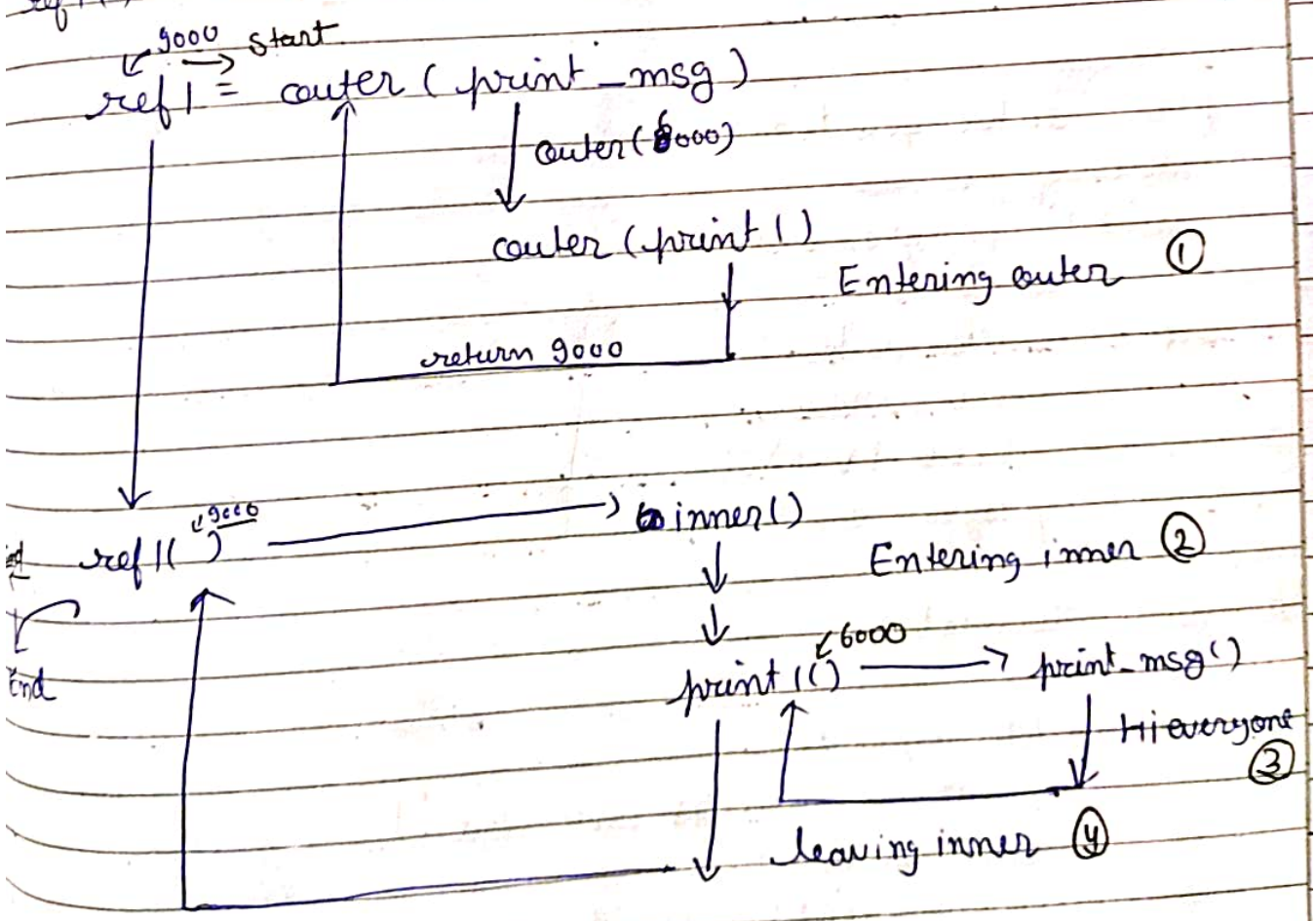
#3 ~~9000~~ 6000

```
def print_msg()
    print ("Hi Everyone")
```

6000

```
def outer (print()):
    print (" Entering outer")
    9000 ~~print()~~ def inner():
        print ("~~leaving~~ Entering inner")
            → print()
            → print("Leaving inner")
    return inner
```

```
ref1 = outer(print_msg)
ref1()
```



12/1/22 without decorator

#4

9000

```
def print_msg():
    msg = "Hello everyone"
    return msg
```

7000

```
def outer (print1) :
    print ("inside outer")
    8000 def inner()
            print ("inside inner")
            ref1 = print1
            msg = ref1()
            newstr = msg.upper()
            print (newstr)
    return inner
```

```
ptr1 = outer(print_msg)
ptr1()
print ("program End")
```

O/P
```
inside outer
inside inner
HELLO EVERYONE
program ends
```

#5

* with decorator

```
def outer (print1) :
    print ('address called')
    def inner ():
        print ('inside inner')
        ref1 = print1
        msg1 = ref1()
        newstr = msg1.upper()
        print (newstr)
    return inner
```

decorator

```
@outer
def print_msg():
    msg = 'Hellow world'
    return msg
```

note: the input f^n write after decorator.

note: decorator must write first.

calling 1

```
ptr1 = outer(print_msg)
ptr1()
```

calling 2

```
print_msg()
```

#6

```
def outer(print1):
    def inner():
        ref1 = print1
        msg1 = ref1()
        newstr = msg1.upper()
        print(newstr)
    return inner
```

```
@outer
def morning():
    msg = 'good morning'
    return msg
```

```
@outer
def evening():
    msg = 'good evening'
    return msg
```

```
@outer
def afternoon():
    msg = 'goodafternoon'
    return msg
```

```
@outer
def night():
    msg = 'good night'
    return msg
```

```
morning()
afternoon()
```

```
evening()
night()
```

* function overloading

{ In Python function overloding is not supported

```
def fun():
    print("fun has no Parameter")
```

Not supported

```
def fun(a):
    print("fun has 1 Parameter")

def fun(a,b):
    print("fun has 2 Parameter")
```

```
def fun():
    print("inside first fun")

def fun():
    print("inside 2nd fun")

def fun():
    print("inside 3rd fun")
```

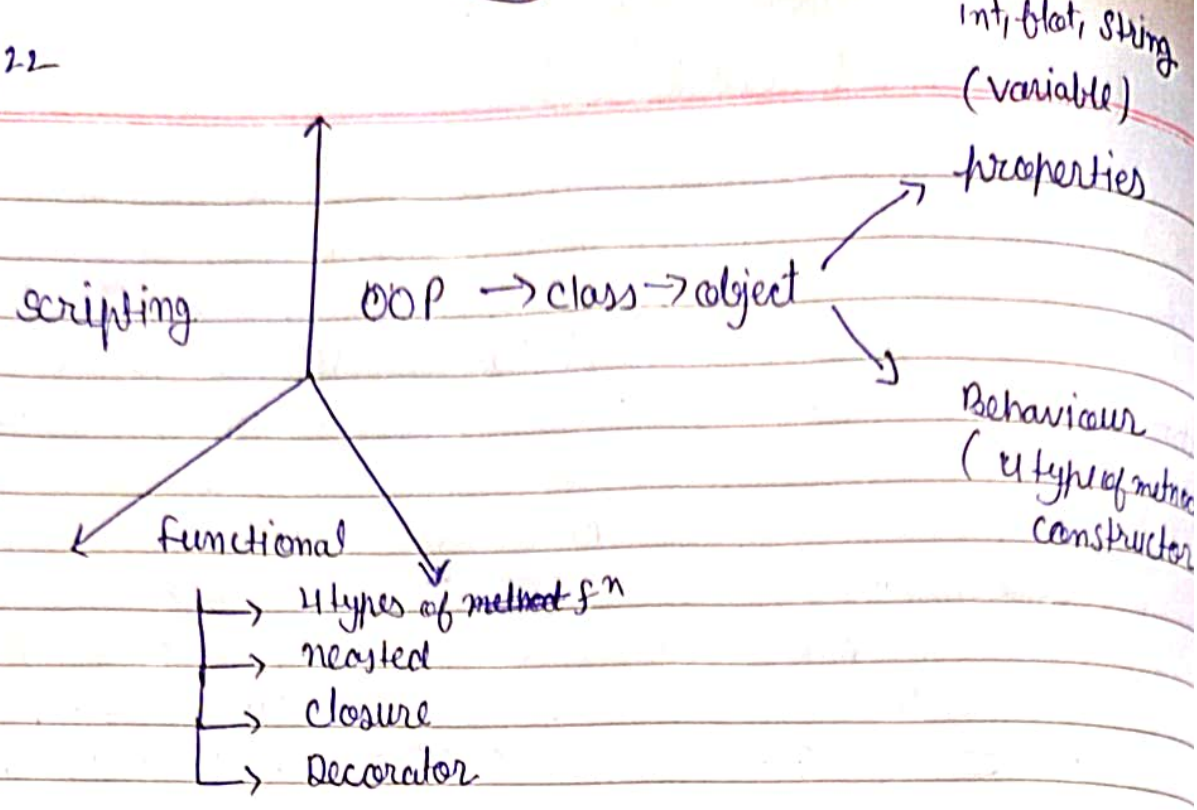
fun() → inside 3rd fun

⇒ if we have many f^n with same name, the last is executed, when we call f^n.

Recap type of variable

```
a=10                          } global variable
def fun1()
    b=20                      } local variable

def outer()
    x=100                     } Non local variable
    def inner()
        y = 200               } local variable
    inner()

outer()
class Ball
    d=100                     } Static variable
    def __init__(self):
        self.a=20             } object / instance variable

    def Bounce()
        z=99                  } local variable

c1 = ~~class~~ Ball()         } address variable
```

13/1/22



Pillars of object oriented programming

variable "Data"
Methods

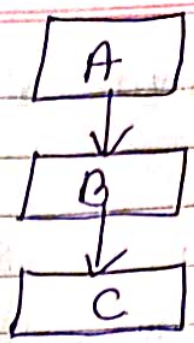
→ to find relation b/w these 2 we use pillars
ie encapsulation
inherence
Polymorphism

1 Encapsulation :- it is providing protection to the data from outside Access.

ex
AC → coolent
coconut → water
Pen → ink
Human → Brain / Heart

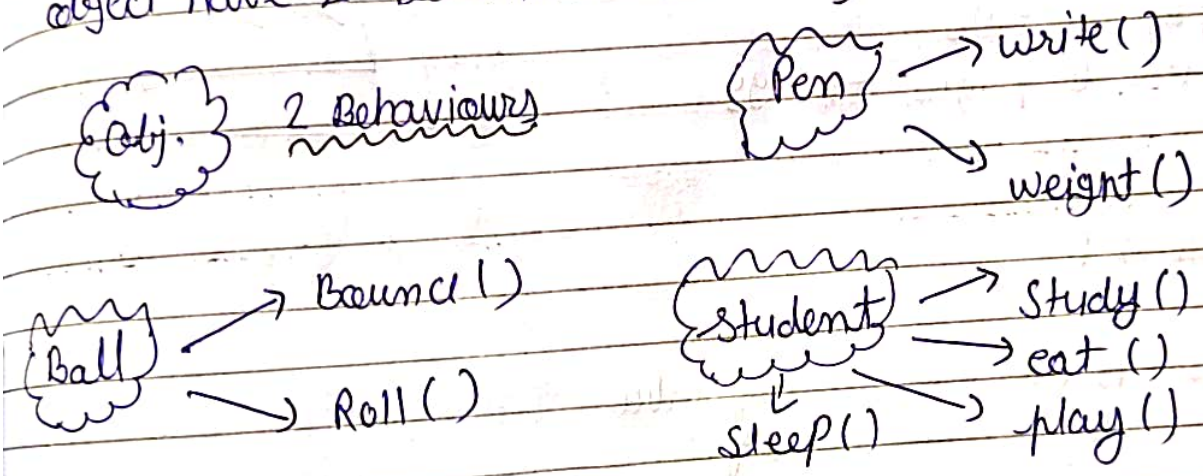
when there is some sensitive data to protect we use encapsulation.

#2 Inhertence : to get something from previous ~~Hierarchy~~ Hierarchy.
ie these are linked to each other in Hierarchy

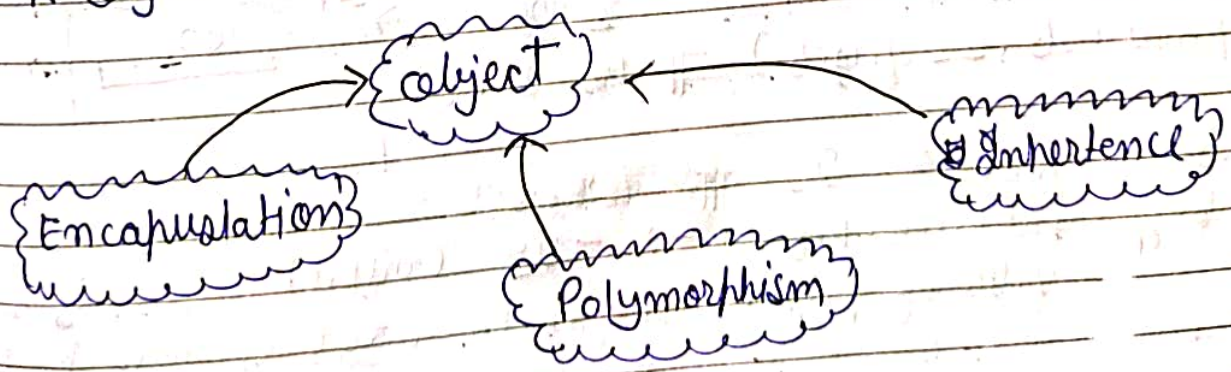


#3 Polymorphism :-

→ An object must have at least 2 Behaviour. if object have 2 Behaviour then only it is called as object.



⊛ A object ~~consist~~ Stand upon all these 3 pillars



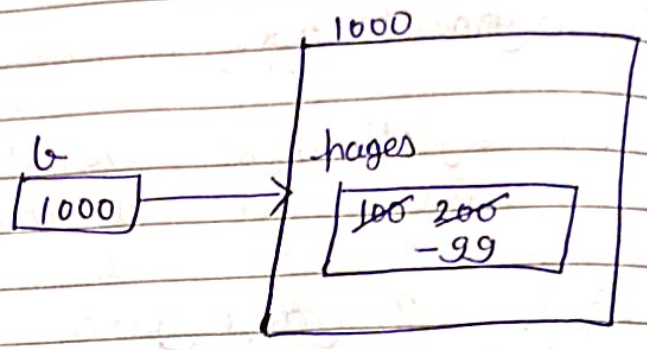
* Accessing private data

~~def~~ case 1:

```
Class Book:
    def __init__(self):
        self.Pages = value
```

self, value

```
b = Book(100)
print(b.pages) → 100
b.pages = 200
print(b.pages) → 200
b.pages = -99
print(b.pages) → -99
```

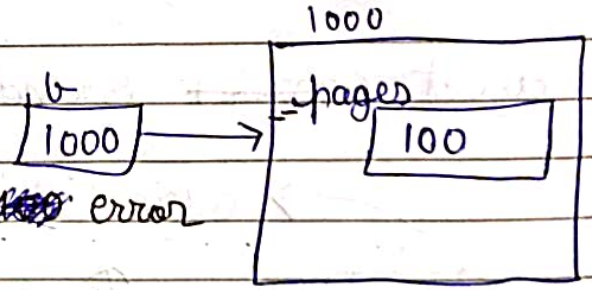


case 2:

```
Class Book:
    def __init__(self):
        self.__pages = value
```

self, value

```
b = Book(100)
print(b.__pages)  # error
b.__pages = -99
print(b.__Pages) → -99 (output)
```



value of private data member does't change & can't access from outside

we get error as attribute error.

So when we run this print(b.__pages) → Error

But

Second Statement print(b.__Pages) → -99

ie it is a different copy

case 3:

```
Class Book:
    def __init__(self, value):
        self.__Pages = value

    def display()
        print(self.__Pages)

b = Book(100)
print(b.__Pages)        → error
b.__pages = -99
print(b.__Pages)        → -99 (copy)
b.display()             → 100 (current value)
```

__Pages is private data so it can't Access outside class.

of __Pages

* Accessing ~~protected data~~ private data By class method

#1

```
Class Book
    def __init__(self, value):
        self.__Pages = value

    def getdata(self):
        return self.__Pages

    def Setdata(self, value)
        if (value >= 1)
            self.__Page = Value

b = Book(100)
print(b.getdata)        → 100
b.setdata(200)
print(b.getdata)        → 200
b.setdata(-99)
print(b.getdata)        → 200
```

getter

setter

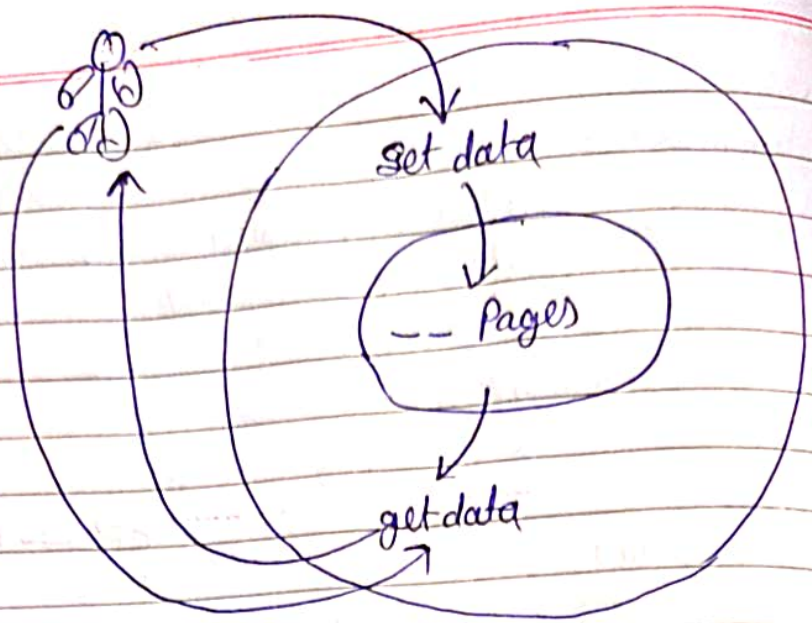
So, Now we can Access private data By getdata & Setdata methods

~~they gives use~~

this is called As "Data Binding"

Because they give us "Controlled Access"

so



Now we have two different f^n for data reading & writing
But in Python we use Hundreds of function or methods
so can we minimize this. Yes

2 # By property

```
Class person:
    def __init__(self):
        self.__name = " "

    def getter (self):
        return self.__name

    def setter (self, values):
        self.__name = value

    getset = property (getter, setter)

P1 = person()
p1. getset = " Ramu"   # setter (getset in left)
result = p1. getset    # getter (getset in right)
print (result)
```

By using prope
we can use getter &
setter at same
time.

so Same name can do both getter & setter for
method invokation

14/1/22

#3 Accessing private data By decorator

```
Class Person
    def __init__(self):
        self.__name = ""
```

getter & setter are always in same sequence
(i) getter
(ii) setter

```
① property
    def data Access(self):
        return self.__name

② ~~data Access Person~~ property or data Access.setter
    def data Access(self, value)
        self.__name = value

P1 = Person()
P1.data Access = "Ramu"
result = P1.data Access
print(result)
```

* Access Specifier :

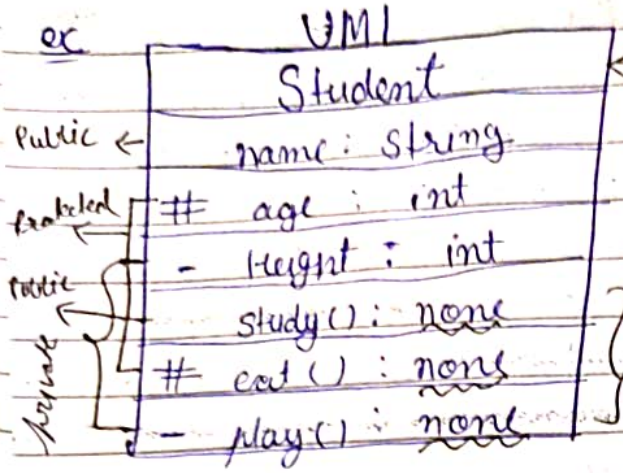
(i) public
(ii) protected
(iii) private

} they indicate level of protection of data in class.

* UML → (Unified modular language): it is intended to provide a standard way to visualize the design of a system.
so it have information of class, we code According to UML

ex

UML



← S is capital so it is name of class
Blank represent public
__ represent private
represent protected

} method return none (no return)

so Class will be

```
class Student:
    def __init__(self):
        self.name = "Arjun"      # Public
        self._age = 23           # protected
        self.__Height = 10       # private

    def study(self):
        ____
        ____                     } # public

    def _eat(self):
        ____
        ____                     } # protected

    def __play(self):
        ____
        ____                     } # private
```

* Inheritance :

time ↓ money make ↑

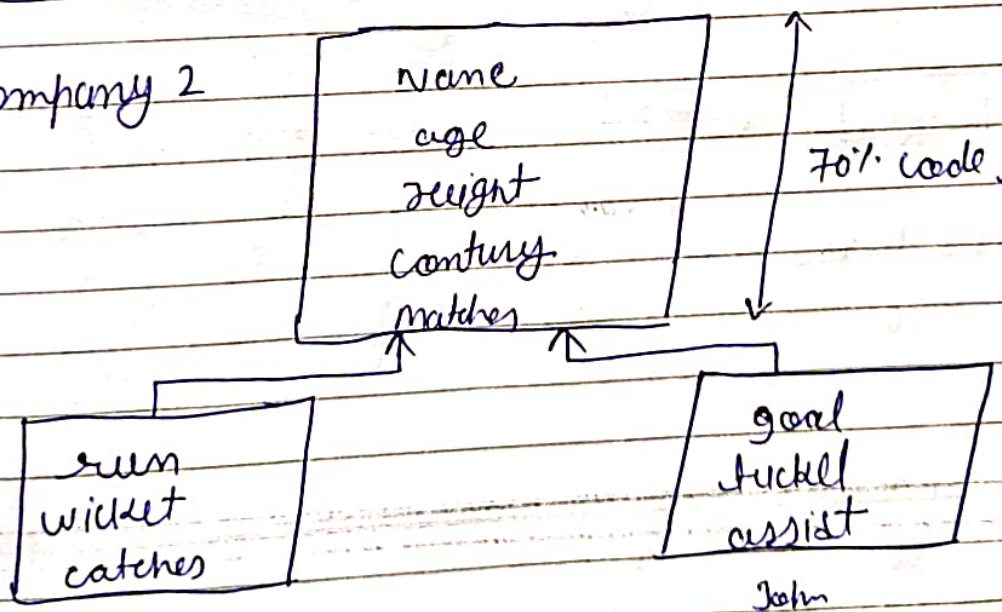
Inheritance (Reuse) it helps to ↑es the profitability of company.

ex 1

Company 1 (Cricket)	Company 1 (FootBall)
Name age Height country matches run wicket catches	Name age Height contry matches goal tackel assist
RR	John
6 month , cost = 1 cr	6 month , 1 cr

ex 2

Company 2



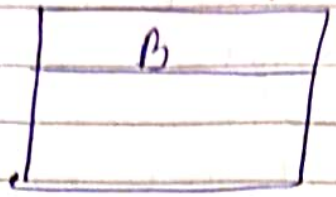
RR
6 month, 1 cr

John
3 month , 3 cr
time ↓es, money ↑es

Now,

Class A:

Class B:



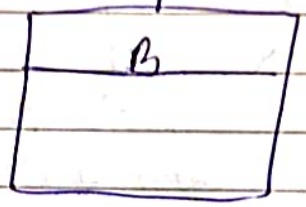
unrelated class

So, syntax

Class A :

A is parent, super, Base class

Class B(A) :



B is child, sub, derived class

#1 Code

```
Class Parent:
    def __init__(self):
        self.a = 10

class child(Parent):
    def __init__(self):
        self.b = 20

C1 = child()
print(C1.b)  → 20
print(C1.a)  → error (Attribute error)
```

But why? ↳ 'child' object has not attribute 'a'.

☆☆

Constructor of Parent class will not get called automatically
- its a duty of programer to call constructor of Parent class Manually

so, you must invoke constructor of parent class within child class, to access instance variable of Parent class by object of child class.

#2

```
                                  # self = 1000
class Parent :                                          1000
    def __init__(self) : ↓④                  ③
        self.a = 10 ↓⑤                              a | 10
                                                    b | 20

class child(Parent) :
                                  #self = 1000      ⑥
    def __init__(self) :
①       ② ↓ Parent.__init__(self)                   # constructor of
            ⑦ ↓ self.b = 20                           Parent class callin
                                                      child class
                                  ⑧
C1 = child
⑨ ↓ print (C1.b)  →  20   } Now it doest give erro
⑩ ↓ print (C1.a)  →  10
```

end

3 "Related Class"

```
Class A:                    1000
    def __init__(self)
        ⑥↓ self.a = 10

Class B:

    def __init__(self):     1000
        ④↓ A.__init__(self):   1000
            ⑧↓ self.b = 20

Class C:

    def __init__(self):     1000
        ②↓ B.__init__(self)    1000
            ⑩↓ self.c = 30

start
0↓ C1 = C()
12↓ print (C1.c)  → 30
13↓ print (C1.b)  → 20
14↓ print (C1.a)  → 10
```

1000

a	10
b	20
c	30

17/1/22

```
class cargoplane:
    def takeoff(self):
        print("plane took off")
    def fly(self):
        print("plane flying")
    def land(self):
        print("plane has landed")
    def carry(self):
        print("carrying cargo")
```

```
class passenger:
    def takeoff(self):
        print("plane took off")
    def fly(self):
        print("plane flying")
    def land(self):
        print("plane has landed"
    def carry passenger(self):
        print("carrying Passenger"
```

```
class fighter:
    def takeoff(self):
        print("plane took off")
    def fly(self):
        print("plane flying")
    def land(self):
        print("plane has landed")
    def carryweapon(self):
        print("carrying weapon")
```

Here 3 class have 3 ~~function~~ method method takeoff, fly, land. These 3 ~~function~~ have similar behaviour in class

```
c = cargoplane()
c.takeoff()
c.fly()
c.land()
c.carry()
```

```
P = Passenger()
P.takeoff()
P.fly()
P.carryPassenger()
P.land()
```

```
f = fighter()
f.takeoff()
f.fly()
f.carryweapon()
f.land()
```

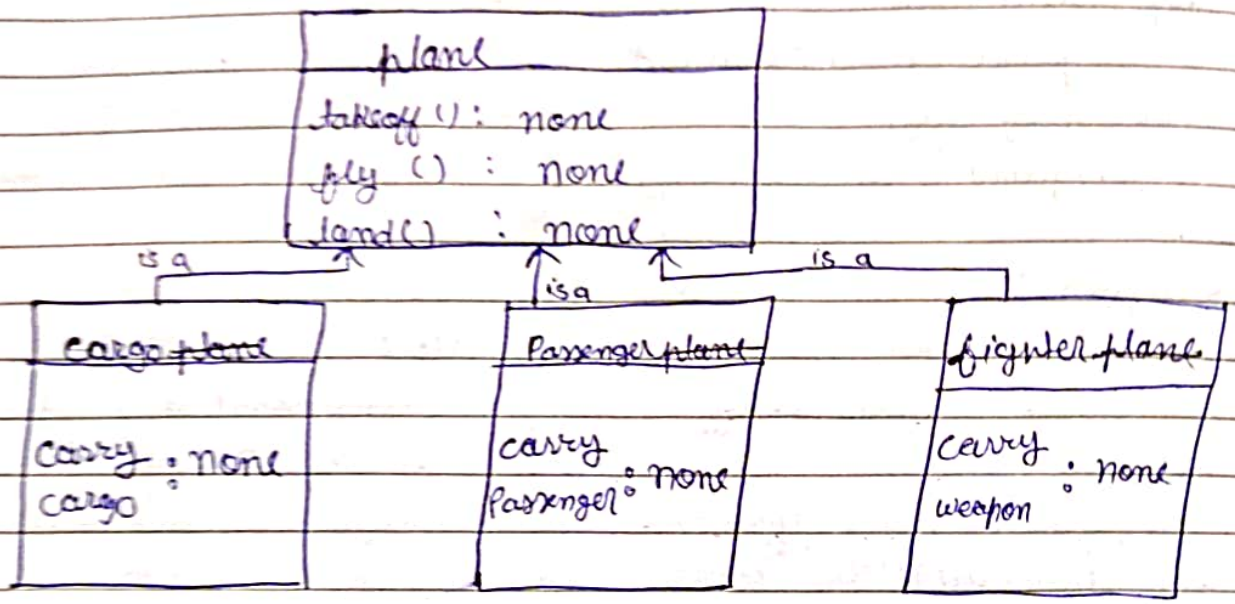
takeoff, fly, land 3 method are defined many time & called many time, But it is not efficitive way to define classes.

for 3 class we invoke 12 method & define 12 method

×

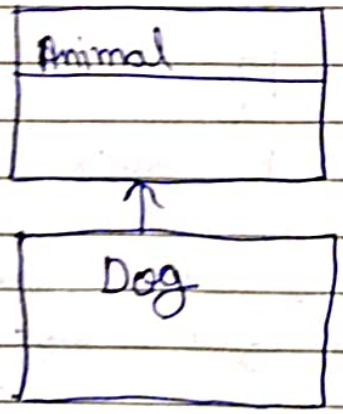
So, How to declear & define effective classes.

So cargo plane
Passenger plane
fighter plane } take off, fly, land } Common Behaviour (plane)



"is a" relationship

cargo plane is a plane
Passenger plane is a plane
fighter plane is a plane } cohesive meaning



Not repeate Base class name again & again in child class

it is not standard.

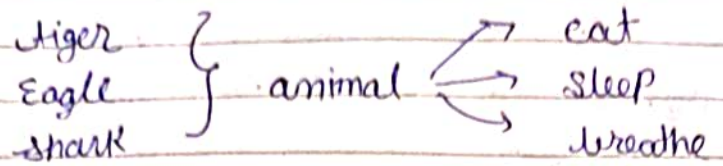
#

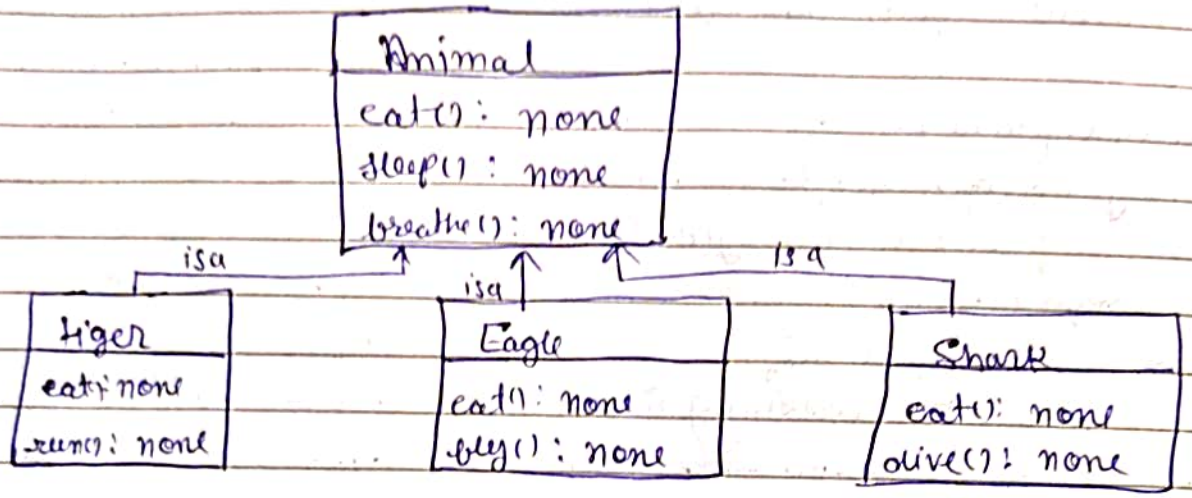
```
Class plane:
    def takeoff(self):
        print("plane took off")
    def fly(self):
        print("plane flying")
    def land(self):
        print("plane landed")

class cargo(plane):
    def cargo(self):
        print("carrying cargo")
class Passenger(plane):
    def Passenger(self):
        print("carrying Passenger")
class fighter(plane):
    def weapon(self):
        print("carrying weapon")
```

c1= cargo()	P1= Passenger()	f1=fighter()
c1.takeoff()	P1.takeoff()	f1.takeoff()
c1.fly()	P1.fly()	f1.fly()
c1.cargo()	P1.Passenger()	f1.weapon()
c1.land()	P1.land()	f1.land()

so we have 3 class 12 ~~four~~ method invoke, 6 method defination
so By using inheritance method defination are reduced.

* method overriding : take something from Parent & change it in child class, ie known as overriding.






```
class Animal :
    def eat (self):
        print (" Animal is eating ")
    def sleep(self):
        print (" Animal is sleeping ")
    def breathe (self):
        print(" Animal is breathing")
```

} Base class method

```
Class shark (Animal):
    def eat (self):
        print (" shark is swimming & eating ")
    def dive (self):
        print (" shark is diving")
```

} overridden method

} specialized method

```
class tiger (Animal):
    def eat (self):
        print (" tiger is running & eating ")
```

} overridden method

```
def run (self):
    print(" Tiger is running")
```
} specialized method

```
Class Eagle (Animal):
    def eat (self):
        print ("eagle is flying & eating ")
```
} overridden method

```
    def fly (self):
        print (" Eagle is flying ")
```
} specialized method

```
t = tiger()
e = eagle()
s = shark()
```

t.eat()	e.eat()	s.eat()
t.sleep()	e.sleep()	s.sleep()
t.breathe()	e.fly()	s.breathe()
t.run()	e.breathe()	s.dive()

so

eat() is overridden method
sleep, breathe() are inherited method
dive(), fly(), run() are specialized method

specialized method : these method belongs to class, in which they defined, they can't be invoked by other class object.

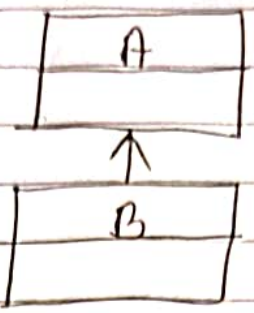
18/1/22

* Types of Inheritence

(i) Single Inheritence

Class A :

class B(A) :

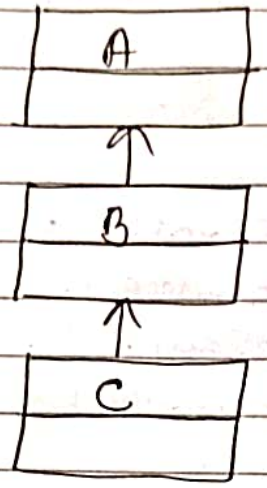


(ii) multilevel Inheritence

Class A :

class B(A) :

class C (B) :

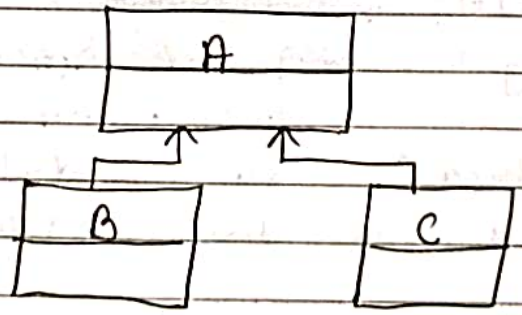


(iii) Heirarchical inheritance

Class A :

Class B(A) :

Class C(A) :

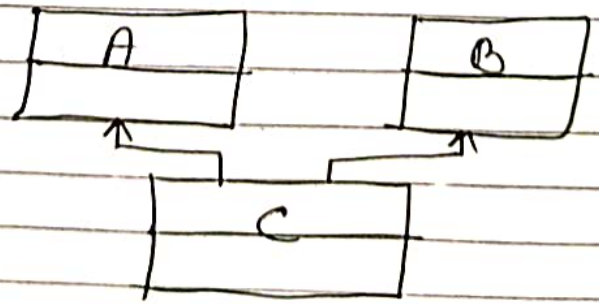


(iv) multiple Inheritence

Class A :

class B :

Class C(A,B) :



#1

```
Class A:
    def dispA(self):
        print("inside A")

Class B:
    def disp B (self):
        print(" inside B")

Class C (A,B):
    def dispC (self):
        print(" inside C")
```

```
C1 = C()
C1.dispA() -> inside A
C1.dispB() -> inside B
C1.dispC() -> inside C
```

#2

```
Class A:
    def display (self):
        print(" inside A")
class B:
    def display (self)
        print(" inside B")
class C (A,B):
    def display (self)
        print("inside C")
```

```
C1 = C()
C1.display() -> inside C
C1.display() -> inside C
C1.display() -> inside C
```

#3

```
Class A:
    def disp (self):
        print(" inside A")
Class B:
    def disp (self):
        print(" inside B")
Class C (A,B):
    def display (self):
        print(" inside C")
```

```
C1 = C()
C1.display() -> inside C
C1.disp() -> inside A
```

```
# Class C (B,A)
```

then

```
C1.display() -> inside C
C1.disp() -> inside B
```

So, if we have 2 method with same name in Parent class then the method of class, which is inherited first is invoke

when we ~~inherit~~ inherit ~~m~~ multiple classes, then there is always a possibility of name clash of methods to avoid this, check method name in this Heirarchy

great grand Parent class
↑
grand Parent class
↑
Parent class
↑
current class

⇒ MRO Algo / C3 Algo. are used to reduce name clash.

* Polymorphism :

```
Class plane:
    def takeoff (self):
        print (" plane took off ")
    def fly (self):
        print (" plane fly ")
    def land(self):
        print (" plane landed ")

Class cargo(plane):
    def takeoff(self):
        print (" cargo plane took off ")
    def fly (self):
        print (" cargo fly ")
    def land (self)
        print (" cargo land ")
```

```
Class Passenger (plane):
    def takeoff (self):
        print (" passenger took off")
    def fly (self):
        print (" passenger fly")
    def land (self):
        print (" passenger land")
```

⇒ Here method name is same But implemention is diff.

```
c = cargo()        c.takeoff()   c.fly()   c.land()
p = Passenger      p.takeoff()   p.fly()   p.land()
```

Here class design is correct But invocation of method can be optimized so,
we define a function

```
def allowplane (ref):
    ref.takeoff()
    ref.fly()
    ref.land()
```

} 3 method are called inside function

```
1000
c = cargo ()
2000
p = Passenger ()

                1000
allowplane (c)
allowplane (p) 2000
```

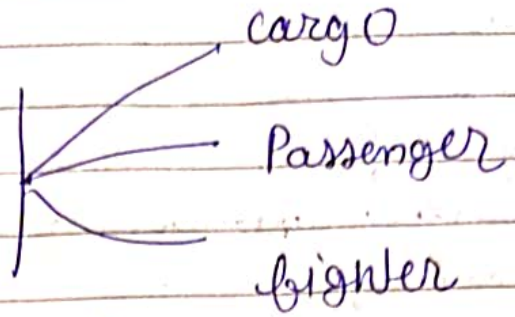
} function call

ie is called "POLYMORPHISM"

Here allowplane function can be used to call all method of classes, ie same function, for different input, give different output.

Here

allowplane(ref):

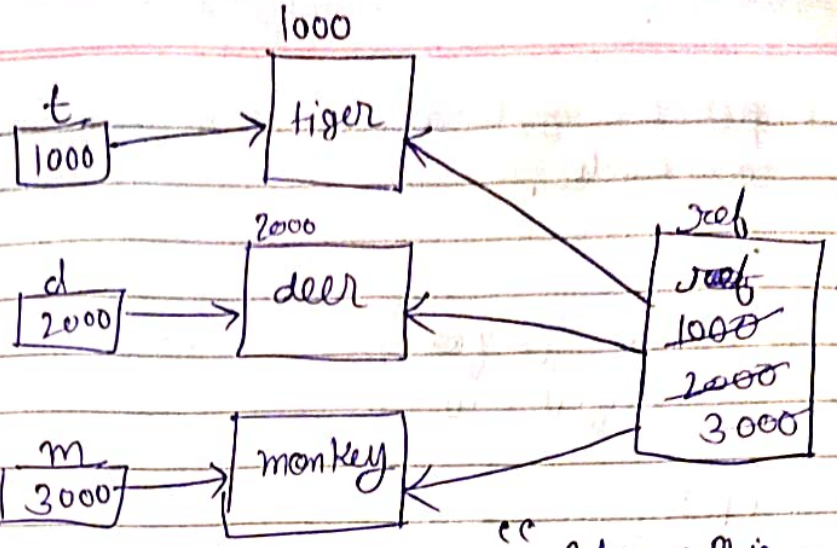


ie 1 : many relationship

So, when an entity exibit diff output on diffirent input ie known as Polymorphism

ex 2
```
Class tiger:
    def eat (self):
        print (" tiger is eating
    def sleep(self):
        print(" tiger is sleeping ")

Class Deer:
    def eat(self):
        print ("deer is eating ")
    def sleep(self):
        print (" deer is sleeping ")

Class monkey:
    def eat (self):
        print (" monkey is eating ")

    def sleep (self):
        print (" monkey is sleeping ")

def allowAnimal(ref):
    ref.eat()
    ref.sleep()
```
} Polymorphic f^n

```
t = tiger()
d = deer()
m = monkey()

allowAnimal(t)
allowAnimal(d)
allowAnimal(m)
```



"Polymorphism"

(*) Invoke a method of a Parent class within a child class

```
Class A
    def display(self):
        print(" inside A ")

Class B(A):
    def display(self):
        print(" inside B")

Class C(B):
    def display(self):
        print(" inside C ")

Class D(C):
    def disp(self):
        A.display(self)
        B.display(self)
        C.display(self)

d = D()
d.disp()
```

* Super () method : it is used to call Parent constructor.

```
Class A :
    def __init__(self)
        self.a = 10

Class B(A) :
    def __init__(self) :
        super().__init__() or A.__init__(self)
        self.b = 20

Class C (B) :
    def __init__(self) :
        super().__init__()
        self.c = 30

C1 = C()
print (C1.c)  → 30
print (C1.b)  → 20
print (C1.a)  → 10
```

So, As we know to use variable of Base class in derived Class, we must call constructor of Base class in derived class, for this

A.__init__(self) ✓ A is Base class name

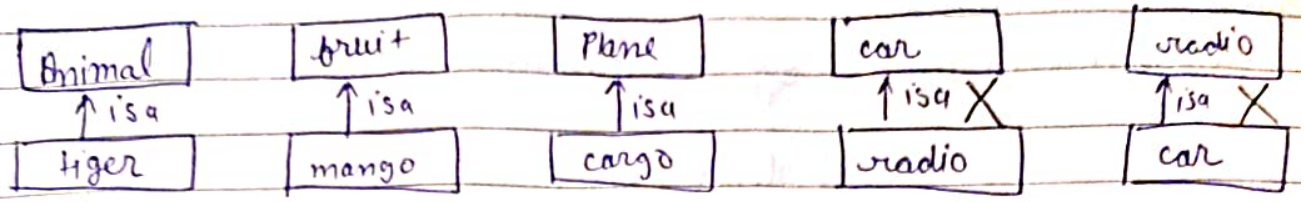
We can also use super() method to call construtor of Base class in derived class.

super().__init__()

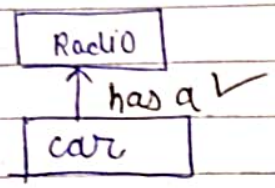
In this we do not need to specify Base class name & we don't give self as parameter for __init__()

19/1/22

* delegation Model: it says create an object inside the main Class to establish " has a " relation



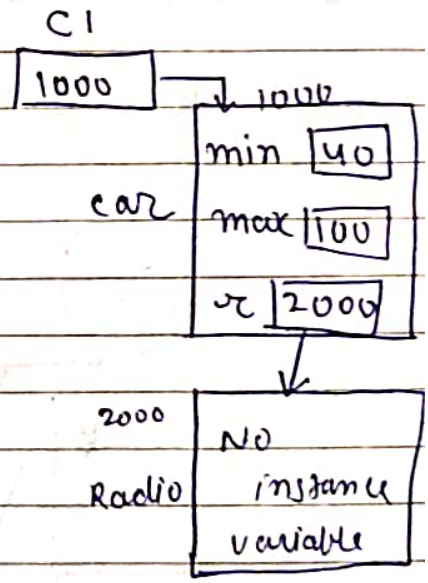
But



so car " has a " radio relation it means inside car we have radio.

So to establish "has a" relation we create an object of Radio inside car (main) class.

Code delegation_model_1




```
Class Radio:
    def turnon(self,x):
        if (x==1):
            print("Radio is on")
        else:
            print("Radio is off")

Class car:
    def __init__(self,min,max):
        self.min = min
        self.max = max
        self.r = Radio()        # object of Radio inside constructor of car.

c1 = car(40,100)
print(c1.min) -> 40
print(c1.max) -> 100
c1.r.turnon(1) -> Radio is on
c1.r.turnon(0) -> Radio is off
```

So, having an object inside another object, to establish " has a " relation is known as "Delegation Model".

A "has a" relation can be establish into class by delegation

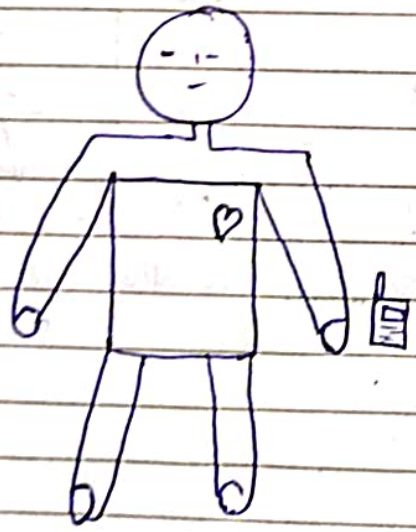
delegation consist of Composition & Aggrigation

Delegation
- Composition → composed object
- Aggrigation → Aggrigate object

* composed object: An object that cannot exist without main object, is known as composed object

* Aggrigate object: An object that can exist without main object, is known as Aggrigate object.

ex 1

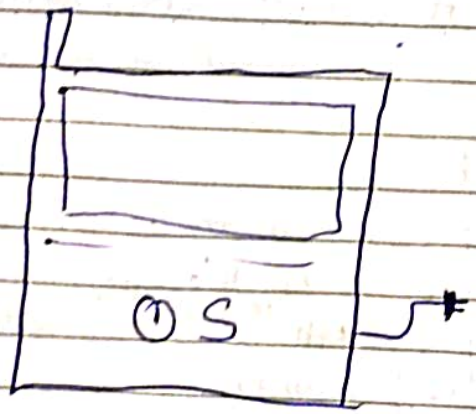


Heart ← has a — Human

mobile ← has a — Human

Heart → composed object
mobile → aggrigate object
Human → main object

ex



OS ← has a — mobile

Charger ← has a — mobile

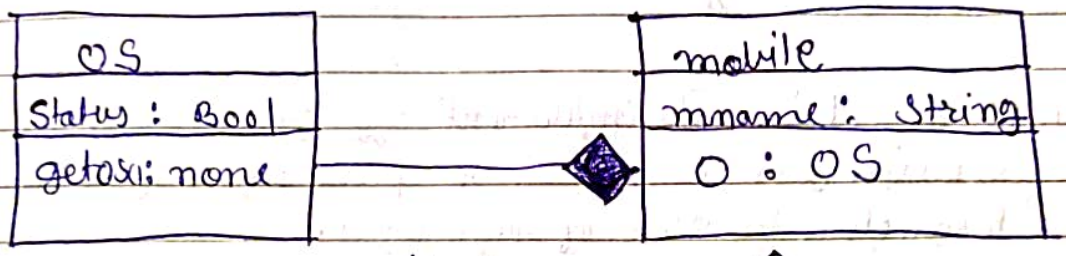
OS → composed obj
Charger → aggrigate obj
mobile → main obj.

⊗ In order to implement a composed object we must create the composed object as a instance variable within the class of main object

* Composed object

⇒ UML (OS - mobile), "has a" relⁿ, composed obj

To represent "has a" relation, we use diamond
To represent it is composed object, we make solid diamond
To represent it is aggrigate object, we make Hollow diamond



so mobile has a OS ◇
OS is composed obj ◆

```
class OS :
    def __init__(self):
        self.status = True
        print ("OS is installing")
    def getOS(self):
        print ("OS is still executing")

class Mobile :
    def __init__(self):
        self.mname = "mj"
        self.O = OS()                    # composed obj
        print("mobile created")
        print ("OS installed")
m1 = mobile()
print(m1.mname)          → mj
print(m1.O.Status)       → OS is installing, mobile created
m1.O.getOS()             → OS installed, OS is still executing
```

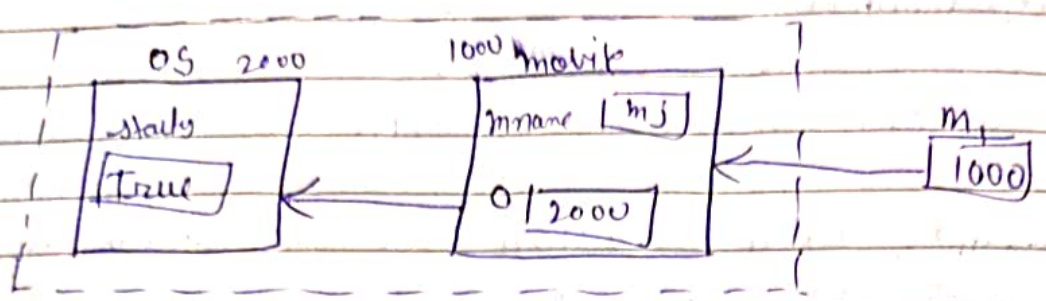
del is a Keyword to delete

if

```
del m1
print(m1.o.status)
m1.o.getOS()
```

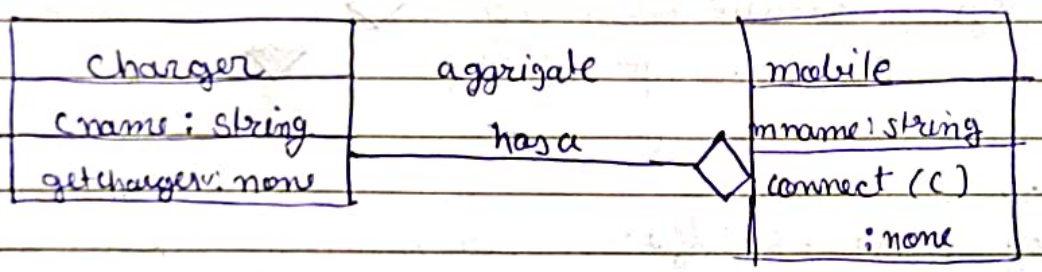
} nameerror, m1 is not defined

why?



so if m1 is ~~def~~ deleted then o is automatically deleted

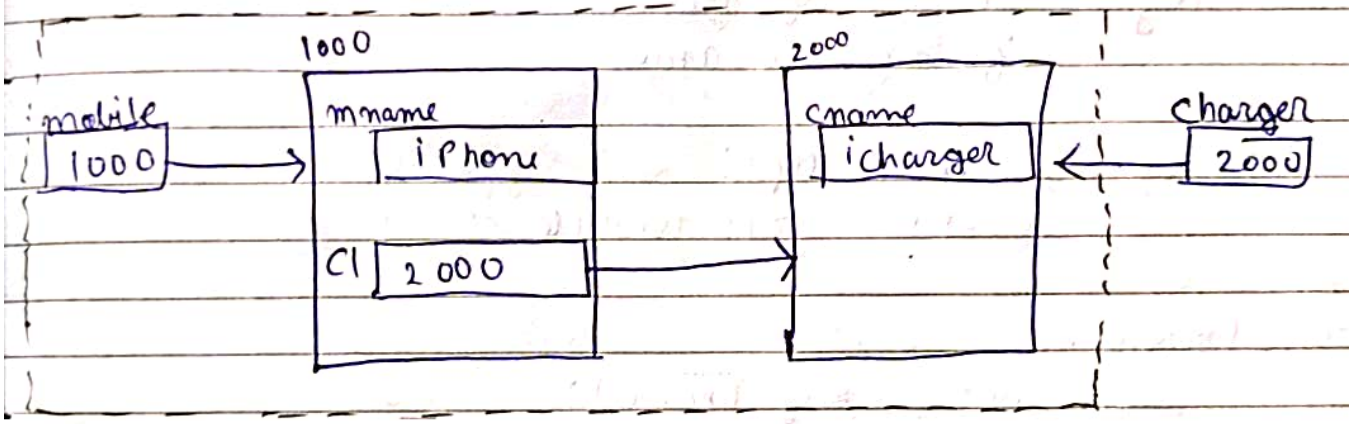
* aggrigate object: To implement aggrigate object ie "has a" relation b/w classes, we pass object of child class as Parameter to class of main object.




```
class charger:
    def __init__(self, cname):
        self.cname = cname
        print("charger is ready")

    def getcharger(self):
        print("charger can be used for charging

class mobile:
    def __init__(self, mname):
        self.mname = mname
        self.c = " "                    # to store object
        print("mobile created")
```

↙ object
2000

```
def connect(self, c1):
    self.c1 = c1
    print("mobile is connected to charger")

mobile = Mobile("iPhone")
charger = charger("icharger")
mobile.connect(charger)        # object Passed to main class
print(mobile.mname)
print(mobile.c1.cname)         -> 1000.2000.cname -> icharger
mobile.c1.getcharger()         -> 1000.2000.getcharger() ->
del mobile                     # deleting mobile object
print("after deleting main obj")
print(charger.cname)           -> 2000.cname
print charger.getcharger()     -> 2000.getcharger
```

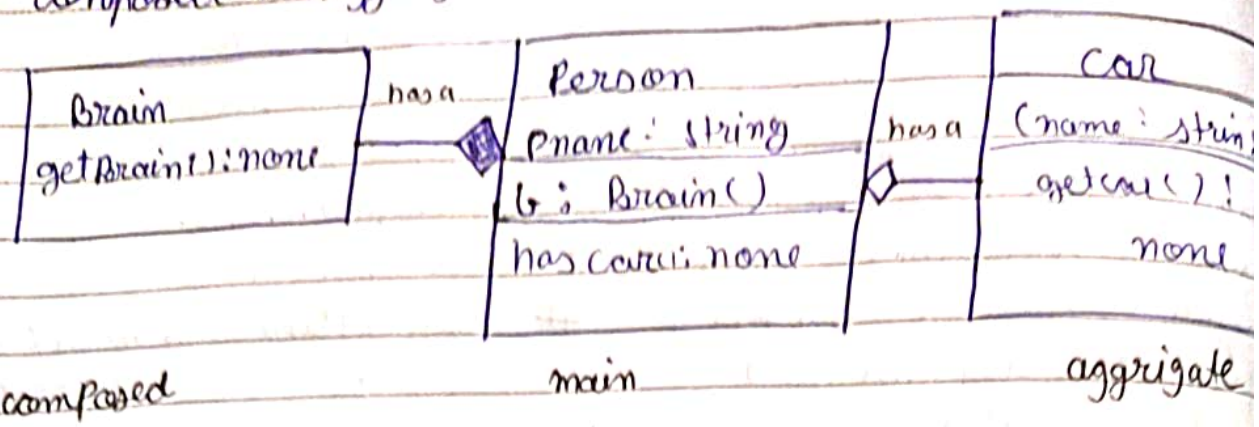


O/P

mobile created
charger is ready
mobile is connected to charger
iPhone
icharger
charger can be used for charging
after deleting main obj
icharger
charger can be used for charging

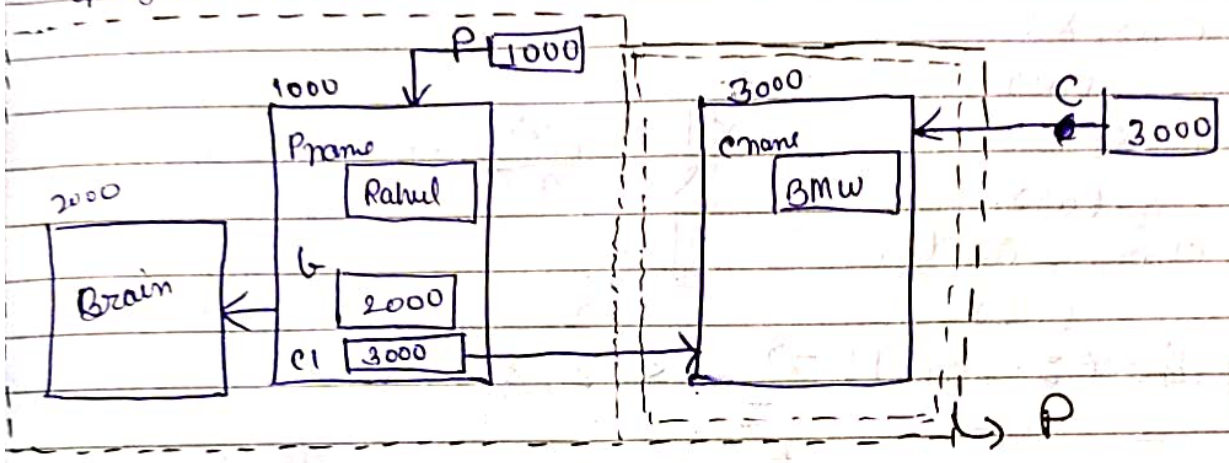
#=3

⊛ composed & aggrigate




```
class Brain:
    def getbrain(self):
        print("Brain is still working")

class car:
    def __init__(self, cname):
        self.cname = cname

    def get car(self):
        print("car is available")

class Person:
    def __init__(self, Pname):
        self.Pname = Pname
        self.b = Brain()          # composed
        print("person ready")
        print("Brain ready")
        self.c1 = " "             # to store aggrigate object

    def has car(self, c1)
        self.c1 = c1              # aggrigate object passed
        print("person purchased the car")
```

P = Person ("Rahul") → constructor of Person invoke
C = car ("BMW") → constructor of car invoke
P. hascar(c) → car object pass to Person
Print (P.Pname) → Rahul
P.b. getbrain() → Brain is working
print (P.c1.cname) → BMW
P.c1. ~~get~~ hascar() → person purchased car
del P → object P deleted
print (C.cname) → BMW
C. getcar() → person purchased car
P.getbrain() → error



* MRO : method Resolution order :

→ In Python if there is a class with no Parent then object class is Parent of that Class

→ the process of searching for a method in class Hierarchy of Parent child ie known as MRO

or

→ the order in which is looked for a method in Parent-Child chain is called as MRO

{ [class].mro() → order of MRO }

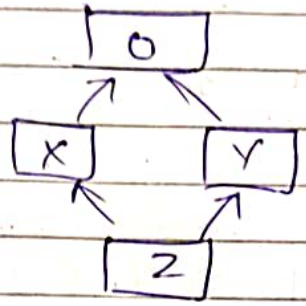
Q1 class A :

```
    def dispA(self):
```

```
a = A()
a.dispA()
print (A.mro()) → <A, object>

Class B(A):
    pass
print (B.mro()) → <B, A, O>
```

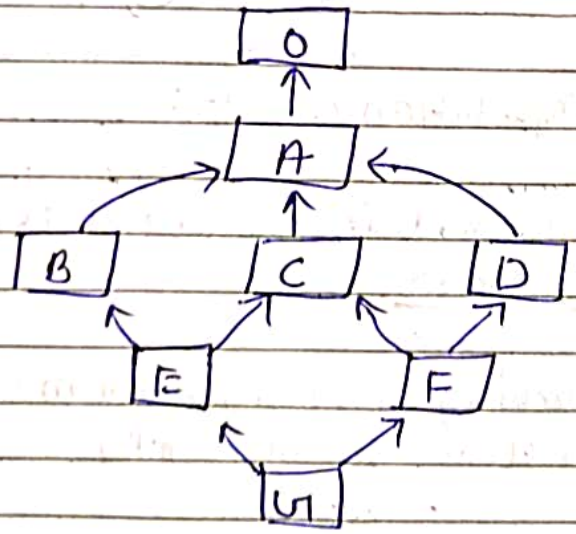
Q2

```
Class X:
    pass
Class Y
    pass
Class Z(X,Y):
    pass
print (Z.mro()) → <Z, X, Y, O>
```



← object

Q1



L0
L1
L2
L3
L4

MRO(A) → <A, O>
MRO(B) → <B, A, O>
MRO(C) → <C, A, O>
MRO(D) → <D, A, O>
MRO(E) → <E, B, C, A, O>
MRO(F) → <F, C, D, A, O>

MRO(G) → Now we use C3 Algo. because after level 3 it is little complex.

MRO(G) → G + C3Algo(MRO(E), MRO(F), EF)

→ G + C3Algo($\underset{H}{E}\ \underset{T}{BCAO}$, $\underset{H}{F}\ \underset{T}{CDAO}$, $\underset{H}{E}\ \underset{T}{F}$

{ if we found first Parmeter H in any Parameter tail then, jump to next Parameter
if we not fear first Parameter H in, any Parameter T, then add H to solution & remove H from all Parameters.
}

⇒ G + C3Algo ($\underset{H}{E}\ \underset{T}{BCAO}$, $\underset{H}{F}\ \underset{T}{CDAO}$, $\underset{H}{E}\ \underset{T}{F}$)

so

GE + C3Algo ($\underset{H}{B}\ \underset{T}{CAO}$, $\underset{H}{F}\ \underset{T}{CDAO}$, $\underset{H}{F}$

so

GEB + C3Algo ($\underset{x}{C}\ \underset{T}{AO}$, $\underset{H}{F}\ \underset{T}{CDAO}$, $\underset{H}{F}$) ×

so

GEBF + C3Algo ($\underset{H}{C}\ \underset{T}{AO}$, $\underset{H}{C}\ \underset{T}{DAO}$,)

so

GEBFC + C3Algo ($\underset{H}{A}\ \underset{T}{O}$, $\underset{H}{D}\ \underset{T}{AO}$) ×

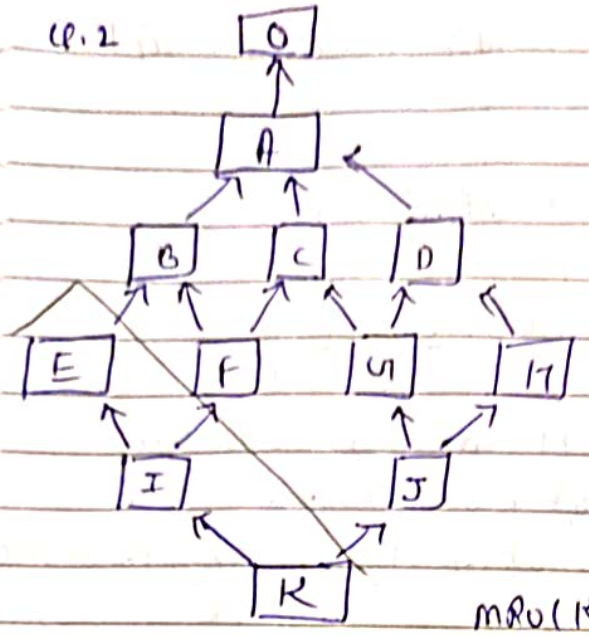
so

GEBFCD + C3Algo (AO, AO) ✓

so GEBFCDAO A

21/1/22

Q.2



MRO(A) → A, O
MRO(B) → B, A, O
MRO(C) → C, A, O
MRO(D) → D, A, O
MRO(E) → E, B, A, O
MRO(F) → F, B, C, A, O
MRO(G) → G, C, D, A, O
MRO(H) → H, D, A, O
MRO(I) → I E F B C A O
MRO(J) → J G C H D A O
MRO(K) → K I E F B J G C H D A O

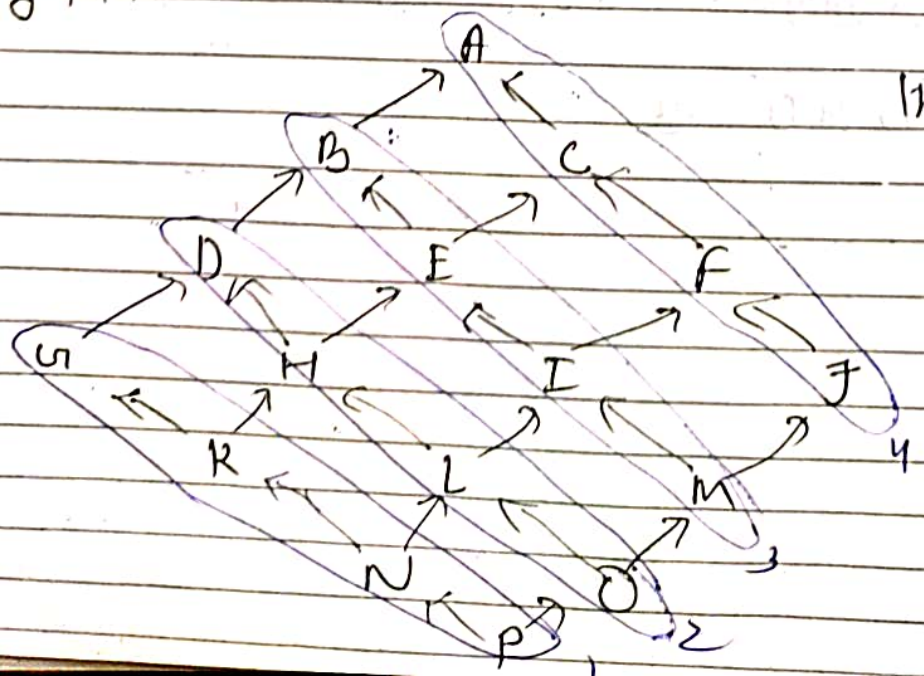
K → K + C3Algo(I E F B C A O, J G C H D A O, I J
K → K I + C3Algo(E F B C A O, J G C H D A O, J
K → K I E + C3Algo(F B C A O, J G C H D A O, J
K → K I E F + C3Algo(B C A O, J G C H D A O, J)
K → K I E F B + C3Algo(C A O, J G C H D A O, J)

K → K I E F B J + C3Algo(C A O, G C H D A O,)

K → K I E F B J G + C3Algo(C A O, C H D A O)
K → K I E F B J G C + C3Algo(A O, H D A O)

⇒ K I E F B J G C H D A O.

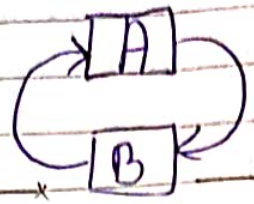
My trick



then MRO(P).

P N K G O L H D
M I E B J F
C A O

* Cyclic Inheritance



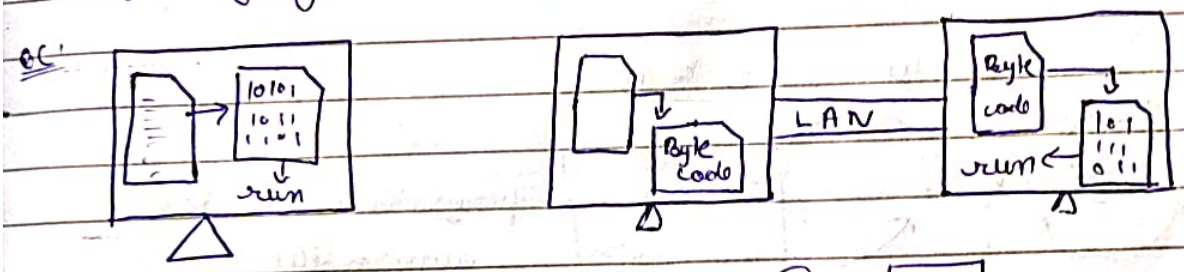
In Python cyclic Inheritance not supported

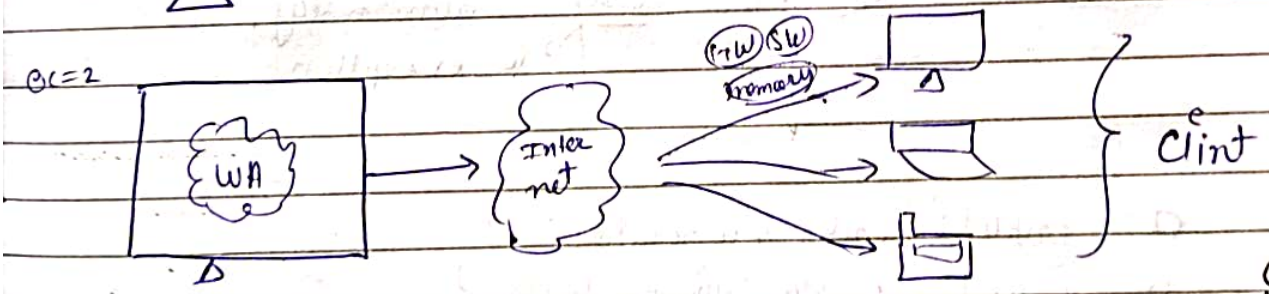
* Exception Handling

→ Exception means error that occur during execution (Runtime)

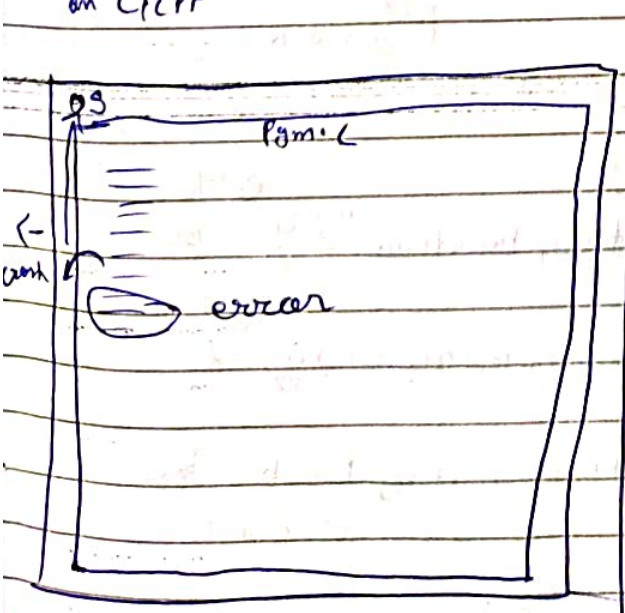
→ Exception Handling is a mechanism that is used to Handle the exception

→ why we handle Exception: if we have error during execution the program should end normally without damaging Hardware, software, memory of client System

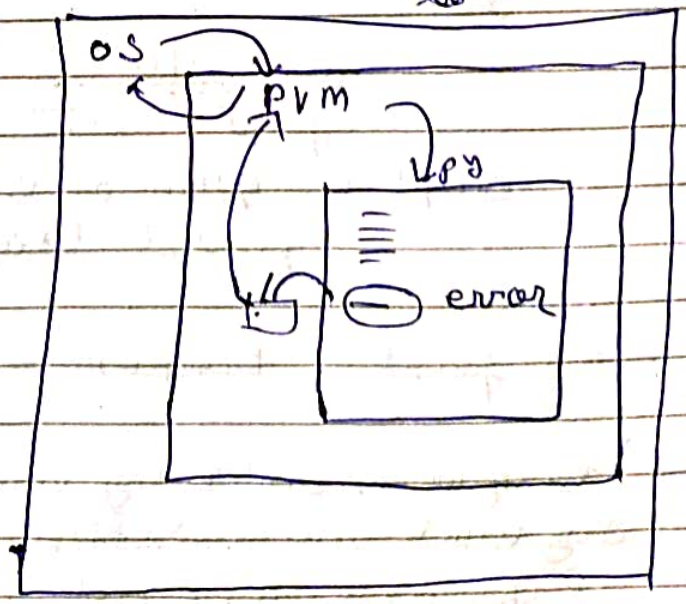




In C/CPP



→ No system crash

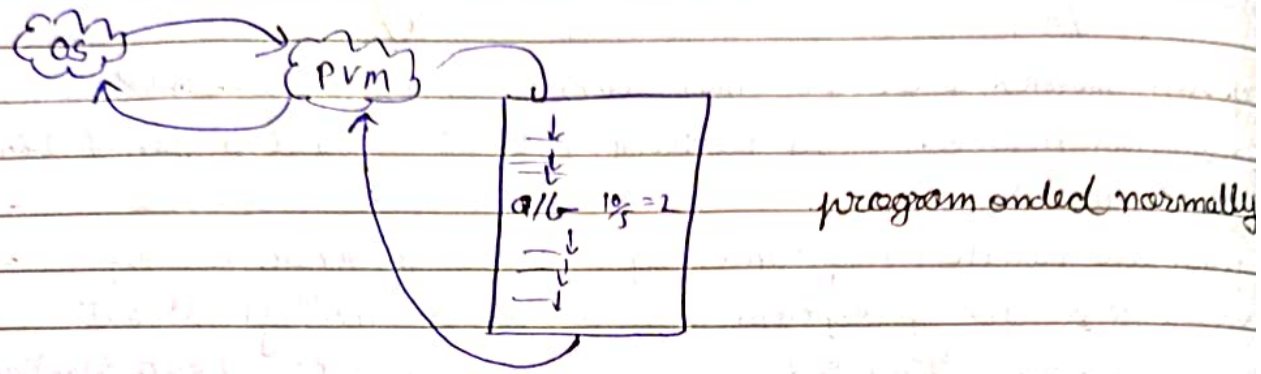


Ex-1

```
a = int(input(" Enter numerator "))
b = int(input(" Enter a denomiter "))
result = a/b
print(result)
print(" program terminated normally ")
```

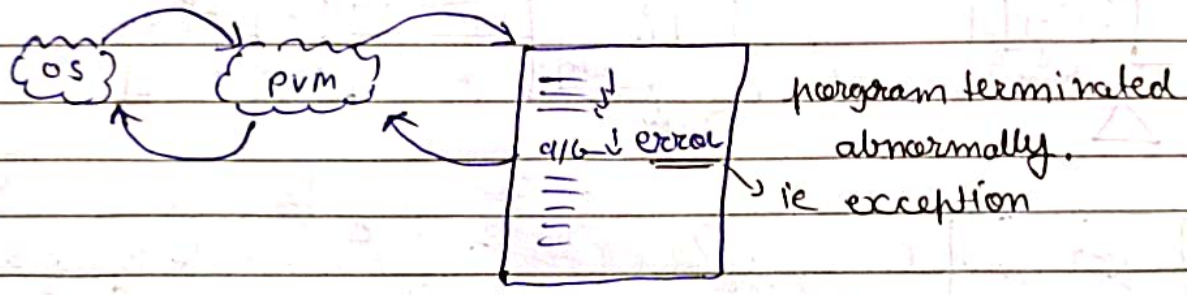
Case-1

a = 10 , b = 5



Case 2

a = 10 , b = 0



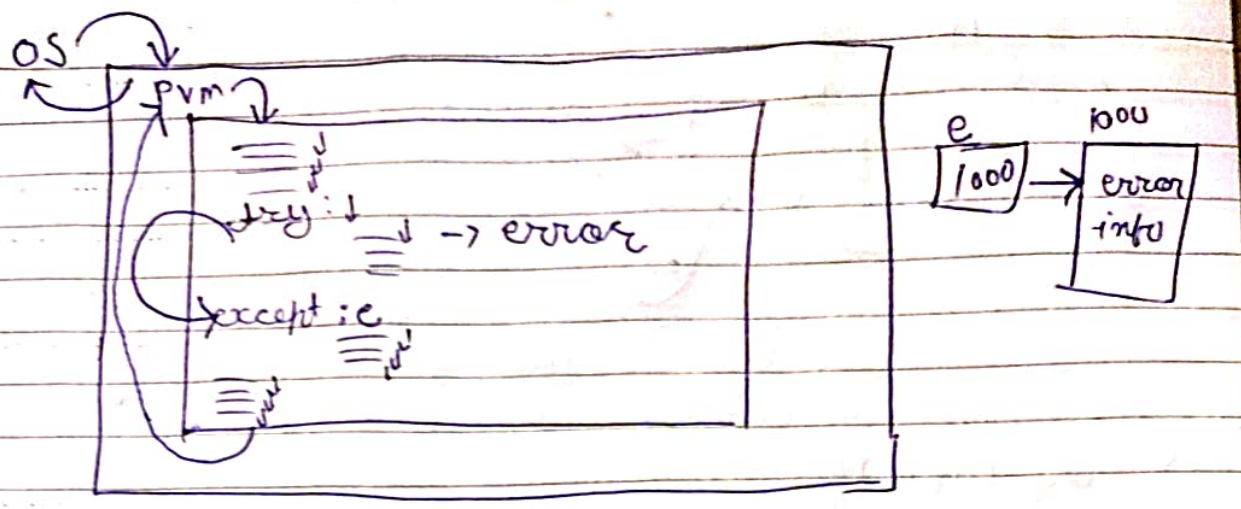
ex 2

```
a = input(" Enter numerator : ")
b = input(" Enter denometer : ")
try :
        res = a/b
        print(res)
except Exception as e :
        print(" Exception Handled ")
print(" Program terminated normally ")
```

try Block

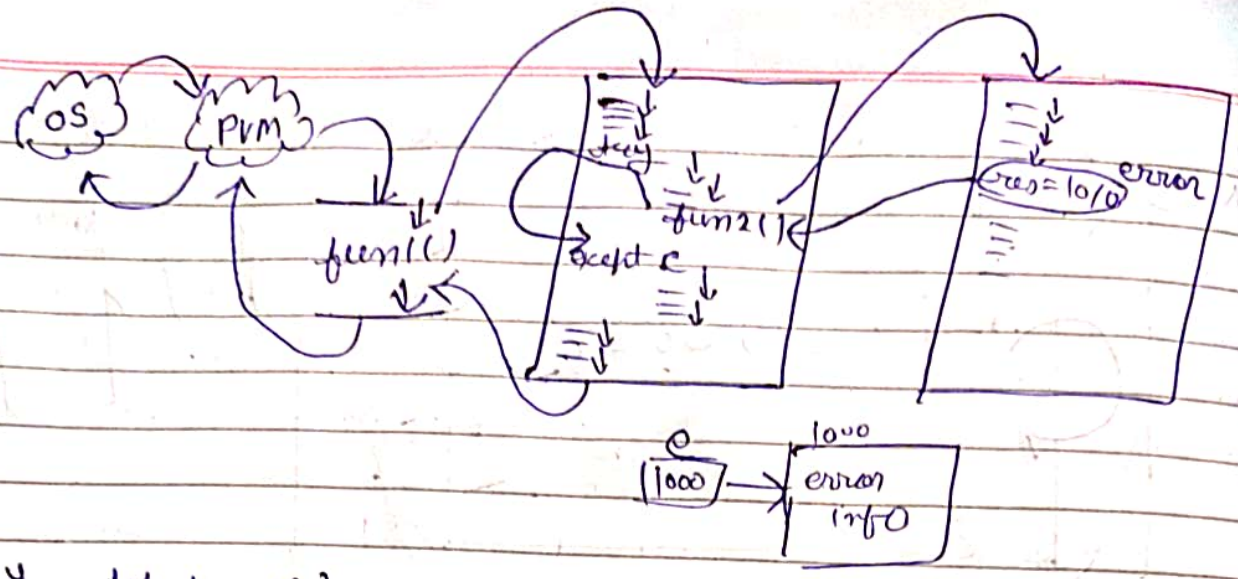
except Block

=> So if we encounter any exception in try Block then try Block will ~~give~~ shift control to Except Block & program will end normally.



-> if an error occur during execution, it is termed as exception
-> if exception is not handled properly, the program will terminate abnormally
-> in order to avoid this, the instruction that could generate exception, should be put in try Block
-> inside except Block, we must write instruction that will Handle exception.

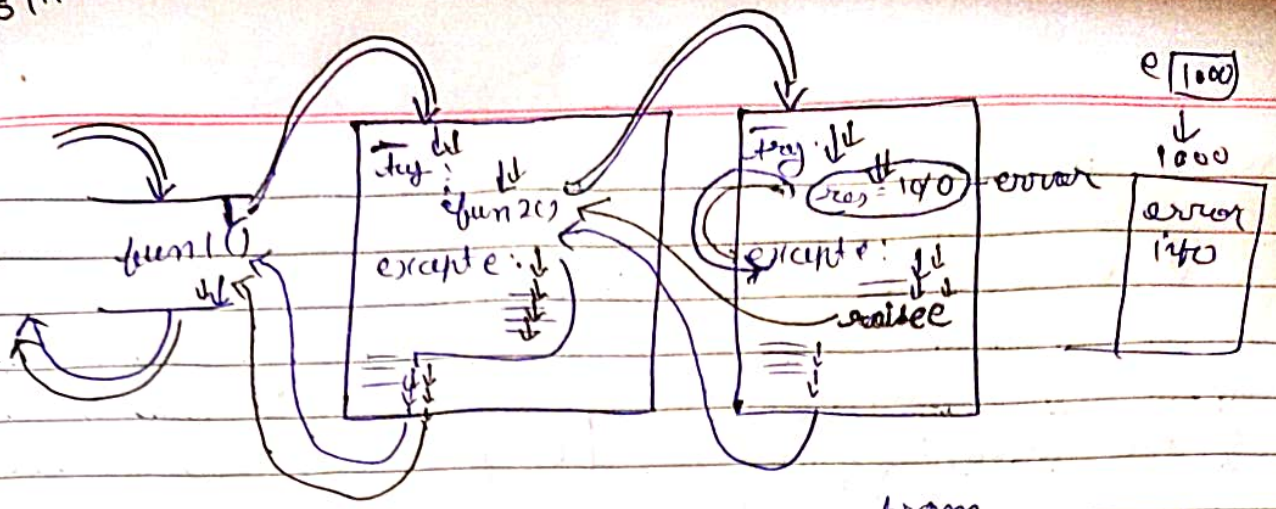
ex 3
```
def fun1()
    print("Entering fun1")
    try:
        fun2()
    except Exception as e:
        print("Exception caught inside fun1")
    print("leaving fun1")

def fun2():
    print("Entering fun2")
    res = 10/0
    print("leaving fun2")

print("program start")
fun1()
print("program end")
```



ex 4

```
def fun1():
    print(" Entering fun1 ")
    try:
        fun2()
    except Exception as e:
        print(" exception caught in fun1 ")
    print(" leaving fun1 ")

def fun2():
    print(" Entering fun2 ")
    try:
        res = 10/0
    except Exception as e:
        print(" Exception Handled in fun2 ")
        raise e
    print(" leaving fun 2 ")

print(" program start ")
fun1()
print(" program end ")
```

25/1/22



⇒ Handling exceptions and rethrowing it from the called f^n to the caller is called as rethrowing an exception. this can be implemented using raise.

---

* Specific Exception :

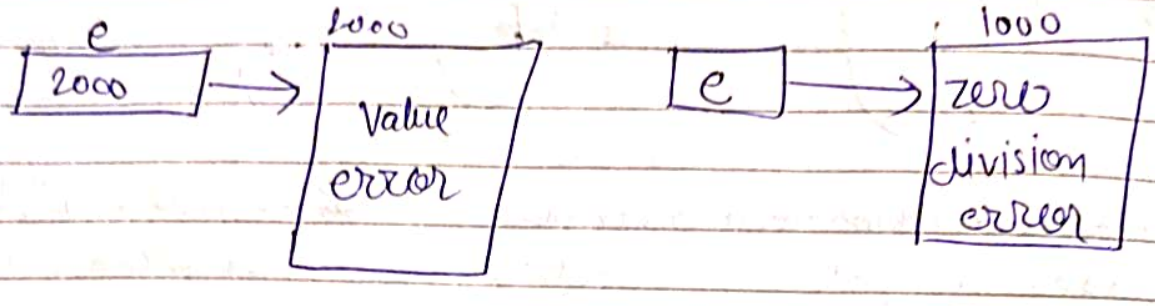
```
try:
    a = int(input("Enter no"))
    b = int(input("Enter no"))
    res = a/b
    print(res)

except ZeroDivisionError as e:
    print("Denominator can't be zero(0)")

except ValueError as e:
    print("give proper input")

except Exception as e:
    print("unknown Exception occurred")

print("Program terminated normally")
```

- a = 10 , b = 0
- a = 10 , b = "abc"

ended

so $a = 10, b = 0$ # zero division error

$a = 10, b = 'abc'$ # Value error



→ the order in which Except Block are written is very important.

⇒ for 1 try Block we can write many except Block, But, when Exception occur only 1 except Block will run, which is matched with occurred error.

⇒ if an execution occure within the try Block, an object crossponding to that particular exception will be created, this object will be compair with the type of Exception Handed in each of the except Block linearly [one by one] even if, we have multiple except Block, but only one of them will be executed.
the order in which multiple except Block are written also matters. In begining we have to write except Block, that can Handle specific exception. towards the end we have to write except Block that are genralized.

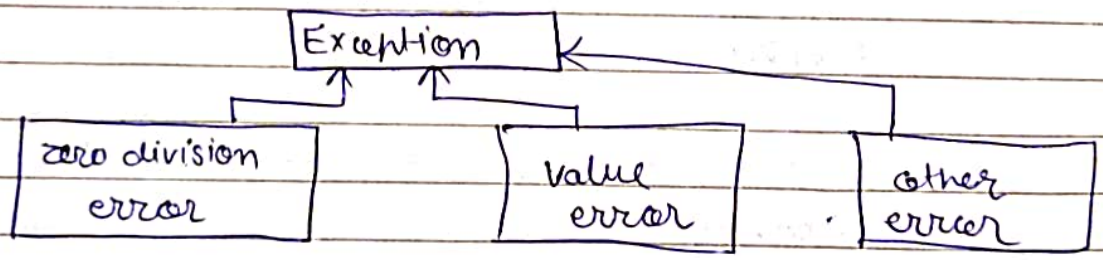
*

__str__() : to give appropriate Exception msg

```
try :
    a = int (input ("Enter no ")
    b = int (input ("Enter no ")

except (zerodivisionerror, Value error ) as e:
    # print (" either zero divison or value error ")

    print ( e.__str__())
        or
    print (e)
print (" program terminated normally.
```

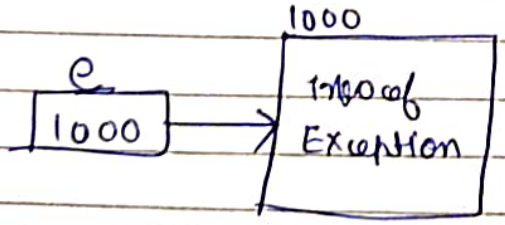
they will appropriate error msg.

⇒ __str__() method is a method of Exception class



So,

we can use this method with all child class.



so e is object of error [Exceptio
so we can invoke __str__() method by e.

* Default Exception Block :

```
try :
    ___
    ___
    ___

except zerodivisionerror as e :
    ___
    ___
    ___
```

try Block

Specific Exception

except value error as e : } specific exception

except Exception as e : } General exception

except : } default exception Block.

* finally Block :- it is executed, irrespective to exception
it means, it will execute surely, no matter we have
exceptionor not.

why we use finally :

way 1

```
try:
    open()
    ===
    close()
except zerodivisionerror as e :
    ===
    close()
except Exception as e :
    ===
    close()
```

Not Standard ✗

way 2 :

```
try :
    open()
    ===
    close()
except zerodivisionerror as e :
except Exception as e :
    ===
finally : close()
```

Standard ✓

finally will execute surely.

ex

```
def fun2()
    print("Establish connection with DB in fun2")
    try:
        res = 10/0
        print(res)
    except Exception as e:
        print("Exception caught in fun2")
        raise e
    finally:
        print("Disconnect DB in fun2")

def fun1()
    print("establish connection with DB in fun1")
    try:
        fun2()
    except Exception as e:
        print("Exception Handled in fun1")
        raise e
    finally:
        print("Disconnect DB in fun1")

print("program start")
    try:
        fun1()
    except exception as e:
        print("Exception Handled in main")

    print("program terminated")
```

O/P without finally:

```
program start
establish connection with DB in fun1
establish connection with DB in fun2
Exception caught in fun2
Exception caught in fun1
Exception Handled in main
program terminated
```

O/P with finally

→ program start
→ establish connection with DB in fun1
establish connection with DB in fun2
exception caught in fun2
Disconnect DB in fun2
exception Handled in fun1
Disconnect DB in fun1
exception Handled in main
program terminated

→ finally statements.

⇒ So, the finally block will be executed even if we raise an exception

* Throwing Exception manually:

So As we know when we got a Exception then 2 steps are completed
(i) create an Exception object
(ii) Throw the object

} to throw the Exception manually we need to do these 2 steps.

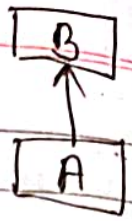
manually throwing an Existing Exception ☆

```
a = int (input (" Enter no"))
b = int (input (" Enter no"))
try :
        if (a<b):
            e = Exception ()
            raise (e)
        else :
            print (a-b)
except Exception as e :
        print (" subtraction not Possible")
print (" program terminated normally ")
```

} manually throwing exception by using class Exception

calling Existing Exception

note A Extends B :
↳ inherit all property & Behaviour



* Creating custom Exception : creating our own new exception

```
Class AlessthanBException (Exception):
    def __init__(self):
        print(" sub not Possible ")
```
} defining Exception

```
a = int(input(" Enter no "))
b = int(input(" Enter no "))
try:
    if (a<b):
        e = AlessthanBException()
        raise e
    else:
        print(a-b)
except AlessthanBException as e:
    print(" value of a<b ")
print(" program terminated normally ")
```
} throwing Exception

But here print(e), print(e.__str__()) will not work why?

* steps to create custom Exception (4 Steps)

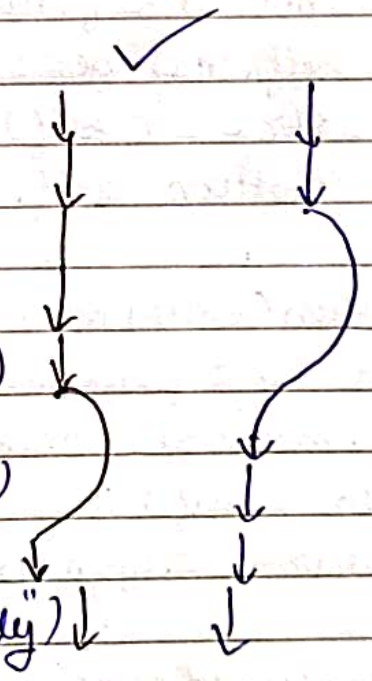
(i) create a Class for custom Exception
(ii) extend Exception class (ie inherit from)
(iii) Pass the error msg.
(iv) overide __str__() method

```
a = int(input(" Enter no "))
b = int(input(" Enter no "))
try:
    if (a<b):
        raise AlessthanBException(" Sub not Possible as a<b ")
    else:
        res = a-b
        print(res)
except AlessthanBExceptionase :
    print(e)                    # print(e.__str__())
print(" program terminates normally ")

class AlessthanBException(Exception):
    def __init__(self, message):
        self.message = message
    def __str__(self):
        return self.message

Class minBalException(Exception):
    def __init__(self, message):
        self.message = message
    def __str__(self):
        return self.message

Class NegAmtException(Exception):
    def __init__(self, msg)
        self.msg = msg
    def __str__(self):
        return self.msg

Bal = 5000
amt = int(input(" Enter ammount :")
try:
    if (amt > Bal):
        raise minBalException(" Amt > Bal ")
```

```
elif ( amt < 0 ):
        raise NegAmtException(" Amt < 0 ")
else:
        Bal = Bal - Amt
        print (" updated Balance ", Bal)

except minBalanceException as e :
        print (e)
except NegAmtException as e :
        print (e)
print (" program terminated normally ")
```

* Else Block

```
try:
        print (inside try")

except Exception as e :
        print (" inside except ")
else:
        print (" inside else ")
finally:
        print (" inside finally")
```



— if we got Exception
— if we not got exception

* Variation of try / Except / else / finally Block.

try: —	X	Except: — X	alone not Possible
else: —	X	finally: — X	

```
try:
    try:
    except:
```

```
except:
    try:
    except:
```

```
else:
    try:
    Except:
```

```
finally:
    try:
    Except:
```

these all are allowed ie try / except Pair inside any Block is allow.

```
try:
except:
else:
finally:
```

correct & valid sequence

```
try:
else:
```

X Not allow

```
try:
finally:
```

✓ allowed

★

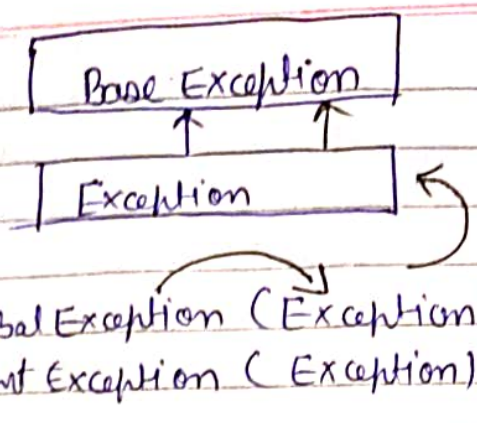
```
try:
    print("inside try")
    exit()
except exception as e
    print("inside exception")
else:
    print("inside else")
finally:
    print("insidefinally")
```

```
Except:
else:
finally:
```

X without try anything is Not Allowed

O/P inside try
inside finally

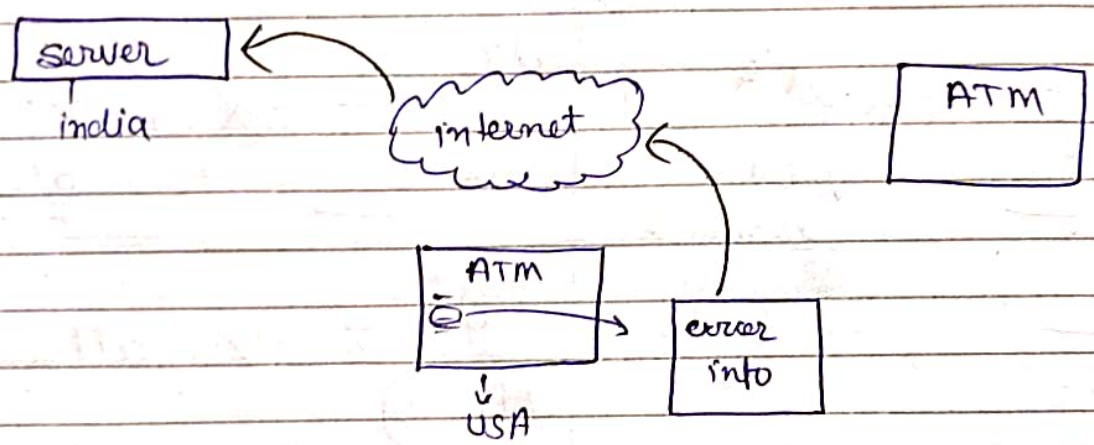
*



minBalException (Exception)
NegAmtException (Exception)

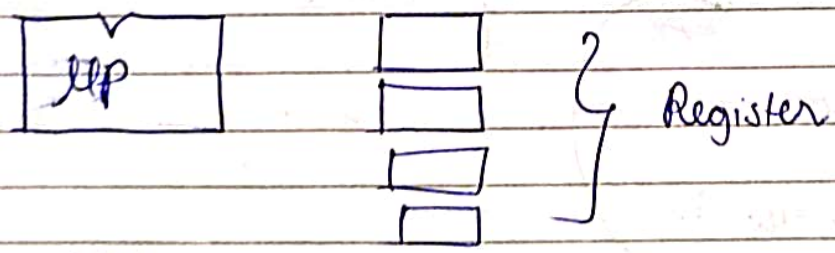
so,
we can throw object which are connected to Base Class.

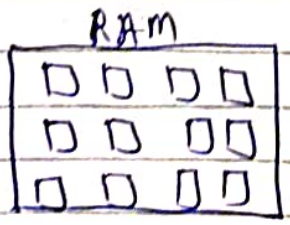
* why we throw Exception / Error



---

* lists :





variable, But tracing become difficult
-> searching & Sorting not Possible

Array : sequential & continious memory allocation

problem in Array : (i) Array is " Homogenous "
(ii) Not Dynamic data ie size can't exhand

linked list : dynamically allocated, ~~sequential~~

problem in linked lisit : (i) linked list is " Homogeneous "

To solve problem of linked list " Lists " is introduced in Python

→ Hetrogenous in nature
→ Dynamic data structure

eg L = [10 | 20 | 30 | 40 | 50] ✓
L = [10 | 10.5 | Ramu | a] ✓

L[0] = 10
L[1] = 10.5
L[2] = "Ramu"
L[3] = 'a'

Syantax :
listname = [value 1, value 2 - - - - -]

* value 1 & value 2 can be int, float string, complex

#1

a = 10
b = 10.5
c = "mj"
D = [10, 20, 30]
print (type (a)) → class int
print (type (b)) → class Float
print (type (c)) → class Str
print (type(d)) → class list

* valid lists:

```
a = [ 10, 20, 30 ]                ✓
b = [ 10, 10.5, 20.40 ]           ✓
c = [10, "Ramu", "mj", 10.5 ]     ✓

d = [ ]                # Empty list
```

#

```
i=0
l=[ ]
while True :
    num = int (input ("Enter no"))
    l.insert ( i, num )   # insert element in list.
    i=i+1

    print (" want to continue press 1
            to end press 2 ")
    choice = int (input ("Enter choice")
        if (choice ==1)
            continue
        elif ( choice ==2)
            Break
        else:
            print ("invalid input")
            Break
print (l)  ->  [10,20,30,40]
```

* list slicing

```
        -10  -9  -8  -7  -6  -5  -4  -3  -2  -1
a = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
     0   1   2   3   4   5   6   7   8   9

print (a[0]) -> 10
print (a[-9] -> 20
print (a[1:7] -> [20,30,40,50,60,70]
print (a[:4]) -> [10,20,30,40]
```

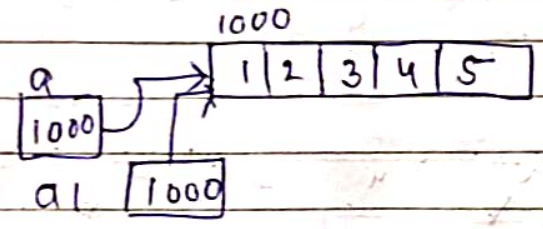
print (a[8:3 :-1]) -> [90, 80, 70, 60, 50]
print (a[-10: -7]) -> [10, 20, 30]
print (a[-1, -5, -2]) -> [100, 70]

print (a[: : -1]) -> reverse list

* Shallow copy & Deep copy

#1
a = [1, 2, 3, 4, 5]
a1 = a
print (a is a1) -> True

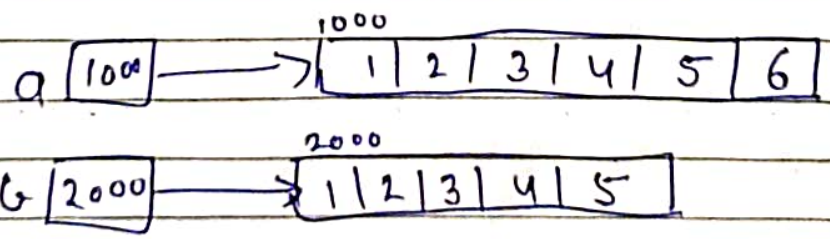
" SHAllow copy"



#2 a = [1, 2, 3, 4, 5]
b = a. copy () # copy f^n
print (a is b) -> False

" Deep copy"

print (a) -> [1, 2, 3, 4, 5]
print (b) -> [1, 2, 3, 4, 5]
a.insert (5, 6)
print (a) [1, 2, 3, 4, 5, 6]
print (b) [1, 2, 3, 4, 5]



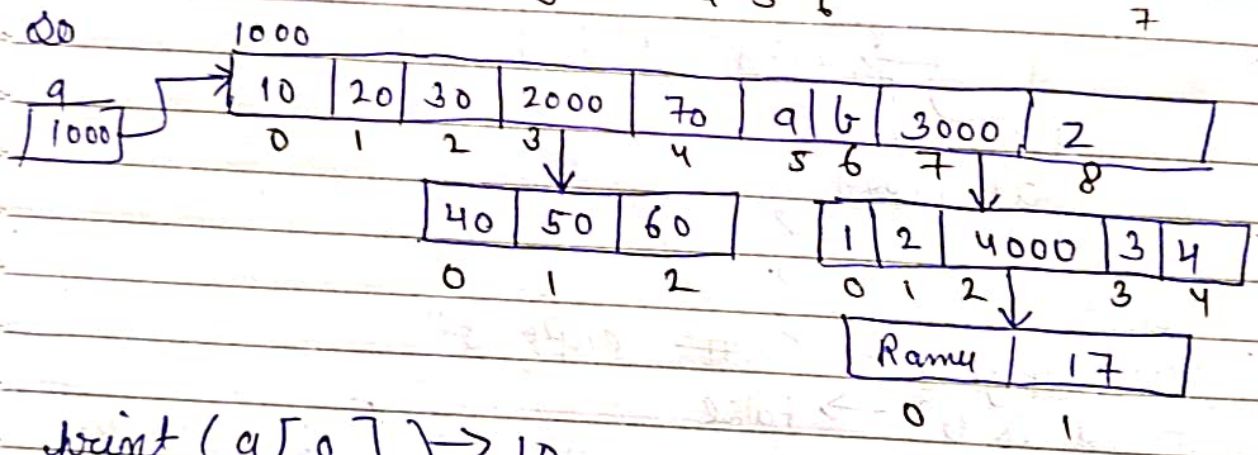
Deep copy: it make separate copy of data & Assign it to new variable ie new variable Point to it.

```
                    0   1   2
a = [ 10, 20, 30, [ 40, 50, 60 ], 70 ]
      0   1   2          3        4
```

print (a[0]) → 10
print (a[3][0]) → 40
print (a[3][2]) → 60
print (a[4]) → 70

* Nested list :

```
                    0   1   2                 0   1        2
a = [ 10, 20, 30 [ 40, 50, 60 ], 70, 'a', 'b' [ 1, 2 [ "Ramu", 17 ] ]
      0   1   2        3        4    5    6        7        2
```



print (a[0]) → 10
print (a[3][0]) → 40
print (a[7][1]) → 2
print (a[7][2][1]) → 17
print (a[7][2][0]) → Ramu

* operation on list

1. concatination :

```
a = [ 10, 20, 30 ]
b = [ 40, 50, 60 ]
c = a+b
print(c) → [10, 20, 30, 40, 50, 60 ]
```

2. Repeat :

```
a = [1, 2, 3]
b = a * 3
print(b) → [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

{ - and / are not supported in lists }

3. in

```
a = [10, 20, 30, 40, 50]
if (55 in a) → False
if (50 in a) → True
```

4. Not in

```
a = [11, 22, 33, 44, 55]
if (50 not in a) → True
if (33 not in a) → False
```

* function on lists

#

```
a = [10, 20, 30, 40, 50]
print(min(a)) → 10
print(max(a)) → 50
print(len(a)) → 5
```

#

```
a = [10, 20, 30, 40, 50]
b = [10, 20, 0, 40, 25]
print(all(a)) → True
print(all(b)) → False
```

all function return true if all element of list are non zero

#

```
a = [10, 20, 30, 40, 50]
b = [10, 20, 0, 40, 25]
c = [0, 0, 0, 0, 0]

print(any(a)) → True
print(any(a)) → True
print(any(a)) → False
```

any function return True if there is at least 1 non zero element in the list

#

```
d = [55, 11, 33, 22, 44]
sorted(d)                    # Ascending order
print(d) →                   [11, 22, 33, 44, 55]

sorted(d, reverse = True)    # descending order
print(d) →                   [55, 44, 33, 22, 11]
```

note: when we sort, the list in memory is sorted

* Method with list

a = [1, 2, 3, 4, 5] # insert method

a.insert (5, 6) → [1, 2, 3, 4, 5, 6]
a.insert (3, 100) → [1, 2, 3, 100, 5, 6]

append method

a.append(70)
a.append (80) } [1, 2, 3, 100, 5, 6, 70, 80]
a.append (11, 22, 33) [error]

extend method

b = [11, 22, 33]
a.extend(b)
or
a.extend([11, 22, 33]) } [1, 2, 3, 100, 5, 6, 70, 80, 11, 22, 33]

To create dimensional list

b = [1, 2, 3]
b.append (4, 5, 6) # error
b.append ([4, 5, 6]) → [1, 2, 3, [4, 5, 6]]
ie 2D list created

so b → | 1 | 2 | 3 |

b → | 1 | 2 | 3 | 2000 | (1000)
↓
| 4 | 5 | 6 |

so to insert element or to Tes size of list we have 3 method insert, append, extend

or to create multi-D list we have append([list]) method

* To remove element from list

a = [10, 20, 30, 40, 50, 60]

remove method

a.remove (30)
print (a) → [10, 20, 40, 50, 60]

a.pop() # pop method
print (a) → [10, 20, 40, 50]
a.pop ()
print (a) → [10, 20, 40]

del keyword

del a[1]
print (a) → [10, 40]

a.clear() # clear method
print (a) → [ ]

del a → delete list a
a.clear → empty list a

b = [10, 20, 30, 40]

del b[2] → [10, 20, 40]

del b
print (b) → Error

b is not defined, reference is deleted.

b = [10, 20, 30, 40]

b.clear()
print (b) → [ ]
empty list

data is deleted, reference is not deleted.

* other methods

```
a = [ 10, 20, 30, 40, 50, 60, 10, 20, 10 ]
print ( a. index(10) ) -> 0          # index return index
print ( a. index(50) ) -> 4          of first occurence of
                                     data.

print ( a. count(10) ) -> 3          # count return total
print ( a. count(60) ) -> 1          occurence of data.

a. sort()                            # sorting
print ( a ) -> [ 10,10,10, 20,20, 30, 40, 50, 60 ]

a. sort ( reverse = True )           # reverse sorting
print ( a ) -> [ 60, 50, 40, 30, 20, 20, 10, 10, 10 ]

b = [ 11, 55, 33, 22, 44 ]           # reverse
print ( b. reverse ) -> [ 44, 22, 33, 55, 11 ]

a = list ( range (10) )
print ( a ) -> [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ]
```

=> to convert collection of data into list we use "list" f^n.

* list traversal

#1

```
a = [ 10, 20, 30, 40, 50 ]
print(a)                -> [ 10, 20, 30, 40, 50 ]

for data in a :
    print (data)        -> 10
                           20
                           30
                           40
                           50
```

#2 If we want to print index along with data

```
for index, data in enumerate(a):
        print( index, "  ", data )
```

```
->   0    10
     1    20
     2    30
     3    40
     4    50
     5    60
```

* list Comprension :

ex1
```
a = [10, 20, 30, 40, 50]
b = []
for i in a
        b.append(i+1)
print(b)  ->  [11, 21, 31, 41, 51]
```

=>

```
b = [ i+1 for i in a ]
print(b) ->  [11, 21, 31, 41, 51]
```
} list comprension

ex2
```
a = [1, 2, 3, 4, 5]
b = []
for num in a
        data = num*num
        b.append(data)
print(b) ->  [1, 4, 9, 16, 25]
```

=>

```
b = [num*num for num in a]
print(b) -> [1, 4, 9, 16, 25]
```

Syntax :

list_name = [(mathematical op for n in list/collection)]

ex 3
```
a = [ "ramu", "rita", "rahul", "arjun" ]
b = [ ]
for n in a:
        data = n.upper()
        b.append(data)
print(b) → [ "RAMU", "RITA", "RAHUL", "ARJUN" ]
```

⇒
```
b = [ n.upper() for n in a ]
print(b) → [ "RAMU", "RITA", "RAHUL", "ARJUN" ]
```

ex 4
```
evenlist = [ ]
n = int(input(" Ente no : ")
evenlist = [ num for num in range(i, n+2+1)
                    if (num % 2 == 0) ]
print(evenlist) → [ 2, 4, 6, 8, 10 ]  if n = 5
```

ex 5
```
a = " abc 1234 #n5y6z7 "
b = [ char for char in a if (char.isdigit() == True) ]
print(b) → [ 1, 2, 3, 4, 5, 6, 7 ]
```

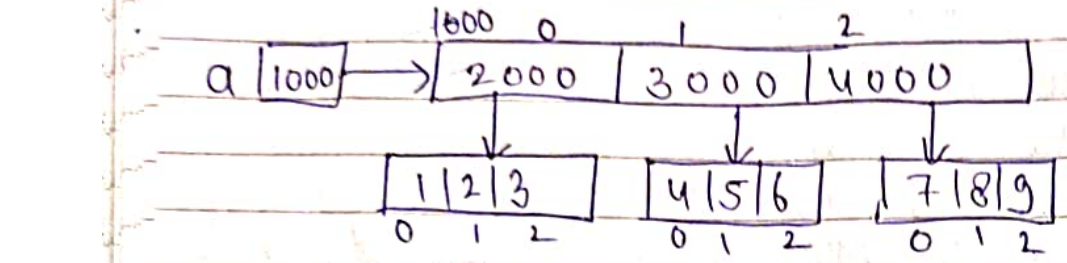
ex 6
```
def square(num):
        return num*num
a = [ 1, 2, 3, 4, 5 ]
b = [ square(n) for n in a ]
print(b) → [ 1, 4, 9, 16, 25 ]
```

* Representing 2D & 3D Array.

ex1

```
a = [ [1,2,3],[4,5,6],[7,8,9] ]
print(a[1][2]) -> 6
print(a[2][0]) -> 7
```

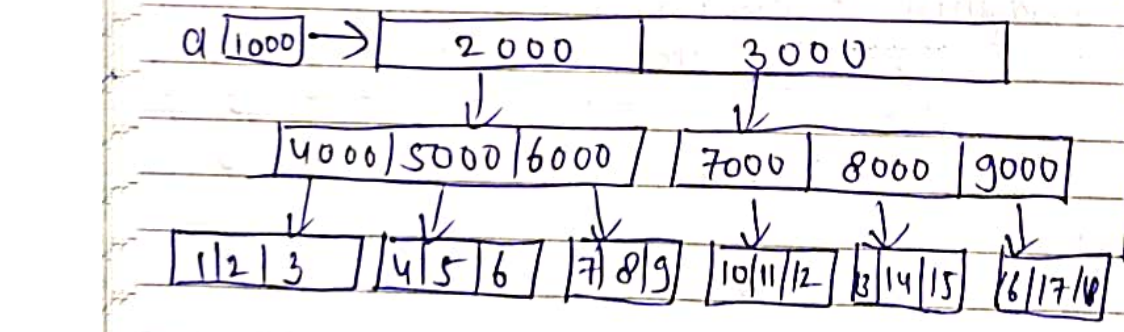
	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9



ex 2

```
a = [ [ [1,2,3],[4,5,6],[7,8,9] ], [ [10,11,12]
      , [13,14,15],[16,17,18] ] ]
```

```
print(a[0][2][1]) -> 8
print(a[1][2][0]) -> 13
```



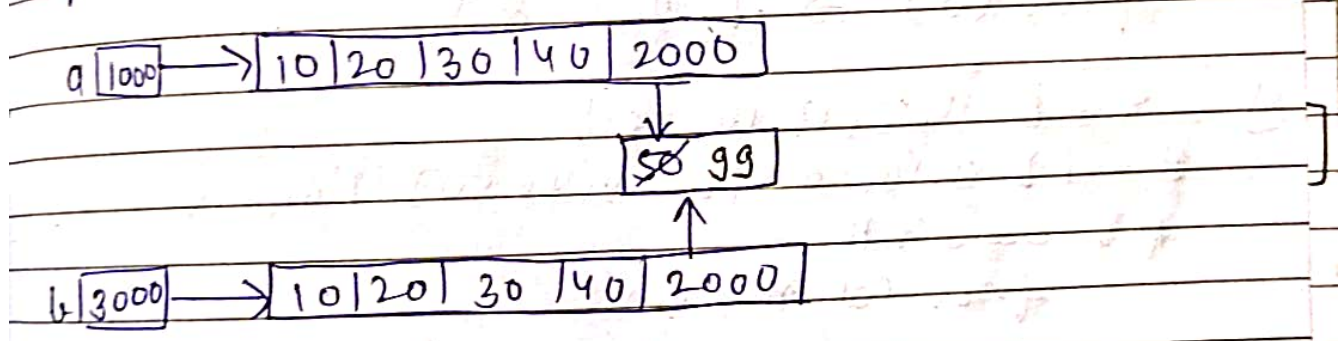
WAP to program to store element in 2D list also print 2D list.

```
row = int(input(" Enter row"))
col = int(input(" Ente col"))
a=[]
b=[]
for i in range(rows):          # to store
    for j in range(col)
        num = int(input(" Enter no"))
        b.append(num)
    a.append(b)
    b=[]
```

```
for i in range(row):                    # to print
    for j in range(col)
        print( a[i][j], end=" ")
    print()
```

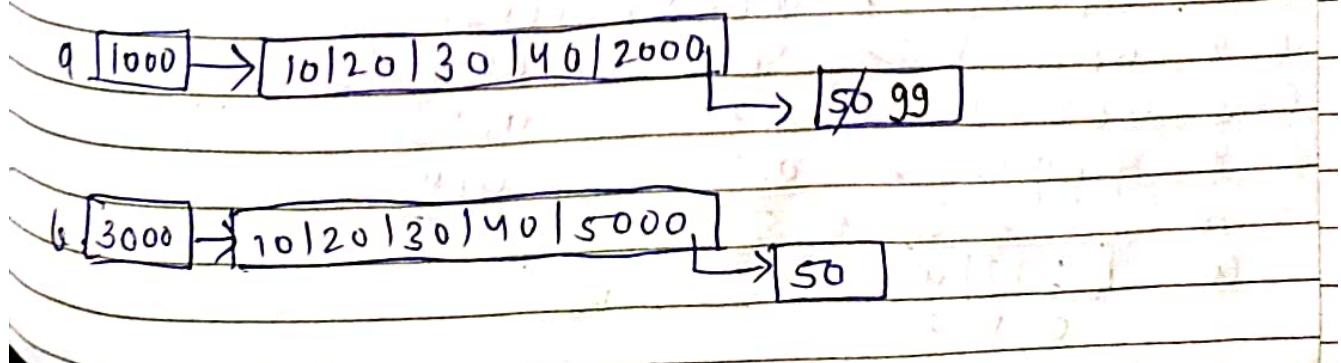
* Copy VIS Deep copy

```
a = [10, 20, 30, 40, [50]]
b = ~~copy~~ a.copy()
print(a)  →  [10, 20, 30, 40, [50]]
print(b)  →  [10, 20, 30, 40, [50]]
a[4][0] = 99
print(a)  →  [10, 20, 30, 40, [99]]
print(b)  →  [10, 20, 30, 40, [99]]
```



* import copy

```
a = [10, 20, 30, 40, [50]]
b = copy.deepcopy(a)                    # deepcopy
print(a)  →  [10, 20, 30, 40, [50]]
print(b)  →  [10, 20, 30, 40, [50]]
a[4][0] = 99
print(a)  →  [10, 20, 30, 40, [99]]     # diff
print(b)  →  [10, 20, 30, 40, [50]]
```



2/2/22

* list compression Example :

1)

```
a = [[ 1,2,3], [4,5,6], [7,8,9] ]
b = [ a[i] [len(A) - 1 - i ] for i in range (len(a))
```

i=0 a[0] [3-1-0] => a[0][3] = 3
i=1 a[i] [3-1-1] => a[1][1] = 5
i=2 a[2] [3-1-2] => a[2][0] = 7

b = [3, 5, 7]

2)

```
a = [[1,2,3], [4,5,6], [7,8,9]]
b = [ a[row][1] for row in (0,1,2) ]
print(b) -> [ 2,5,8 ]
```

3)

```
r = [ 11,12,13,14,15 ]
A = [[ 0,10,20], [ 30,40,50], [60,70,80] ]
for row in A
        for col in row :
                r.append (col+10)
print(r)
-> [11,12,13,14,15,10,20,30,40,50,60,70,80,90, ]
```

4)

```
a = [1,2,3,4,5]
for i in range (1,5)
        a[i-1] = a[i]
for i in range (0,5)
        print (a[i])
```

i=1 a[1-1] => a[0] = a[1]
i=2 a[2-1] => a[1] = a[2]
i=3 a[3-1] => a[2] = a[3]
i=4 a[4-1] => a[3] = a[4]

a =	1	2	3	4	5
a =	2	3	4	5	5
	0	1	2	3	4

5) l = [3, 4, 6, 1, 2]
l[1:2] = [7, 8] => l[1] = [7, 8]
print (l) → [3, 7, 8, 6, 1, 2]

✓ ✗ ✗ ✗

6) l = ["good", "oh!", "cool!", #405]
b = [n for n in l if (n.isalpha() or n.isdigit())]
print (b) → ["good"]

7) print ([i+j for i in "abc"] for j in "def"]

abc
① ② ③

def
① ② ②

ad, bd, cd
ae be ce
af bf cf

☆ [['ad', 'bd', 'cd'], [ae', 'be', 'ce'], [af', 'bf', 'cf']]

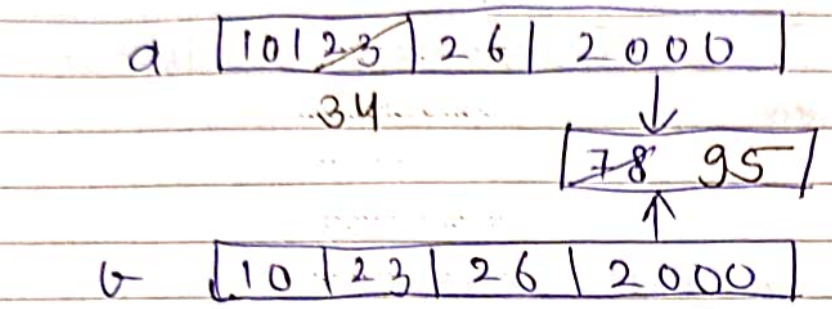
8) a = [1, 2, 3]
b = a.append (4)
print (a) → [1, 2, 3, 4]
print (b) → [ ] # append doesn't return any value so b is empty

9) a = " hello"
b = list ((x.upper(), len(x)) for x in a)
tuple

hello
① ② ③ ④ ⑤

print (b)
→ [('H', 1), ('E', 1), ('L', 1), ('L', 1), ('O', 1)]

10) a = [10, 23, 26, [78]]
b = list(a) # deepcopy
a[3][0] = 95
a[1] = 34
print(b) → [10, 23, 26, [95]]



* list Unpacking
a = [10, 20, 30]
b = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
a1, a2, a3 = a
b1, b2, b3 = b

print(a1) → 10	print(b1) → [1, 2, 3]
print(a2) → 20	print(b2) → [4, 5, 6]
print(a3) → 30	print(b3) → [7, 8, 9]

if x1, x2 = a → Error
x1, x2, x3 = a → ✓ ; x1 = 10, x2 = 20, x3 = 30

* list Recap : list are dynamic, mutable, Heterogeneous, insertion order is maintained, duplicate are allow

fn	method
min()	append() ✓
max()	insert() ✓
all()	extend() ✓
any()	remove() ✓
len()	pop() ✓
sorted() ✓	clear() ✓
sum()	index()
reverse() ✓	count()
	sort() ✓
	reverse() ✓

to implement marked method & fn. collection must be "MUTAGIE"

so List are mutable

* TUPLE * ( )

→ Hetrogenous data
→ it also maintain insertion order
→ duplicate are allow
→ Immutable

syantax

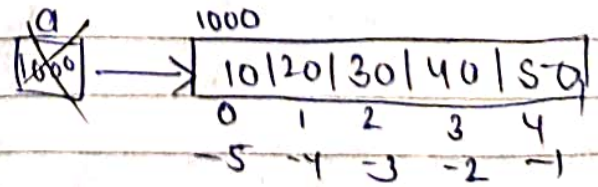
tuplename = (value1, value2, value3 ----)

```
t = ( 1, 2, 3, 4, 5 )        ✓
t = ( 1, 1.0, 2, 2.0 )       ✓
t = ( 1, 1.0, " Ramu " )     ✓
t = ( )          # emptytuple
```

```
#  a = ( 1, 2, 3, 4, 5 )
   print (a)              →  ( 1, 2, 3, 4, 5 )
   print ( type (a))      →  < class ' tuple ' >
```

```
#  b = (1)
   print (b)              →  1
   print ( type (b))      →  < class ' int ' >

   b = ( 1, )
   print (b)              →  (1)
   print ( type (b))      →  < class ' tuple ' >
```

```
#  a = ( 10, 20, 30, 40, 50)
```

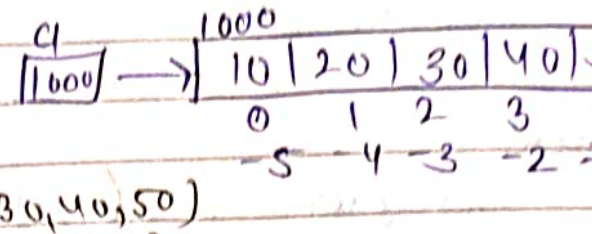



```
a[2] = 99     X error
del a[3]      X error
del a         ✓ tuple referecd deleted
```

} Immutable data }

* Tuple Slicing :

a = (10, 20, 30, 40, 50)



print (a[0]) -> (10)
print (a[1:3]) -> (20, 30)
print (a[: :]) -> (10, 20, 30, 40, 50)
print (a[4 : 1 : -1]) (50, 40, 30)
print (a[: : -1]) (50, 40, 30, 20, 10) # Reverse

#

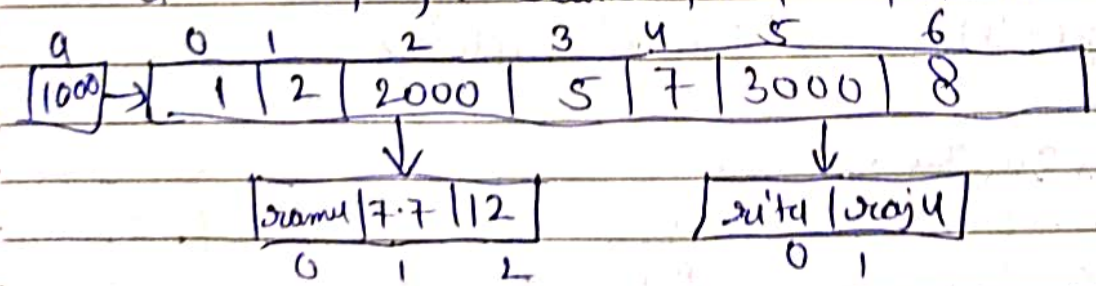
a = (11, 22, 33, 44, 55)
for i in a :
    print (i, end = " ") -> 11 22 33 44 55

for index, data in enumerate (a)
    print (index, " ", data)

0	11
1	22
2	33
3	44
4	55

* Nested tuple

a = (1, 2, ("ramu", 7.7, 12), 5, 7, ("rita", "raju"), 8



print (a[2][0]) -> ramu
print (a[5][1]) -> raju
print (a[4]) -> 7

* list with tuple : valid ✓

student_info = ("Arjun", [75, 61, 83])
name, marks = student_info
print(name) → Arjun
print(marks) → [75, 61, 83]

* inserting & deleting data within tuples

a = (1, 2, 36, 25, 71)

a | 1 | 2 | 36 | 25 | 71 |

Now, we want to add 99 after 2

b = a[0:2] + (99,) + a[2:]

so,

b | 1 | 2 | 99 | 36 | 25 | 71 |

c = a[0:4] + (11, 12, 13, 14, 15) + a[4:]

c | 1 | 2 | 36 | 25 | 11 | 12 | 13 | 14 | 15 | 71 |

for deleting we can use same technique

a = (1, 2, 36, 25, 71)
# want to delete 36

d = a[0:2] + a[3:]
d = | 1 | 2 | 25 | 71 |

we need to create new tuple for insertion & deletion

```
a = (10,20,30)
print(a)          → (10,20,30)
print(type(a)) → class 'tuple'
```

By default collection are treated as tuple

```
a = 10,20,30
print(a)          → (10,20,30)
print(type(a)) → class 'tuple'
```

if there are a collection of no that are enclosed with in any type of Bracket, the By default we threat that collection of no as "Tuple"

ex

```
class calculator
    def arthops(self, a,b):
        r1 = a+b
        r2 = a-b
        r3 = a*b
        r4 = a/b
        return r1,r2,r3,r4
c1 = calculator
res1, res2, res3, res4 = c1.arthops(10,20)
```

so

Here res1, res2, res3, res4 = (r1, r2, r3, r4)

ie Tuple unpacking is done here

* zip function: perform "Parallel iteration"

```
name = ["ABD", "DHoni", "Virat", "Rohit"]
runs = [7000, 9000, 12000, 12000]
country = ["SA", "IND", "IND", "IND"]
IPLf = ["RCB", "CSK", "RCB", "MI"]
```

```
player_info = list(zip(names, runs, country, iplf))
print(player_info)
```

⇒ zip fn will take out all list data which have same index from all list & put it in a Bundle (tuple)

O/P : [("ABD", 7000, "SA", "RCB"), # tuple inside a list.
("Dhoni", 9000, "IND", "CSK"),
("Virat", 12000, "IND", "RCB"),
("Rohit", 12000, "IND", "MI")]

So, note : the zip f^n can be used to parallely iterate through a particular index of all the list. the data at that particular index in all the table will be extracted & Bundled inside a tuple.

☆ we have an issue with zip f^n : the output of zip f^n (no of tuple) is equal the length of shortest list. this is because the zip f^n will stop iterating after all the element of shortest list have been extracted

ex
let we remove ipl f of Rohit & virat -
Now we run again & check for output.

O/P
[("ABD", 7000, "SA", "RCB"),
("Dhoni", 9000, "IND", "CSK")]

* zip longest () f^n : this function works similar to zip but it will give all data.
So for same condition ∀ virat & Rohit have no ipl f.

```
from itertools import zip_longest
player_info = list(zip_longest(name, run, country, ipl f))
print (player_info)
```

O/P
[('ABD', 7000, 'SA', 'RCB'),
('Dhoni', 9000, 'IND', 'CSK'),
('Virat', 12000, 'IND', 'None'),
('Rohit', 12000, 'IND', None)]

we can use fillvalue = '#' to change default value.

So zip longest ~~f~~ gives 'None' for empty Block

* sets * { }

sets are dynamic, mutable, Heterogenous.
⇒ insertion order may & may not be maintained
⇒ duplicate are not allowed

Syntax : a = { }

a = { 10, 20, 30, 40, 50 } ✓
a = { 10, 10.5, "Ramu", 'a' } ✓

a = { } ✗ error
a = set() # empty set

a = { 1, 2, 3, 4, 5, 1, 1, 7, 2 }
print (a) # in any order it can print.
print (type (a) → ('class set')

a = { 10, 20, 30 }
print (a [2]) ✗ } error
print (a [1 : 3]) ✗

⇒ As in sets insertion order is not maintained
so we can't print value by their index
ie index & slicing are not allow in set.

a = { 1, 2, 3, 4, 5, 11, 1, 2, 3 }
for i in a
print (i, end = " ")
⇒ we can't predict output order
⇒ But we can predict no of element in output
because duplicate are not allowed
so for this
no of element = 6
output order ⇒ it can by anything

to print index & value in set (enumerate ())

```
c = { 11, 22, 33, 44, 55 }
for index, Data in enumerate(c)
        print (index, "  ", Data)
```

* to insert element in Sets

```
a = set()
n = int (input (" Enter no of element to store : "))
for i in range (n):
        data = int (input (" Enter no : ")
        a. add (data)        # to insert one element
print(a) -> { 1, 2, 3, 4 }

a. update ( [13, 14, 15] )  # to add list in set
print (a) :-> { 1, 13, 14, 3, 2, 15, 4 }   ie multiple data
```

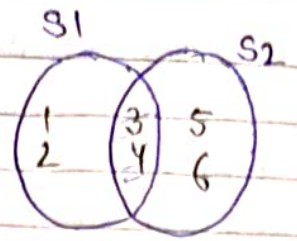
* to delete element in sets

```
        data = int (input (" Enter element to delete : ")
        a. discard (data)          # to delete element

#> try :
        a. remove (data)        # to delete element
        print (" Data deleted ")
except Exception as e :
        print (" Element not present in set ")
```

discard() & remove() Both works similarly But if <u>remove () does not find element in set, then it will throw an exception.</u>
if discard does not find element in set, then it will do nothing

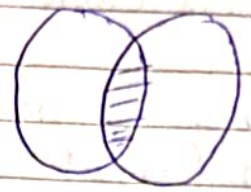
* sets properties

#1 S1 = { 1, 2, 3, 4 }
S2 = { 3, 4, 5, 6 }
S3 = S1. union (S2)
→ { 1, 2, 3, 4, 5, 6 }



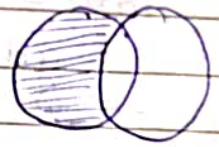
S1∪S2 = S1 + S2 - S1∩
Singl

#2 S4 = S1. intersection(S2)
print(S4) → { 3, 4 }



#3 S5 = S1. difference (S2)

S1 = { 1, 2, 3, 4 }
S2 = { 3, 4, 5, 6 }
print(S5)
→ { 1, 2 }



S1 − S1∩S2

#4 S6 = S1. symmetric_difference (S2)

A+B − A∩B

#5 S1 = { 1, 2, 3, 4 }
S2 = { 3, 4 }
print (S1. issubset (S2)) → False
print (S2. issubset (S1)) → True

print (S1. issuperset (S2) → True
print (S2 . issuperset (S1) → False

# a = { 1, 2, 3, 4 }
b = { 3, 4 }
c = { 5, 6 }

no common element

print (a. isdisjoint(b)) → False
print (a. isdisjoint(c)) → True
print (b. isdisjoint(c)) → True

4/2/22

```
# s1 = { 1, 2, 3, 4 }
  s2 = { 3, 4 }
  s1.intersection_update(s2)        # s1 updated by s1∩s2
  print(s1) → { 3, 4 }
  print(s2) → { 3, 4 }

# s1 = { 1, 2, 3, 4 }
  s2 = { 3, 4 }
  s1.difference_update(s2)
  print(s1) → { 1, 2 }              # s1 updated by
  print(s2) → { 3, 4 }                diff of s1, s2

# s1 = { 1, 2, 3, 4 }
  s2 = { 3, 4 }
  s2.difference_update(s1)          # s2 updated by diff of
  print(s1) → { }                     s1, s2
  print(s2) → { 1, 2, 3, 4 }

# s1 = { 1, 2, 3, 4 }               # s1 updated by
  s2 = { 3, 4 }                       s2+s1 - s1∩s2
  s1.symmetric_difference_update(s2)
  print(s1) → { 1, 2 }
  print(s2) → { 3, 4 }

# s1 = { 5, 6, 7, 8 }               # s2 updated by
  s2 = { 7, 8, 9, 10 }                s1+s2 - s1∩s2
  s2.symmetric_difference_update(s1)
  print(s1) → { 5, 6, 7, 8 }
  print(s2) → { 5, 6, 9, 10 }
```

* frozen set : (i) duplicate not allow
(ii) insertion order may or may not maintained
(iii) Heterogeneous
(iv) Immutable *

fs = frozenset([10, 20, 30, 40])
print(fs) → {10, 20, 30, 40}

fs.add
fs.update
fs.remove
fs.discard
} Not work, Because it is "Immutable"

* "Inside set we can't have mutable collection", So Inside set we can't have (i) nested set, list, But we can have tuple as tuple is immutable.

{1, 2, {3, 4}, 5} ✗ set
{1, 2, [3, 4], 5} ✗ list
{1, 2, (3, 4), 5} ✓ tuple

* Dictionary : we store data as Key value Pair
* Syantax:

```
dict_name = { Key1 : value1,
              Key2 : value2,
              :    :    :
              Keyn : valuen }
```

ex

```
employee = { "name" : "Rahul",
             "age" : 22,
             "Salary" : 15000,
             "location" : "Bhl" }
```

Key : it gives descripti

Key = name, age, salary, location
Value = "Rahul, 22, 15000, bhl

student_info
1000 → 1000

#

```
student_info = {"Name" : "mj",
                "marks" : 78,
                "Age" : 22,
                "RollNo" : "1822",
                "loc" : "BhL",
                "Hobby" : "Dance"}
```

Key	value
Name	mj
marks	78
Age	22
RollNo	1822
loc	BhL
Hobby	Dance
gender	male

print (student_info) → Print whole dictionary
print (student_info['Age']) → 22
print (Student_info['Hobby']) → Dance
student_info['gender'] = 'male' # item added in dictionary
print (type (student_info)) → <class 'dict'>
note : In dictionary we don't have index so slicing is not supported

```
# for i in student_info :
        print (i, end = ' ')
```
→ Name, marks, Age, RollNo, loc, Hobby, gender

```
# for i in student_info :
        print (student_info[i])
```
→ mj, 78, 22, 1822, BhL, Dance, male

```
# for i in student_info.values () :
        print (i)
```
→ mj, 78, 22, 1822, Bhl, Dance, male

```
# for key, value in student_info.items () :
        print (key, " : ", value)
```
→
Name : mj
marks : 78
Age : 22
RollNo : 1822
Loc : Bhl
Hobby : Dance
gender : male

2

* Shallow copy / deep copy

d = { 1:111, 2:222, 3:333, 4:444, 5:555

1)

print (d[4]) → 444

~~print~~ (d[6]) = 666

print (d) → print whole dictionary

d[3] = 300

print (d) → print whole dictionary with updated value

del d[4] # 4 : 444 deleted

print (d) → {1:111, 2:222, 3: 300, 5:555, 6:666}

1	111
2	222
3	~~333~~ 300
~~4~~	~~444~~
5	555
6	666
7	777

2) d1 = d # shallow copy (Both point same object)

d[7] = 777

print (d) } {1:111, 2:222, 3:300, 5:555, 6:666, 7:777}

print (d1)

3) d2 = d1.copy() # Deep copy (Both point diff object

del d1[5]

print (d) } {1:111, 2:222, 3:300, 6:666, 7:777}

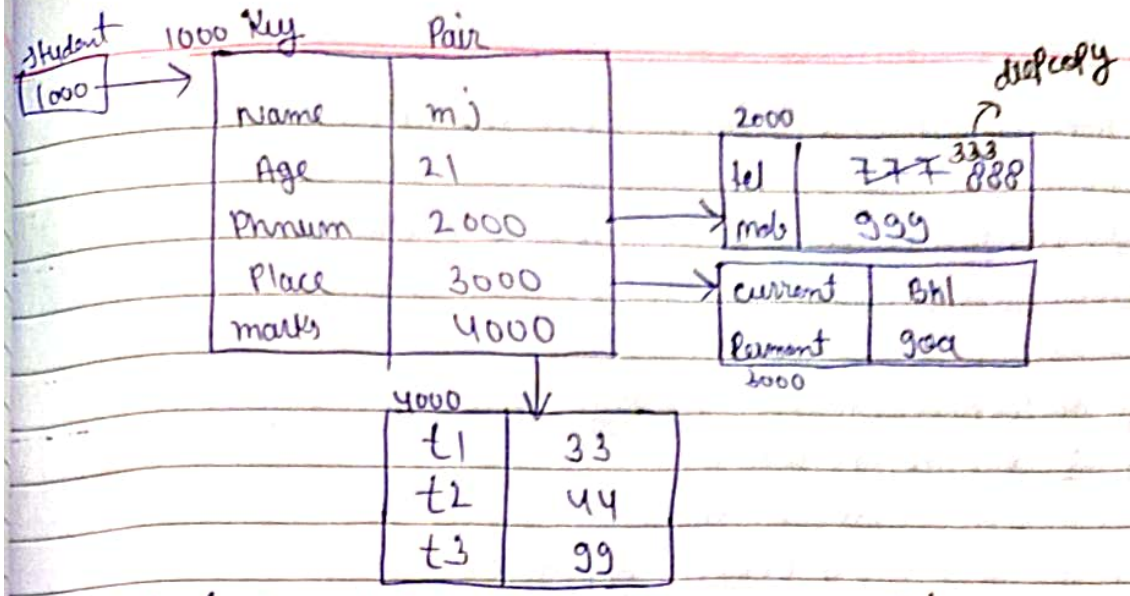
print (d1)

print (d2) { 1:111, 2:222, 3:300, 5:555, 6:666, 7:777}

* Nested dictionary

4)

```
student = { 'Name' : 'mj'
            'Age' : 21
            'Phnum' : { 'tel' : 777
                        'mob' : 999
                      },
            'Palace' : { 'current' : 'BHI',
                         'Permant' : 'goa'
                       },
            'marks' : { 't1' : 33,
                        't2' : 44,
                        't3' : 99
                      }
          }
```



print (student ['Age']) → 21

print (Student ['Place']['permant']) → 900

print (student ['Phnum']) → {tel : 777, mob : 999}

student (['Phnum']['tel']) = 888

print (student ['Phnum']['tel']) → 888

import copy

s1 = copy.deepcopy(student) # deep copy

print (student ['Phnum']['tel'])
print (s1 ['Phnum']['tel'])
} → 888

student ['Phnum']['tel'] = 333 # element updated

print (student ['Phnum']['tel']) → 333

print (s1 (['Phnum']['tel'])) → 888

note : Always give double quote (" ") whenever you are accessing dictionary

* Dictionary comprihension :

#1 dict1 = {i:i for i in range (1,10)}

print (dict1) → {1:1, 2:2, 3:3, 4:4, 5:5, 6:6, 7:7, 8:8, 9:9}

#2 dict2 = {i : i*i for i in range (1,10)}

print (dict2) → {1:1, 2:4, 3:9, 4:16, 5:25, 6:36, 7:49, 8:64, 9:81}

dict3 = {i:i*i for i in range (1,10) if (i%2==0)}

print (dict3) → {2:4, 4:16, 6:36, 8:64}

In comprehension size of data is change during runtime dynamically
so comprehension can be done on list, sets, Dictionary (mutable)
comprehension can't be done on tuple {Immutable}

* zip function : it accept only 2 Parameter (Key : Value)

#1 empid = [101, 102, 103, 104]
empname = ["mj", "mgJ", "manu", "manoj"]
empdict = dict(zip(empid, empname))
print(empdict) → {101 : "mj", 102 : "mgJ", 103 : "manu", 104 : "manoj"}

note : zip function return object, we can change data type by type casting.

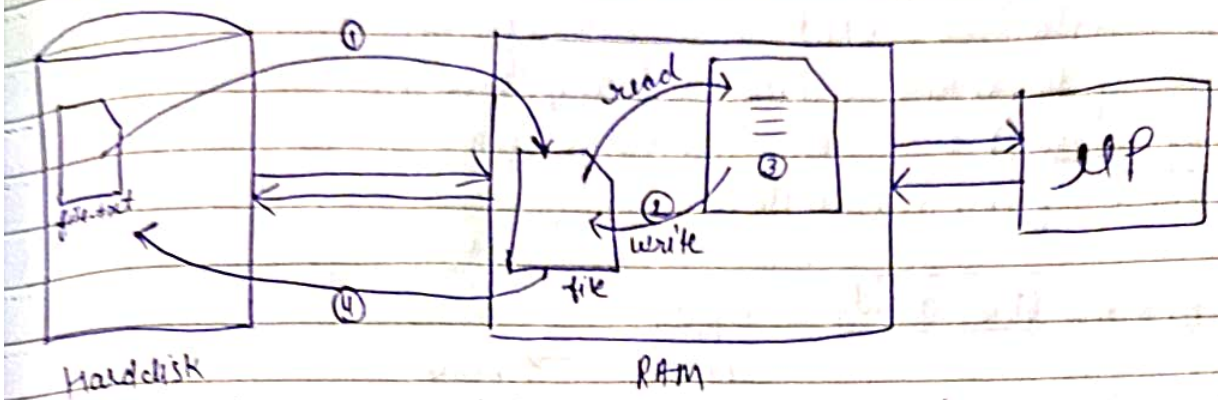
#2 empid = [101, 102, 103, 104]
empname = ["mj", "mgJ", "manu", "manoj"]
empsal = [10000, 15000, 20000, 25000]
emploc = ["Banglore", "goa", "Bhl", "Noida"]

empdict = dict(zip(empid, empname, empsal, emploc)) → ERROR
so ie only 2 Parameter
empdetails = list(zip(empname, empsal, emploc))
empdict = dict(zip(empid, empdetails))
print(empdict)
list list
→ {101 : (mj, 10000, Banglore), 102 : (mgJ, 15000, goa), 103 : (manu, 20000, Bhl), 104 : (manoj, 25000, Noida)}

—x—x—x—x—x—x—x—

* file processing : A file is a container that is used to store data permanently
the activities involving, reading, writing, processing a data of a file is known as "file processing".

there are 2 types of file (i) Binary file : these are the files in which data stored, in terms of 0 & 1
(ii) text file : In these data stored in char format.



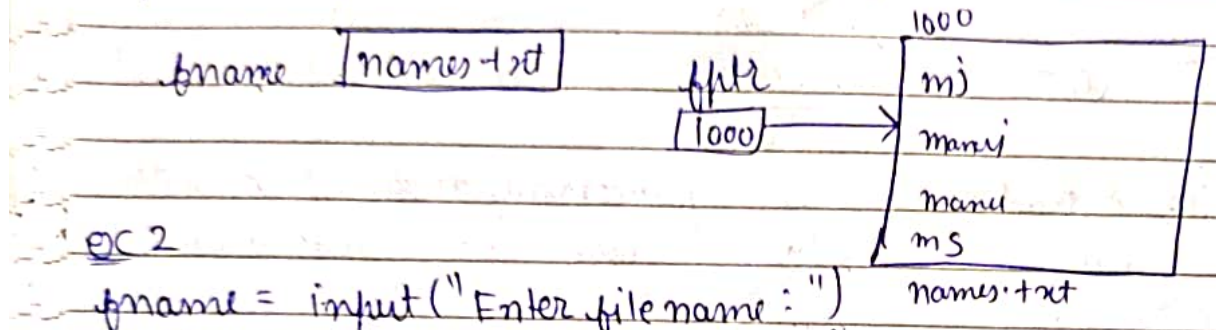
① open file (load file) ② processing file
③ read / write ④ close file (save file)

note : we can have img, mp4, mp3 } Binary file.

* open function : the open function is used to open a file by loading it onto the RAM & make the file, accessable to the entire program.
(i) when we use openf^n, we must provide the file name & mode in which file is opened as parameter.
(ii) if the file with given name is not present in the working directory, then a new file will be created.
★ (iii) if we open file in write mode "w", all the existing data in the file will be erased.
(iv) if we want our privious data along with new data, then open file in append mode "a". so old data is preserve & new data is appended.
data modes :
(i) "r" → read mode, data read from file
(ii) "w" → write mode, data write on file
(iii) "a" → append mode, data appended on file
(iv) "rb" → read binary mode, binary file readed
(v) "wb" → write binary mode, data write on file.

ex 1 WAP to take 5 name as input from user & write & store them in file

```
fname = input ("Enter file name")
fptr = open (fname, "w")      # opening a file
for i in range (5)
        data = input ("Enter a name")
        fptr.write (data)     # fptr.write (data + "\n")
fptr.close()                  # closing a file
print(" five name written into file ")
```



ex 2

```
fname = input("Enter file name :")
fptr = open ( fname, "a")
for i in range (5)
        name = input (" Enter name ")
        id = input (" Enter id ")
        loc = input (" Enter location ")
        Sal = input (" Enter Salary ")
        data = id + " " + name + " " + loc + " " + Sal
        fptr.write (data)
fptr.close()
print(" five record added ")
# to read file
fname = input (" Enter a file name ")
fptr = open (fname, "r")
data = fptr.read () # to read data from file
print (data)
fptr.close()
```

```
data = fptr.readlines ()
print (data) → ['101 Rahul BHI 15000', '102 mj HD 20000', '103 AB BL 10000']
```

① line ② line ③ line

we have 3 variation to read data

(i) fptr.read(15) : Read first/next 15 Byte only from file.

(ii) fptr.readline() : Read a single line from a file

(iii) fptr.readlines() : Read & return all the lines in the file, in a list.

* split() : the split function is used to split a given string. By default it split string whenever it encounter space (' ').

the individual element of string that are form after the split operation are put inside a list & returned

ex str = " mj is enjoying class"

```
data = str.split()
print(data) -> ["mj", "is", "enjoying", "class"]
```

Q WAP to count no of line, word, char in a file.

mj is eating
mj is sleeping
mj is playing

```
fname = input("Enter file name")
fptr = open(fname, 'r')
data = fptr.readlines()
print(data) -> ['mj is eating', 'mj is sleeping', 'mj is playing']
line_count = len(data)
word_count = 0
char_count = 0
for x in data
    tchar = len(x)
    wordlist = x.split()
    tword = len(wordlist)
    wordcount = wordcount + tword
    charcount = charcount + tchar

print("No of lines :", line_count) -> 3
print("No of word :", word_count) -> 9
print("No of char :", char_count) -> 39
```

* Some more function

```
fptr = open("Data.txt", 'r')
print('the name of file:', fptr.name)      → data.txt
print('the mode of file:', fptr.mode)      → r
print('the file is closed:', fptr.closed)  → False
print('readable?', fptr.readable())        → True
print('writable?', fptr.writable())        → False
fptr.close()
print('file closed?', fptr.closed)         → True
```

* Binary files :

WAP to copy content of first file in second file.

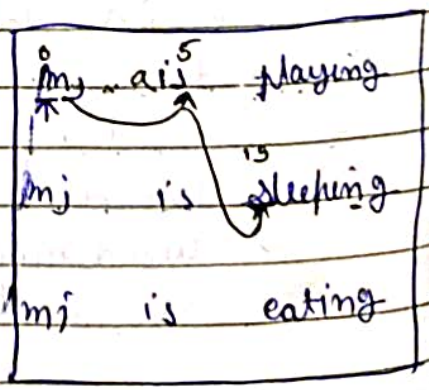
```
fptr1 = open(mycar.jpg, 'r')
fptr2 = open(mynewcar.jpg, 'w')
```

if we give 'r', 'w' then compiler understand them as text file, but we are working on Binary file so use 'rb', 'wb'. ie Binary mode

```
fptr1 = open('mycar.jpg', 'rb')
fptr2 = open('mynewcar.jpg', 'wb')
data = fptr1.read()
print(data)
fptr2.write(data)
fptr1.close()
fptr2.close()
```

* Seek method | tell method

By default fptr Points 0 pos char of file
ie Pos = 0



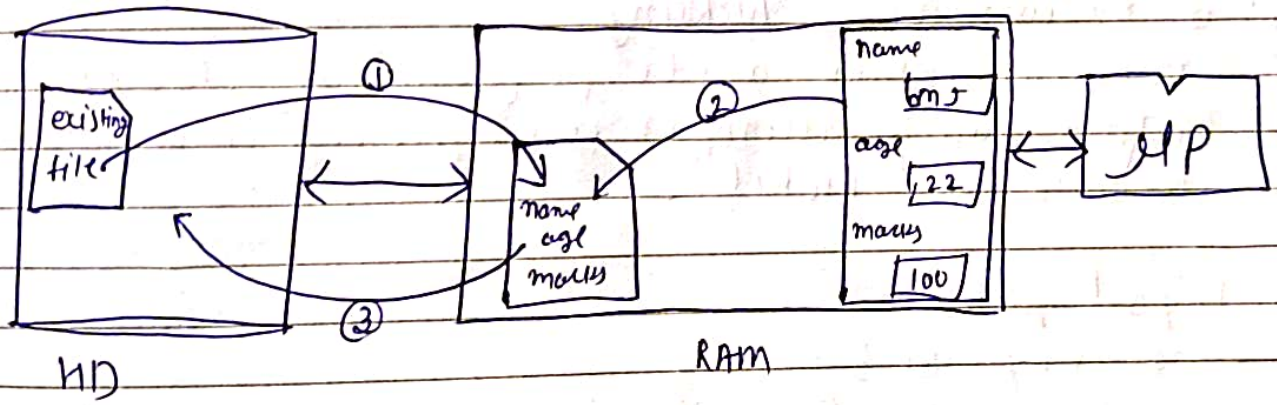
Pos = 0
Pos = 7
Pos = 13

{ tell : it return Position of Pointer is Pointing

{ seek : we can change Position of Pointer by Pass Position in Seek as argument so to move forward & Backward.

```
fptr = open('data.txt','r')
print (fptr.tell())  → 0
fptr.seek (10)
print (fptr.tell())  → 10
print (fptr.read)  → y

fptr.seek (25)
print (fptr.tell())  → 25
data = fptr.read()
print (data)  → n
```

* Serialization / Pickling : it is the process of saving object in Harddisk (which is created during execution of program) for future use.

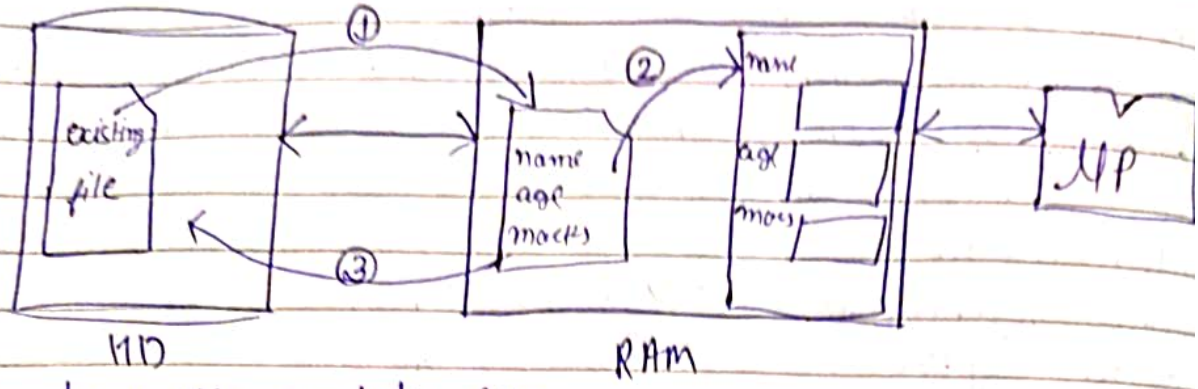


so.

(i) load file onto the ram in 'w' mode.
(ii) dump all data object in file (copy). dump()
(iii) save file (close())

→ the process of storing an object into a file is called as serialization. it is also called as pickling.
→ to perform the operation dump() is used.
→ the dump method is present in a package known as pickel.

* deserialization / depickling / recreate object
to recreate object we perform deserialization



(i) load file onto the RAM
(ii) load object (load())
(iii) close file
→ the process of recreating an object that was stored in a file is called as deserialization. it is also known as depickling.
→ this operation can be performed by load()
→ Both load() & dump() are present in a package known as 'Pickel'.

#

```
import pickle
Class employee :
    def __init__(self, name, age, salary, location):
        self.name = name
        self.age = age
        self.salary = salary
        self.location = location

    def display(self):
        print(self.name)
        print(self.age)
        print(self.salary)
        print(self.location)

e = employee("manoj", 22, 15000, "goa")
```

} # class

object created in serialization

Serialization

```
f = open ("serial.txt", "wb")      # open file
pickle.dump (e, f)                 # dump object in file
print ("object stored in file successfully")
f.close()                          # close file
```

deserialization

```
f = open ("serial.txt", "rb")      # open file
e = pickle.load(f)                 # load file into object
e.display()                        # display object
f.close()                          # close file.
```

---

* data Accessing & processing in CSV file

```
# fptr = open ("emp.txt", "w")
fptr.write ("101 Ramu 15000 900")
fptr.write ("102 Raju 10000 BN")
fptr.write ("103 rita 120000 vij")
fptr.close()
```

fptr →

101	Ramu	15000	900
102	Raju	10000	BN
103	Rita	120000	Vij
⋮	⋮	⋮	⋮

But if we have huge no of record then processing the data is very difficult.

record : details of entity in one row

* to write data in CSV file

```
import CSV
fptr = open("emp.CSV", "w")
w = CSV.writer(fptr)
w.writerow([101, "ramu", 15000, "goa"])
w.writerow([102, "ram", 10000, "Bhl"])
w.writerow([103, "Raju", 12000, "Aj"])
w.writerow([104, "Rita", 5000, "Bang"])
fptr.close()
w.writerow(["id", "Name", "salary", "location"])
```

fptr →

id	Name	Salary	location
101	Ramu	15000	goa
102	Ram	10000	Bhl
103	Raju	12000	Aj
104	Rita	5000	Bang

* to read data from CSV file.

```
import CSV
fptr = open("emp.CSV", "r")
r = CSV.reader(fptr)   # function
print(r) → object
data = list(r)
print(data)
```

→ [101 | Rahul | 15000 | BN] [102 | Ramu | 18000 | Hq]

→
```
for rec in data:
    print(rec)
```

→
```
for rec in data:
    for field in rec:
        print(field, end=' ')
    print()
```

* Automatic closing of file

```
with open ('data.txt', 'w') as fptr:
    for i in range(5):
        n = input ("Enter a name")
        fptr.write(n)
    print ("Names written in files")
print (fptr.closed) → True
```

* Modes to Access file

r → Read
w → write
rt → read primary, w secondary
wt → w primary, r secondary
at → a primary, r secondary
a → append

rb → read binary
wb → write binary
rbt → read primary, write secondary
wbt → w primary, r secondary
abt → a primary, r secondary

{ r, w, a, rb, wb, ab
rt, wt, at, rbt, wbt, abt }

to delete & rename files

```
import os
os.rename ("Data.txt", "Newname.txt")
print ("file is renamed")        # to rename file

os.remove ("names.txt")          # to delete file
print ("file removed")
```

WAP to display the frequency of words in a str

ex

str = " Ramu is eating Ramu is Playing Ramu is sleeping

O/P.

Ramu 3
is 3
eating 1
sleeping 1
playing 1

Algo

→ Split the given string and create a list of wor
→ create an empty dictionary
→ for word in wordlist.
→ Extract the word
→ if (word is already in dictionary)
→ increment the value associated w
word

else :
add the word to the dictionary
make value = 1

Code

```
str = " Ramu is sleeping Ramu is eating Ramu is
        sleeping Ramu is playing

wordlist = str.split()
dict = { }

for word in wordlist :
    if (dict.get(word)):
        value = dict[word]
        value = value+1
        dict[word] = value
    else :
        dict[word] = 1

for key,value in dic.items():
    print("key", ":", value)
```

14/12/22

WAP to display frequency of words in files.

```
f = open('data.txt','r')
data = f.readlines()
print (data)
dict = {}
for line in data
        print (line)
        wordlist = line.split()    # by default = " "
        print (wordlist)
        for word in wordlist:
                if (dict.get(word)):
                        value = dict[word]
                        value = value+1
                        dict[word] = value
                else:
                        dict[word] = 1
for key,value in dict.items():
        print (key, ':', value)
```

* Regular Expression * (Regex)

Regex: it is a string with some special Powers. Regex is normal string, combined with meta characters

Regex = String + metaChar
(*, ., ?, +, [ ], ^, $)

→ In Python a normal string will match another string if & only if the content of Both the string is Exactly the same, But in some application we might require a functionality, where a string matches not only those string that are exactly same But also match those strings that are similar.
In, such case we use Regular Expression (Regex).

1. * : it will match 0 or more (0 --- ∞) occurence of the immediatly previous char.

ex rama* → rama ✓
ramaaa ✓
ram ✓
raam ✗
raamaaa ✗

ex ab*c
→ ac ✓
abc ✓
abbbc ✓
xbc ✗ (must start with a)
abbd ✗ (must end with c)

ex ab*cd*
abcd ✓
abbbc ✓
ac ✓
abbbcddd ✓
xbbydd ✗ (must start with a)
axbbcddz ✗ (must end with d)

2. + : it will match 1 or more (1 --- ∞) occurence of immediate privious char.

ex rama+ → rama ✓
ramaaaa ✓
ram ✗ (must end with a) atleast 1
raam ✗ (start with r
xaam ✗ end with a)

ex ab+c
ac ✗
abc ✓
abbbbbc ✓
xbbbbc ✗
abbby ✗

eg ab*cdt

→ a__c d__

abcd ✓

abbbcdd ✓

acd ✓

ac ✓

axxbbcddz X

3. ? : it will match 0 or 1 occurence of immediate previous char.

eg Rama?

Rama ✓

Ram ✓

Ramaa X

Rammaa X

ram X

rama X

eg ab?cdt

abcd ✓

abc X

acd ✓

axcdy X

abbbcd X

abbcdd X

4. . : it will match any single char

eg ram.

rama ✓

ramb ✓

ramc ✓

rambb X

ram0 ✓

ram9 ✓

ram99 X

eg a.b.c ⇒ a_b_c

ie len = 5

axbyc ✓

axbcy X (must end with c)

a1b1c1 X (len (6))

abc X

aabcc ✓

a1b2c ✓

ex a?b.d*c+ → a b d c

0-∞ 1-∞

abcd ✗ b q c ✓

abdc ✗

abxdc ✓ b1 ddddcccc ✗

aba ✗

5. [ ] : In square bracket we have some char ie character class. we will match only one character within the character class.

ex ram[xy12] → rama ✗

ramx ✓

ramy ✓

ram1 ✓

ram21 ✗ (only 1)

ram2P ✗ (only 1 from class)

ex

ab[mn97]c*d+[xyz] ⇒ a b m c d x y z

97 0-∞ 1-∞

abmcdy ✓ a7cccdddd x2 ✗ b mis

ab91 ✗ axcx ✗ ✗ b mis

ab7d2 ✓

ex

a*b*c+d.[xy12] ⇒ a b c d x y z

0-∞ 0-∞ 1-∞ 12

aaabbbcdz2 ✓

cdqy ✓

abcd77x ✗

ccdc5 ✗

cdm1 ✓

ex abg.[xy12][mn89] len = 5

abgx9 ✓

abgy89 ✗ len 6

ab8x ✗ len 4

ab922 ✗ must end with (m, n, 8, 9)

ab929 ✓

ab9[xy12mn89] len = 4

ab9_ : ab9x9 ✗
ab9y89 ✗
ab8x ✗
ab92 ✓
ab9x ✓

x2[0-9] : x 2 0-9

x20 ✓ x298 ✗
x2x ✗ x29a ✗
x02 ✗ xxxx ✗
x222 ✗ x29 ✓

ab[a-z]12[A-Z] : ab_12_ : len(6)

abe12F ✓ abm12N ✓
ab12E ✗
abaa12X ✗
ab9 2EX ✗

x1[0-9][a-z][A-Z] : x1____ : len(5)

or

x1[0-9 a-z A-Z] x1_ : len(3)

x19aZ : ✓ x19aZ ✗
x1EX : ✗ x1E ✓

(*) x1y[0-9]
↳ want 0-9

x1ya ✗
x1yZ ✗
x1y2 ✓
x1y C3X ✗
x1y 3Z ✗

x1y[^0-9]
↳ not want 0-9

x1ya ✓
x1yZ ✓
x1y2 ✗
x1y3Z ✗
x1y C3X ✗

15/2/22

9x[a-z]* => 9x[a-z][a-z][a-z] ---

9x a ✓
9x z ✓
9x 9 ✗
9x aa ✓
9x aaaaaaaa ✓
9x abcdefgh --- ✓
9x ✓
9x abcdef 123 ✗ (no occur)
x abcd E ✗ (a start with 9) capital char
9abcdef ✗ (x after 9 is must)

9x[a-z]+

9x ✗ (at least 1 char)
9x abcdefgh ✓
9xa ✓

ex ax[^0-9] :

ax2 ✗
axA ✓
axN2 ✗
ax33 ✗

ex ramaaaaa : if we want to match exactly 5 a after ram

ram ✗
rama ✗
ramaaaa ✗
ramaaaaa ✓
ramaaaaaa ✗

ram{5}

ram ✗
ramaaaa ✗
ramaaaaa ✓
ramaaaaaa ✓
ramaaaaaaa ✓

ram{5,}

ram ✗
ramca ✓
ram aaa ✓
ram aaaa ✓
ram aaaaa ✓
ram aaaaaa ✗
ram aaaaaaa ✗

rama{3,5}

so {n} : Exactly 'n' times
{n,} : n & more than 'n' times
{n,m} : min 'n' & max 'm' times.

WA regex to matcha mobile no

[0,9] {10}

or

[9876] [0-9] {9}

or

\+91 [6-9] [0-9] {9}

Now factor
normal char

rama * → meta char

ram a* → normal char
" escape sequence "

WA regex to match email id

_____ @ _____ .com

[a-z A-Z 0-9 _ .] + @ [a-z A-Z] + \.com

[0-9]	→	\d	Single digit
[^0-9]	→	\D	nondigit
[a-z A-Z]	→	\w	single char
[^a-z A-Z]	→	\W	nonchar
[ ]	→	\s	space
[^]	→	\S	nonspace

#

[^] : do the reverse # inside

^ A : must start with A # outside
B $: must end with B

"Ramu is eating, Ramu is sleeping, ramu is playing"

^ Ramu : must start with Ramu
Studying $: must end with Studying

=> WA regex to match a string having exactly 4 cha
→ . . . . or . {4}

=> WA regex to match a line having exactly 10 char
→ ^.....$ or . {10} $

=> ^. {10} $ → line with Exactly 10 char
^. {10, } $ → at least 10 char
^. {10, 15} $ → min 10 char & max 15 char

```
import re
str = ' Ramu Hi Ramuuu Hi Rohan Hi Ram Hi Ramaaa
    Hi Ramuuuuaaaa.
# to see Ram in string
        pattern = " Ramu * "
res = re.search ( pattern, str)
print (res)

res = re.match ( pattern, str)
print (res)
```

Search : search() return the position of the very first substring that matches the regular expression

match : match() will return the position only if the matching substring is there in the begining of the string

findall : findall() return a list containing all the substring that matches to the given pattern. if there is no match then it will return empty list.

WAP to replace all digit with '@'.

```
import re
str = " Ramu 89 Vishal 1234 Suraj 459"
pattern = " [0-9]+"
replace = " @ "
⇒ res = re.sub ( pattern, replace, str)
print (res) → Ramu @ @ vishal @ @@@@ suraj @ @@
```

```
⇒ res = re.subn ( Pattern, replace, str)
print (res) → ("Ramu @ @ vishal @ @@@@ suraj @@@ ", 9)
```

```
⇒ res = re.subd ( pattern, replace, str)
print (res) → Ramu @ vishal @ suraj @
```

WAP to remove space from string

```
import re
str = " Ramu 89 vishal 1234 suraj 459"
pattern = " [0-9]+ "  or  \d+
res = re.split ( pattern, str)
print (res) → [" Ramu ", " vishal ", " suraj "]
```

```
⇒
Pattern = " [ ]+[0-9]+[ ]+ "  or  \s+\d+\s
res = re.split ( pattern, str)
print (res) → ["Ramu", "vishal", "suraj"]
```

ERROR

* let we have a file so Now we want to find
  (i) Ram → aaaa
  (ii) Phone no
  (iii) Email ID

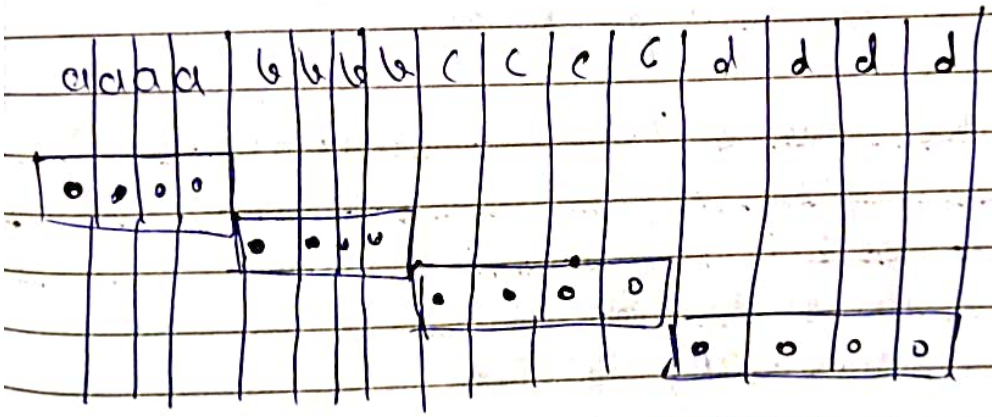
①
```
import re
fptr = open("data.txt", 'r')
data = fptr.read()
pattern = "Rama*"
res = re.findall(pattern, data)
print(res)
```

②
```
import re
fptr = open("data.txt", "r")
data = fptr.read()
pattern = "[0-9]{10}"   or   \d{10}
res = re.findall(pattern, data)
print(res)
```

③
```
import re
fptr = open("data.txt", 'r')
data = fptr.read()
pattern = "[a-z 0-9 _.]+@[a-z A-Z 0-9]
          \.com
res = re.findall(pattern, data)
print(res)
```

```
# import re
str = "Rama is faster at running than Rohan"
pat1 = "^ Rama *"
pat2 = "      Rohan $
res1 = re.findall ( pat1, str)
res2 = re.findall ( pat2, str)
print ( res1)
print (res2)
res3 = re.search ( pat1, str)
res4 = re.search ( pat2, str)
print (res3)
print (res4)          # if not match then it return none.
```

```
# import re
str = " aaaa bbbbcccc dddd"
pattern = "  ....  "
res = re.findall ( pattern, str)
print (res) → ['aaaa','bbbb','cccc','dddd']
```




```
# import re
str1 = " aaaa"
str2 = " bbbbbbb"
str3 = " cccc cccc cccc "
pattern = "  ....  "
res1 = re.findall ( pattern, str1)
res2 = re.findall ( pattern, str2)
res3 = re.findall ( pattern, str3)
print (res1) → ['aaaa']
print (res2) → ['bbbb']
print (res3) → ['cccc', 'cccc', 'cccc']
```

```
let pattern = "^....$"
print(res1)  ⇒  ['aaaa']
print(res2)  →  None
print(res3)  →  None
```

WAP to print all line in a file of length = 5

```
import re
f = open("data.txt", 'r')
data = f.readlines()
pattern = "^.....$"  or  "^.{5}$"

for x in data:
    res = re.search(pattern, x
    if (res):
        print(res)
f.close()
```

```
# import re
str = "Ram Rama Ramaaa Rohan Ramaaaaa"
pattern = "Rama*"
res = re.finditer(patter, str)        # it return item object
print(res)                            # it also collect starting & Ending index
for data in res:
    print(data.start(), end="")
    print(data.end(), end="")
    print(data.group()
    print()
```

```
# import re
str = "RamA Rama rama ramaaa Ramaaa"
pattern = "Rama*"
res = re.findall(pattern, str, re.IGNORE CASE)
for data in res:
    print(data)
```

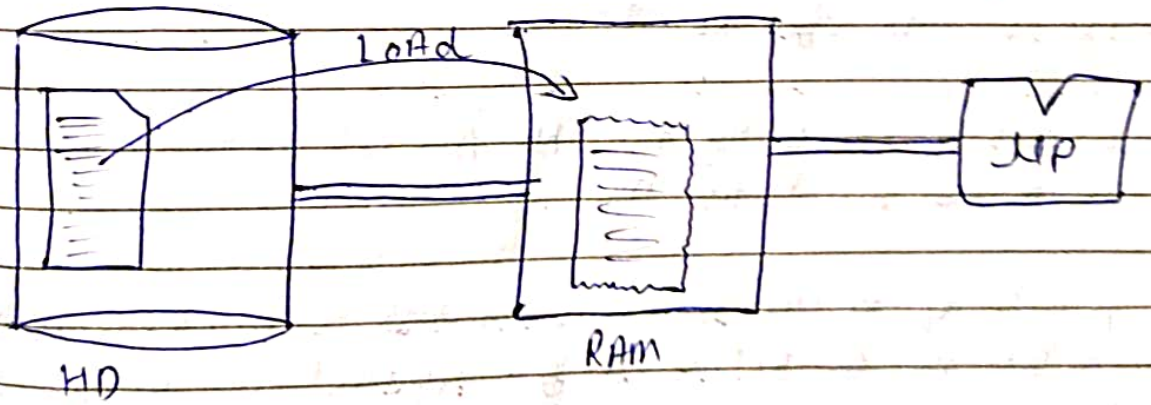
```
# import re
str = " ramaaa ramax "
pattern = re.escape ('rama*')
res = re.findall (pattern, str)
for data in res :
        print (data)  →  rama*
```

{ we can escape regex by \ ★ }
ex re a\?. m* aa\+ y\[\]

```
import re
str = " AAAAA "
replace = " D "
pattern = " A "
res = re.sub ( Pattern, replace, str )
print (res) → DDDDD
```

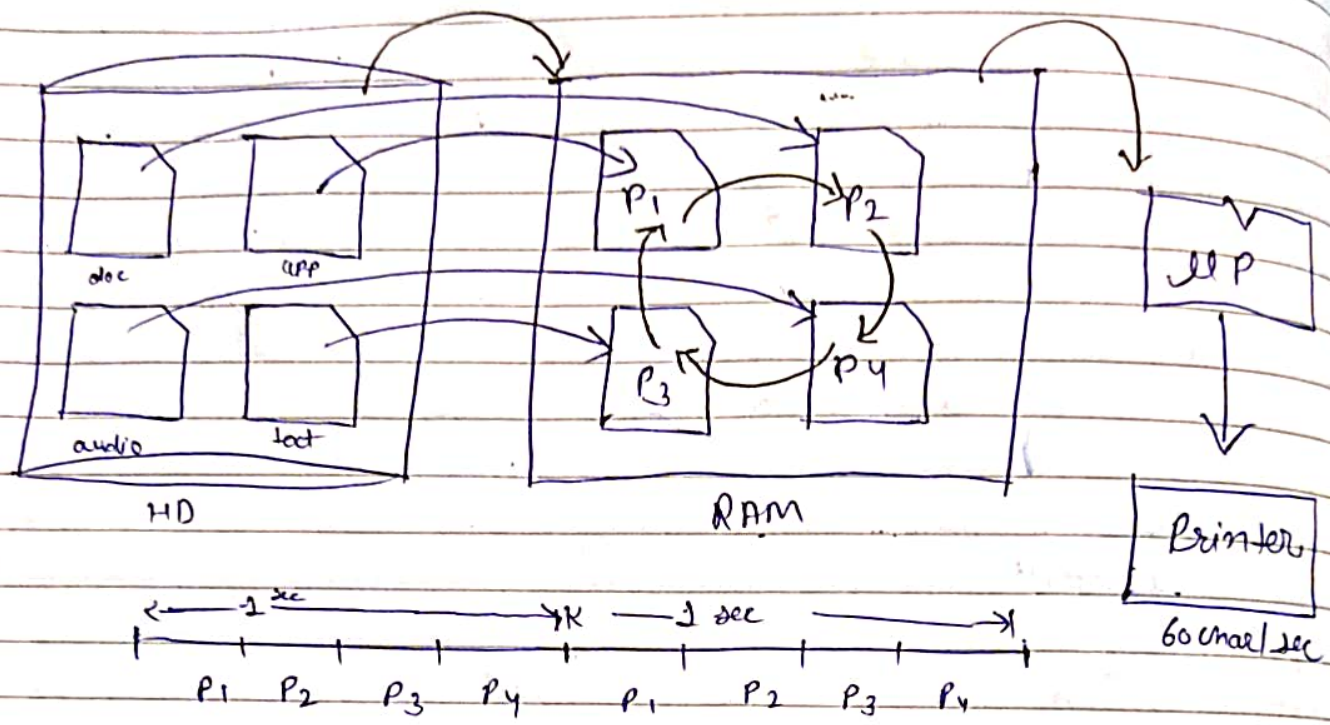
A D AAAAA

```
res = re.sub ( pattern, replace, str, count = 2 )
print (res) → DDAA
```



task / process : the program under Excution

* multitasking :



So, many process run simultaniously is known as multitasking

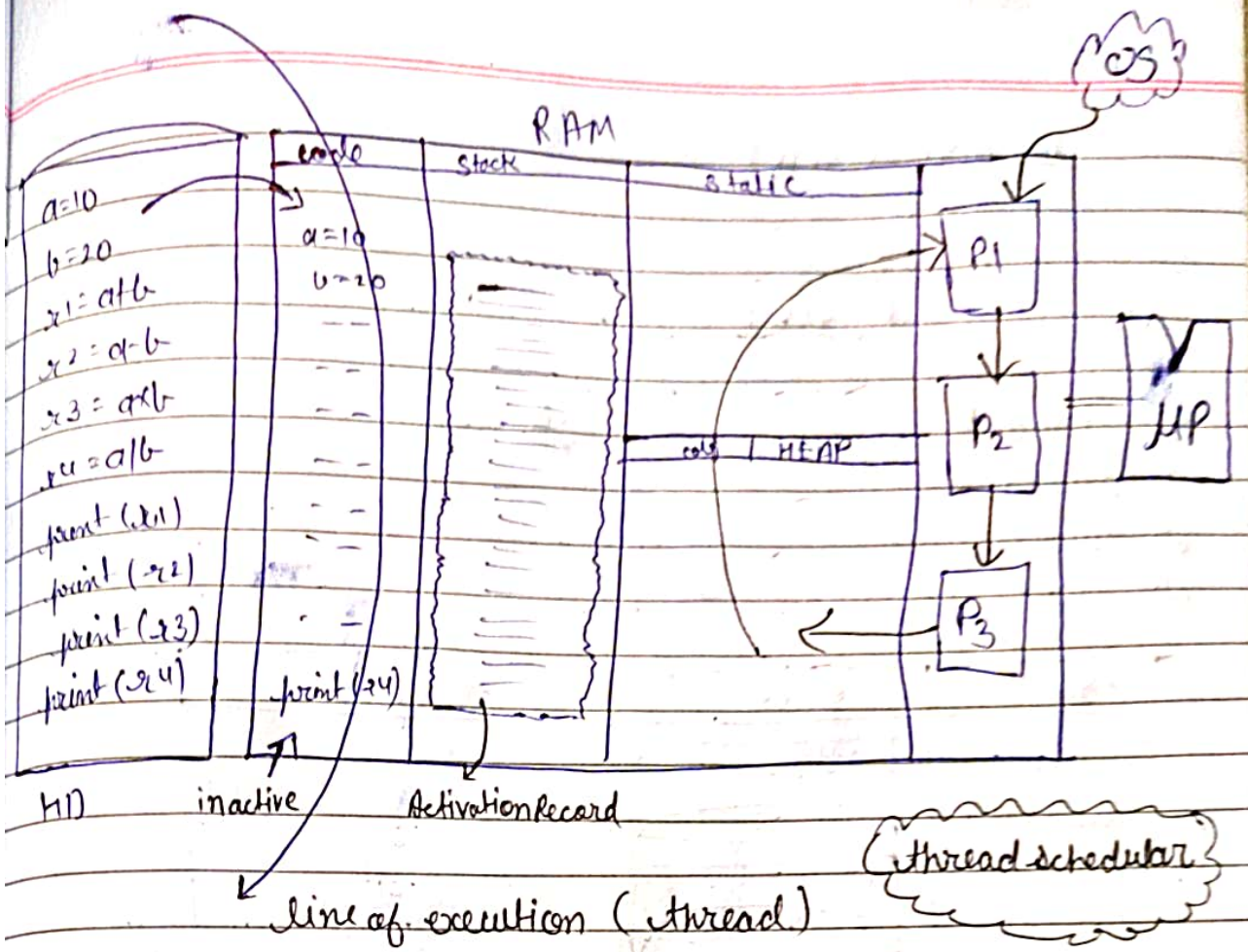
multi → multiple
tasking → process running

→ during process schedule context switch take place which is managed by OS.

the process of executing multiple process simultaniously is called as multi tasking

the process of switching from one task to another task is called as context switching. context switching is taken care by the OS.

→ How time should be allocated to each of the process is decided by OS.



→ the line of execution of a program is also called as a thread

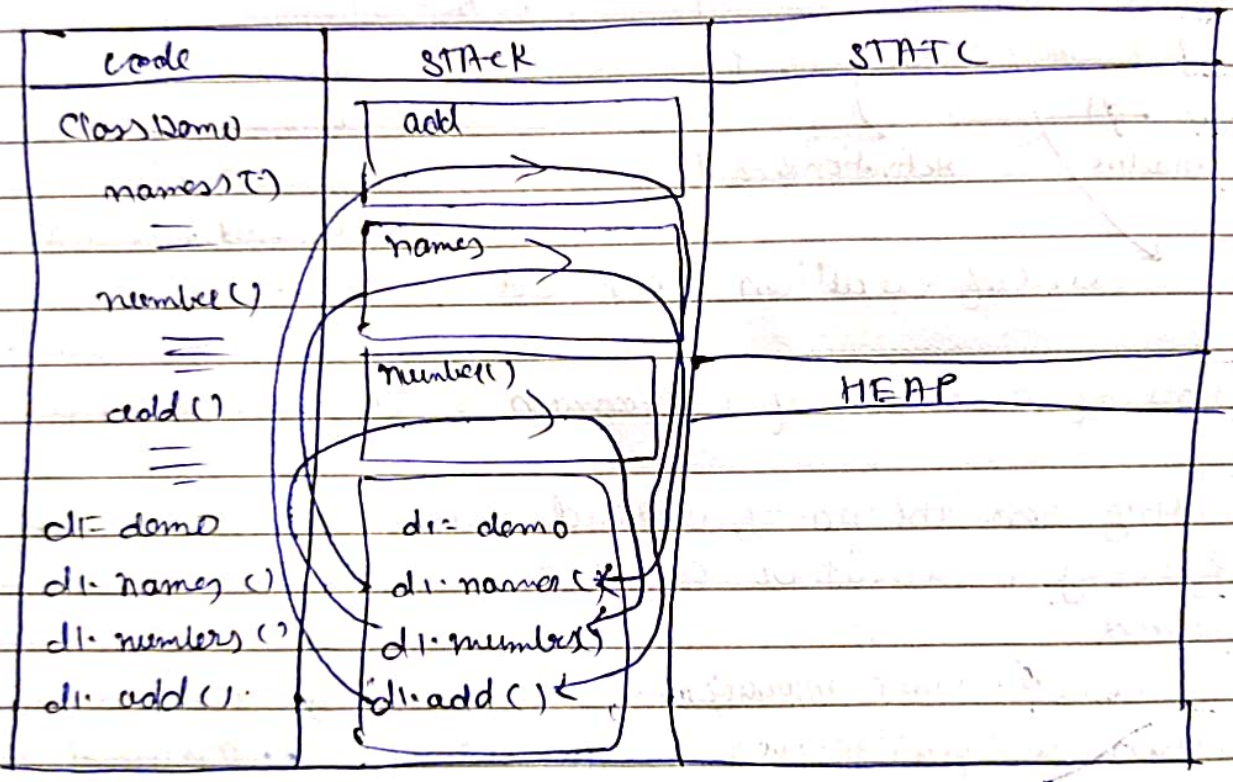
→ depending on the no. of thread in the program we can specify, wheather a program is single or multi threded

→ the overall time management is done by OS.

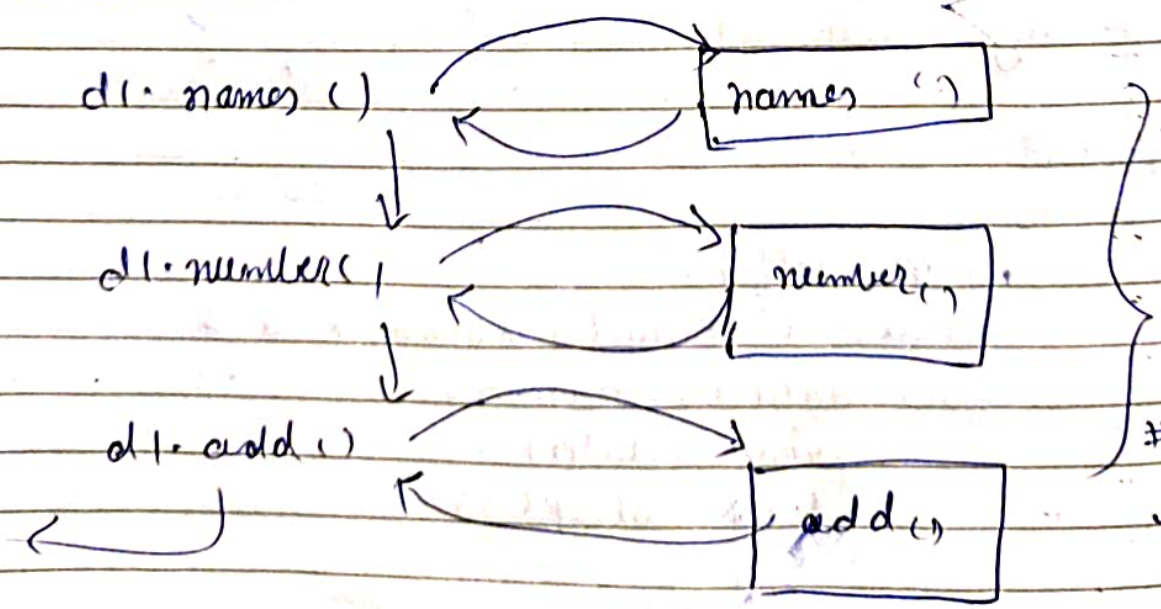
→ within a particular process the time management is done by " threat schedular".

```
import time
class Demo :
    def names(self):
        name = ['Rahul', 'Rohan', 'Rita']
        for data in names:
            print (data)
            time.sleep(3)

    def number (self):
        for i in range (10):
            print(i)
            time.sleep(3)
```

```
def add (self):
    a=10
    b=20
    print (a+b)

d1= demo()
d1.names
d1.numbers
d1.add
```

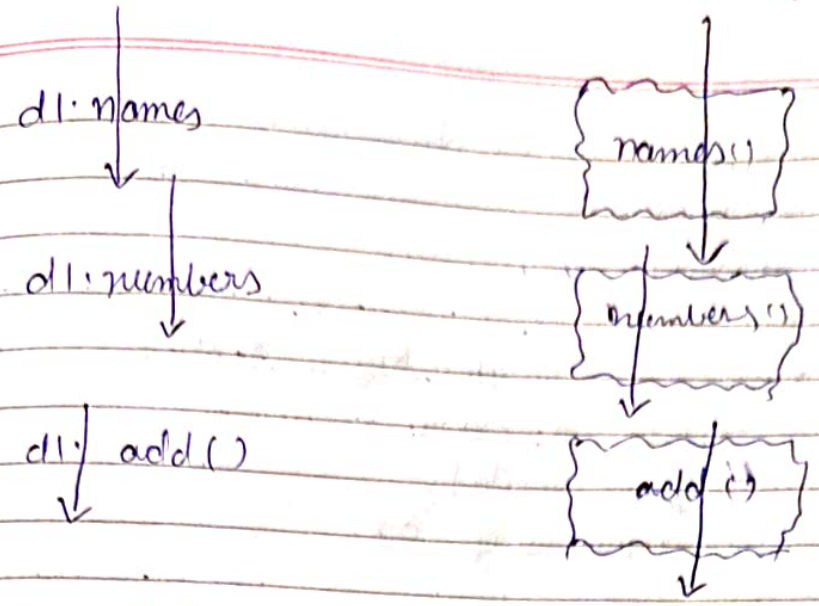


Before



ie seprate line of execution

after



So here we have 4 line of Execution so it is multi threded program

In given program printing name, no, adding 2 no. are independent task. as per best programing practics independent task must be executed simultaniously. this can be done by giving a seprate line of execution for every independent task.

⇒ Steps involving in converting a single thread program into multi threded program

(i) all the independent task in the program must be placed within run method.

(ii) place the independent task within the run method inside a class that extends (Inherit) the thread class.

(iii) create object of these classes.

(iv) call start method (start()) w.r.t each of these object.

start () $\xrightarrow{\text{internally}}$ run()

↳ A seprate stack gets created

```
import time
from threading import *
class task1 (thread):
    def run (self):
        names = ["Rahul", "Raj", "Rita"]
        for data in names:
            print (data)
            time.sleep (8)

class task2 (thread):
    def run (self):
        for i in range (1,10):
            print (i)
            time.sleep(2)

class task3 (thread):
    def run (self):
        a=10
        b=20
        c=a+b
        print (c)
```

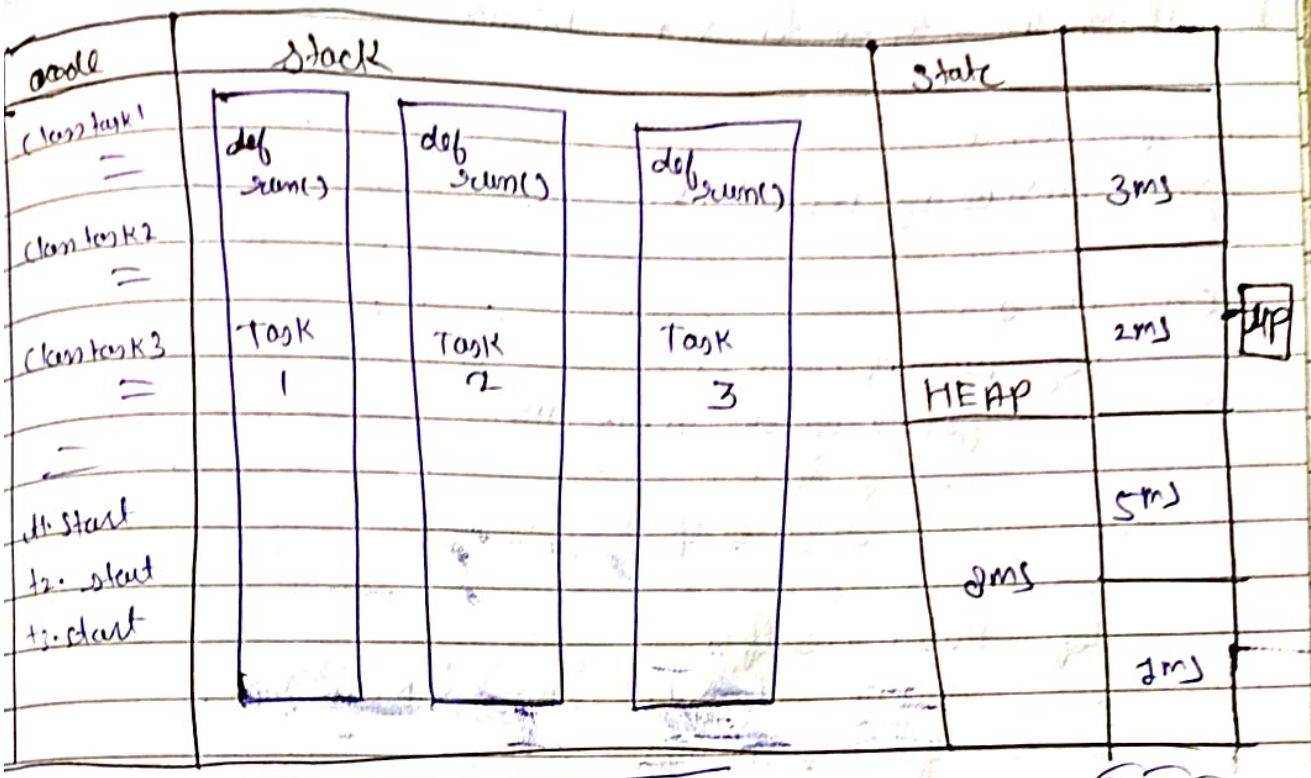
note: we can't predict order of output.

```
t1= task1()
t2= task2()
t3= task3()

t1.start()
t2.start()
t3.start()
```

Start method can create multiple stack.
run method can't.

if we use run() instead of start then program become single thread.

Mj multiple stack => multithread



3 Stack multithreading

TS

So, if we call run() instead of the start() then multiple stack ie multiple line of execution is not created & program behave as single thread program

→ calling the run method directly will not result in an error.

→ the thread object become active only after calling the start method.

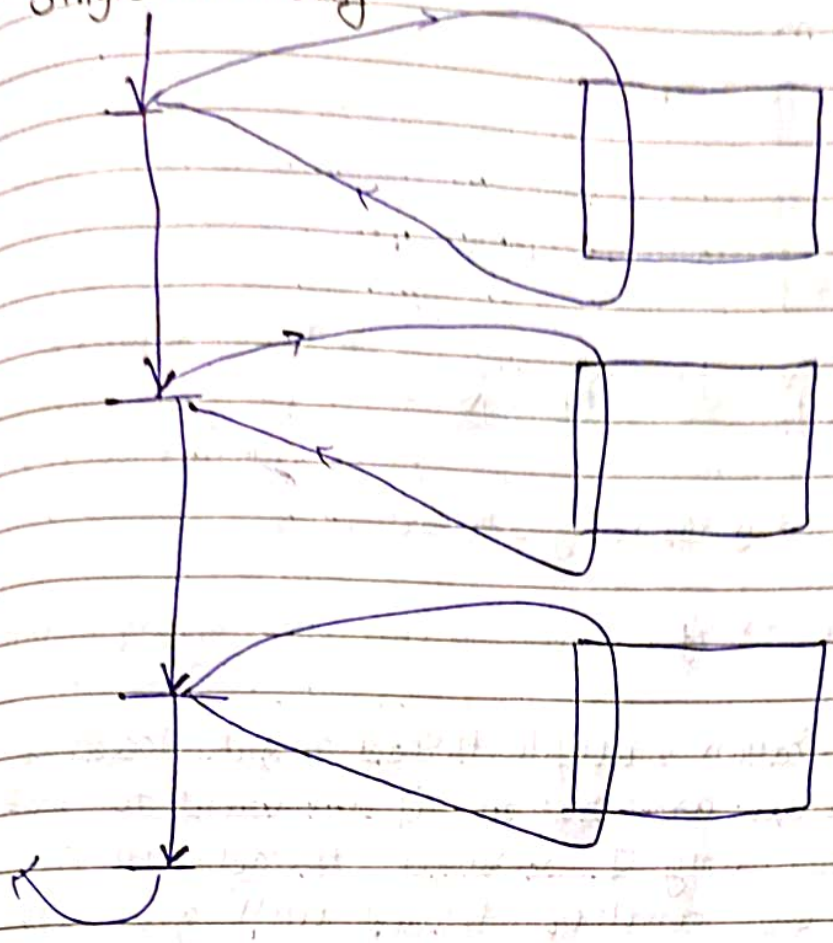
* Setting & getting name of a thread.

```
t1 = task1()
t2 = task2()
t3 = task3()

t1.setname('thread_printName')
t2.setname('thread_printno')
t3.setname('thread_odd')
current_thread().getName() → threadName
```

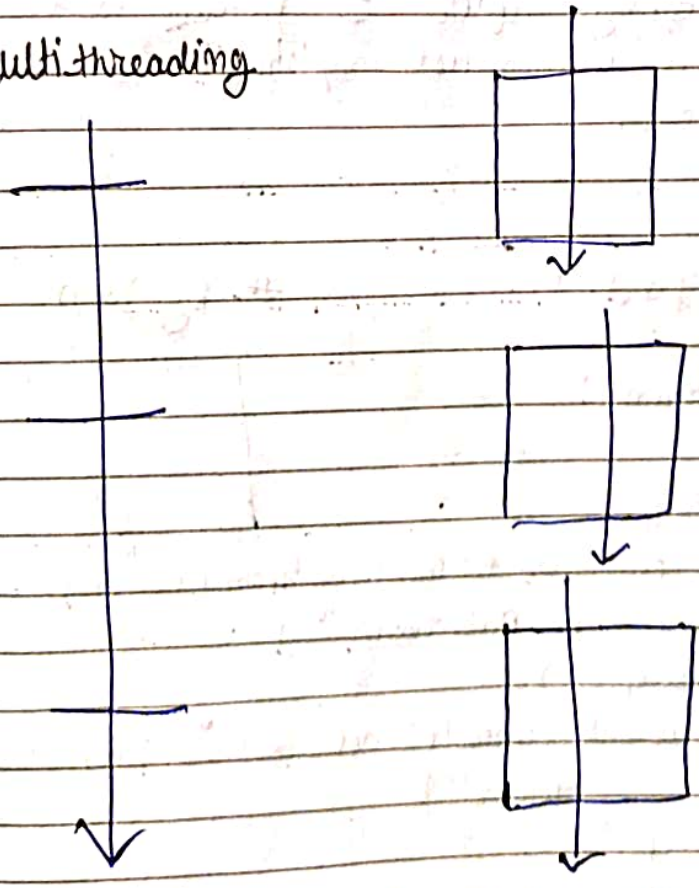
* Alternate method to create thread

```
import time
from threading import *

class Demo :
    def number (self):
        for i in range (1,11):
            print (i)
            time.sleep()

    def name (self):
        name = ['Rahul','Raj','Rita']
        for i in names
            print (i)
            time.sleep()

    def add (self):
        a,b = 10,20
        c = a+b
        print (c)
d1 = Demo

t1 = thread (target = d1.name , name = "print_name")
t2 = thread (target = d1.number, name = "print_number")
t3 = thread (target = d1.add, name = "thread.add")

t1.start()
t2.start()
t3.start()
```

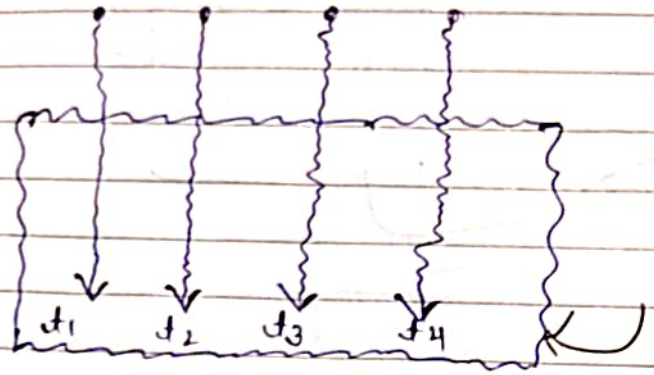
* Single threading



* multithreading



* Sychoronization



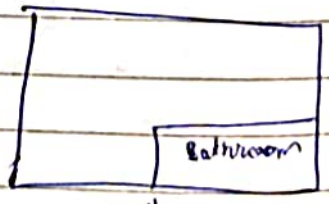
method / Resource / critical Resource / critical section

→ Join() } they allow only 1 thread
→ lock()
→ Semaphore() → if we want we can allow multiple thread

In Some application multiple thread might Access a single resource or method if we want to make sure at a time only 1 ~~resource~~ thread will access the resource or another thread will give access to resource only after 1 thread is completely executed, then we must implement concept of Synchronization.

```
import time
from thread import *
```

By join

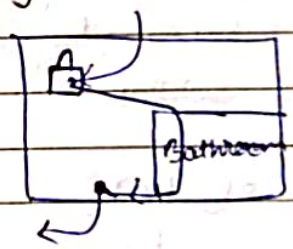



```
Class House (Thread):

    def run (self):
        print ( current_thread ().getName(), "has entered
                in Bathroom " )
        time . sleep (2)
        print ( current_thread().getname(), " has closed the
                door")
        time. sleep(3)
        print ( current_thread(). getname, " has taking Bath")
```

```
H1 = house()
H2 = House()
H3 = House()
H1.setName("Boy1")
H2.setName("Boy2")
H3.setName("Boy3")

H1.start()
H1.join()
~~H2.join()~~ H2.start()
H2.join()
H3.start()
H3.join()
```

```
import time
from threading import *
Class House(thread):
    l = lock()
    def run(self)
        House.l.acquire()
        print(current_thread().getname(), " has
                    entered in Bathroom")
        time.sleep(5)
        print(" current thread().getname(), " has closed the
                    door")
        time.sleep(3)
        print(" current_thread().getname(), " is taking
                    Bath")
        House.l.release()

H1 = House()
H2 = House()
H3 = House
H1.setName("BOY1")
H2.setName("BOY2")
H3.setname("BOY3")
H1.start()
H2.start()
H3.start()
```

By LOCK()



```
import time
from threading import *
Class House (thread):
    l = semaphore()        / l = semaphore (2)
    def run (self):  House.l.acquire()
        print ( current_thread().getname(), " entered in Bathroom")
        time.sleep(2)
        print( current_thread().getname(), " closed the door")
        time.sleep(3)
        print ( current_thread().getname(), " is taking Bath")
        House.l.release()

h1 = House()
h2 = House()
h3 = House()
h1.setname("Bay1")
h2.setname("Bay2")
h3.setname("Bay3")

h1.start()
h2.start()
h3.start()
```

if we want to allow 2 thread.

END