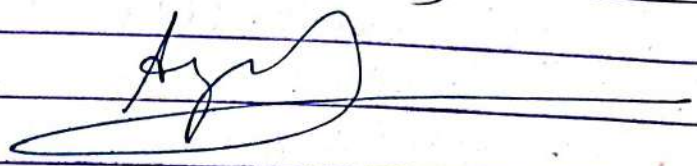
Date : / /
Page :

Reference : ADC ( )
DRF ( )

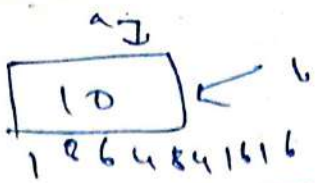


Python

```
>>> num = 5
>>> id(num)
   1864841536
>>> name = 'navin'
>>> id(name)
   93293952
>>> a = 10
>>> b = a
>>> a
   10
>>> b
   10
>>> id(a)
   1864841616
>>> id(b)
   1864841616
```

Same address

If variable in python have data, it point to same address. So memory efficient language.



Date : / /
Page :

```
>>> id(10)
1864841616
>>> k=10
>>> id(k)
1864841616
>>> a = 9
>>> id(a)
1864841600   (diff address)
>>> id(1)
1864841616
>>> k=a
>>> id(k)
1864841600
>>> b = 8
```

Now, no reference for 10 so it is ready for garbage collection.

```
>>> pI = 3.14
>>> pI
3.14
>>> pI = 3.15
>>> pI
3.15
```

So, constant value can be changed in python.

```
>>> type(pI)
<class 'float'>
```

Date : / /
Page :

when nothing assigned to variable its 'none'
none → null

Numeric → ① int
② float
③ complex
④ bool

```
>>> a = 5.6
>>> b = int(a)
>>> type(b)
<class 'int'>
>>> b
5
>>> k = float(b)
>>> k
5.0
>>> k = b
>>> c = complex(b,k)
>>> c
(5+6j)
>>> b < k
True
>>> bool = b < k
>>> type(bool)
<class 'bool'>
>>> int(True)
```

Two parameter (check passing one only).

Date : / /
Page :

1

```
>>> int (False)
0
```

Sequence

- List
- Tuple
- Set
- String
- Range

} Sequence

```
>>> list = [25, 26, 45, 12]
>>> type (list)
<class 'list'>
>>> s = {25, 12, 45, 14}
>>> type (s)
<class 'set'>
>>> t = (25, 26, 45, 50, 12)
>>> type (t)
<class 'tuple'>
>>> string = "nani"
>>> st = 'a'
>>> type (st)
<class 'str'>
```

Dictionary → mapping.

Date : __/__/____
Page : _____

```
>>> range(10)
range(0,10)        (0-9)
>>> list(range(10))
[0,1,2,3,4,5,6,7,8,9]
>>> list(range(2,10,2))
[2,4,6,8]
>>> type(range(10))
<class 'range'>
>>> d={'navin':'Samsung','Rahul':'Iphone'}
>>> d.keys()
dict_keys(['navin','rahul'])
>>> d.values()
dict_values(['Samsung','Iphone'])
>>> d['rahul']
'Iphone'
>>> d.get('Rahul')
'Iphone'
>>>
```

list(range(2,10,2)) → same → diff.

Operators
- Arithmetic
- Assignment
- Relational
- logical
- unary

Date : __/__/____

Page : _____

```
>>> x = 2
>>> y = 3
>>> x/y
0.6666666666666666
>>> x = x + 2 (assignment)
>>> x
4
>>> x += 2
>>> x
6
>>> x *= 3
>>> x
18
>>> a, b = 5, 6
>>> a
5
>>> b
6
```

Unary ops.

```
>>> n = 7
>>> n
>>> -n
-7
>>> n
7
>>> n = -n
>>> n
-7
```

Date : / /
Page :

Relational operators

```
>>> a, b = 5, 6
>>> a < b
True
>>> a > b
False
>>> a == b
False
>>> a = b
>>> a == b
True
>>> a <= b
True
```

Logical operator

```
>>> a = 5
>>> b = 4
>>> a < 8 and b < 5
True
>>> a < 8 and b < 2
False
>>> a < 8 or b < 2
True
```

comment

decimal - binary

decimal → base 10 (0-9)

Binary → base 2 (0-1)

octal → base 8 (0-7)

hexadecimal → (base 16) (0-9 a-f)

```
>>> bin(25)                    25 → 11001
'0b11001'
>>> 0b0101
5
>>> oct(25)
'0o31'
>>> hex(25)
'0x19'
>>> hex(10)
'0xa'
>>> 0xf
15
>>>
```

Swapping

```
a = 5
b = 6

a = a + b      # 11 → 4 bit
b = a - b      # 5  → 3 bit
a = a - b      # 6  → 3 bit
print(a)
print(b)
```

one extra bit wasted.

```
6
5
```

So, new method using XOR

Date : / /
Page :

```
a = a ^ b
b = a ^ b
a = a ^ b
```

or

a, b = b, a

↳ concept of notation

~12

↓

~00001100

~12 → 11110011

-13 → 00001101

(13)

2's compliment

↓

1's comp + 1

11110010

1

11110011 → -13

∴ -13 = ~12

~x = -(x+1)

↳ Bitwise operator

>>> 12 & 13

12

>>> 12 | 13

13

```
  00001100 → 12
& 00001101 → 13
  ---------
  00001100
    ↳ +2
1 00001101
```

XOR → if opp. then true

```
1 1 → 0
0 0 → 0
0 1 → 1
```

XOR → ^

```
  00001100
  00001101
^ 00000001  → 1
```

>>> 12 ^ 13
1

>>> 25 ^ 30
7

>>> 10 << 2
40 → gaining bits

1010 → 0101 → 1010

| 1010 | .000 |

| 101000.00 | 5.000 65.....

40. shifted bit 2 to left.

>>> 10 >> 2
2

1010. (loosing bits)
↓
10
↓
10 → 2

Calling Maths func

>>> x = sqrt(25)
error → ask for func first

>>> import math
>>> x = math.sqrt(25)
>>> x
5.0

>>> x = math.sqrt(15)
>>> x
3.872983346207417

Date : / /
Page :

⇒ 3 → ceil

2.5 ——————.

⇒ 2 → floor

floor (2.9) = 2
ceil (2.1) = 3

>>> print (math.floor (2.9))
2
>>> print (math.ceil (2.1))
3
>>> 3**2
9
>>> print (math.pow (3, 2))
9
>>> print (math.pi)
3.141592653589793
>>> print (math.e) e = epsilon.
2.718281828459045

>>> import math as m (alias)
>>> m.sqrt (25)
5.0
>>> math.sqrt (25)
5.0.

Date : __/__/____
Page : _____

importing only particular mathematical function

```
>>> from math import sqrt, pow
>>> pow(4,5)
1024.0
>>> help('math')
```

↓

Get func^c of math.

#(18) User input in python

```
X = input()
Y = input()
Z = X+Y
print(Z)
```

OR

```
X = input("Enter 1st No.")
Y = input("Enter 2nd No")
Z = X+Y
Print(Z)
```

↓

X = 9
Y = 5
output = 95 (input always give string type)

no 'char' in python.

Date : __/__/____
Page : _____

So

```
x = input()
a = int(x)
```
> `x = int(input())`

```
y = input()
b = int(y)
```
> `y = int(input())`

```
z = a + b
print(z)
```
`z = x + y`

Same output 14.

↓↓

x = 9
y = 5
z = 14 (output)

```
ch = input("Enter a char")
print(ch)
```

input → pqr (by mistake multiple chars)
output → pqr

↓

```
ch = input('enter a char')
print(ch[0])
```

↓↓

input → pqr
output → p

Same

or

`ch = input('Enter a Char')[0]`

Date : __/__/____
Page : _____

eval function - evaluates the expression

```
result = eval(input('enter an exp'))
print result
```

input ⟹ 2+6-1

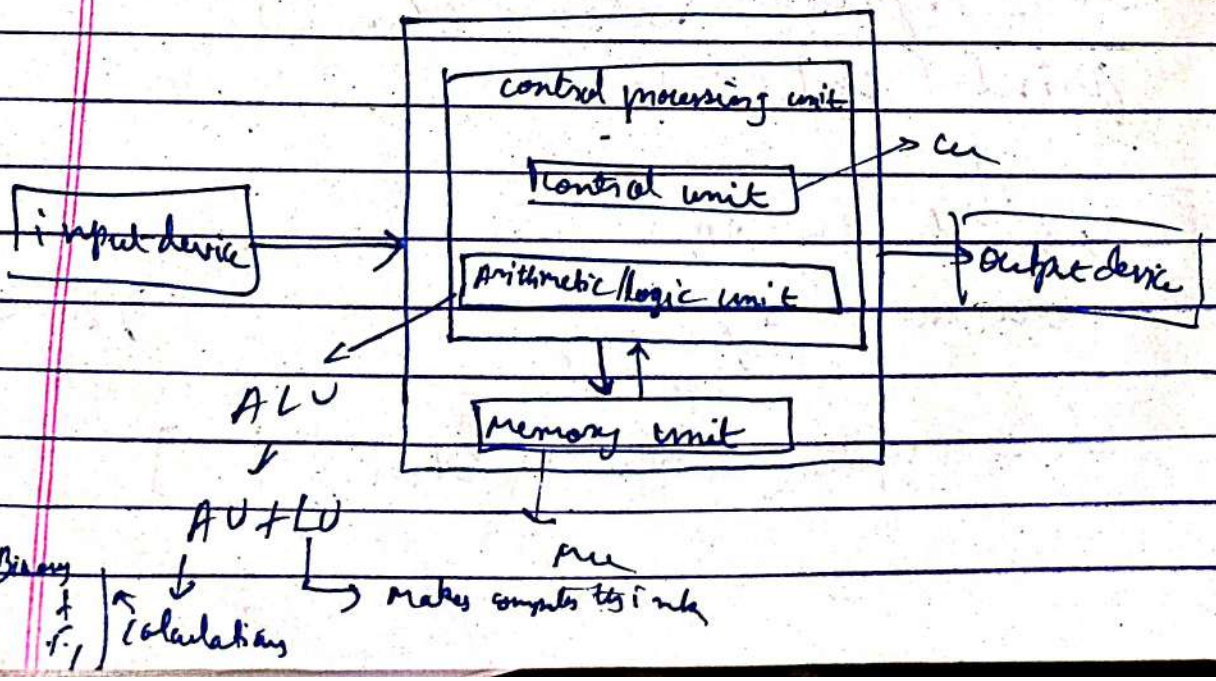
output → (7)

Taking value from commandline

~~(2) (50)~~

```
import sys
x = int(sys.argv[1])
y = int(sys.argv[2])
z = x+y
print(z)
```

Refer 18th video Telusko python last part.



Date : / /

Page :

1 tab = 4 spaces

IF - ELSE

If True:	If False:
print ("Im right")	print ("I am right")
↳ some space needed always	output as
~~not output~~ output → Im right	Not execute

↳ A type of indentation

```
x = 3
r = x % 2
if r == 0:
    print ("Even")
print ("Bye")
```

⇒ Bye

```
if r == 0:
    print ('Even')
else:
    print ('odd')
print ("Bye")
```

```
if x==0 ;
    print("Even")
    if x>5 :
        print('---')
    else :
        print("---")
else:
    print("----")
```

Nested if-else.

```
x = 1 :
if x==10:
    print("one")
elif (x==2):
    print("two")
elif(x==3):
    print("Three")
elif (x==4)
    print("Four")
else.
    print("many").
```

While Loop

```
i=1
while i<=5:
    print("Dynesh")
    i=i+1          (check for i++)
```

Date : / /
Page :

i = 5

while i >= 1 :
    print(" Ayush ", i)
    i = i - 1

Output
Ayush 1
Ayush 2
Ayush 3
Ayush 4
Ayush 5

# i = 5
j = 1
while i >= 1 :
    print(" Ayush ")
    while j <= 4 :
        print("Rocks")
        j = j + 1
    i = i - 1

For Loop (use with tuple, list, etc.)

x = ['navin', 65, 2.5]

for i in x :
    print(i)

Output
navin
65
2.5

x = ['navin']

output

N
A
V
I
N

eg:- for i in ['navin', 65, 2.5]
print(i)
same output

for i in range(10):
print(i)

0
1
2
3
4
9

for i in range(11, 21, 2):
print(i) excluded

11
13
15
17
19

for i in range(21, 11, -2)
print(i) excluded

21
19
17
15
13

for i in range(1, 21):
if(i%5 != 0):
print(i)

output
not print 5, 10, 15, 20

Break, continue, pass

Date : / /
Page :

```
if i > av:
    break
print(" -- ")--
```

pattern printing

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

```
for i in range(4):
    for j in range(4)
        print("#", end=" ")
    print()
```

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

```
for i in range(4)
    for j in range(i+1)
        print("#", end=" ")
    print()
```

For - Else

```
nums = [12, 15, 18, 21, 26]
for num in nums:
    if num % 5 == 0:
        print(num)
```

output = 15

```
nums = [10, 16, 18, 21, 26]
for num in nums:
    if num % 5 == 0:
        print(num)
        break
else
    print("not found")
```

⇒ output = 10

Date : / /
Page :

prime Number

```
num = 10

for i in range (2, num)
    if num % i == 0
        print("Not prime")
        Break
else
    print("prime")
```

Array

All values of same data type. we can expand / shrink its size as well as add elements.

```
import array
    or
from array import *
vals = array('i', [5, 9, 8, 4, 2])
print(vals)
```

output

```
array('i', [5, 9, 8, 4, 2])
```

Date : / /
Page :

```
from array import
vals = array.('i', [5, 9, -8, 4, 2])
print(vals.buffer_info())
```

Output

(59744056, 5) ← size of array
↳ address of array

```
print(vals.typecode)
```

Output

i → (integer)

print.

```
vals = array('i', [5, 9, -8, 4, 2])
vals.reverse()
print(vals)
```

Output

array('i', [2, 4, -8, 9, 5])

```
print(vals[0])
```

Output

5

```
for i in range(len(vals)):
    print(vals[i])
```

Output

5
9
-8
4
2

```
for ai in vals :
    print(i)
```

output
5
9
-8
4
2

Character array

```
from array import *
vals=array ('U', [a, ... i])
```

```
vals = array ('i', [5,9,8,4,2])
newArr = array( vals.typecode, [a for a in vals])
for e in newarray :
    print(e)
```

output
5
9
8
4
2

```
a*a for a in vals
```

output
25
81
64
16
4

* import all func

Input from users in array

```
from array import *
arr = array ('i', [ ])
n = int(input ("Enter the length of array "))
```

Date : / /
Page :

```
for i in range (n):
    x = int(input("Enter next value"))
    arr.append(x)

print(arr)
```

now searching the index of a particular element

```
k = 0
for e in arr:
    if e == val:
        print(k)
        break
    k += 1
```

or

```
print(arr.index(val))
```

↳ func.

Multidimension - Array

working with multidimensional array

```
- numpy
from numpy import *
arr = array([1,2,3,4,5])
print(arr)
```

↳ no need to write data type of array but if we want we can mention it

```
array([1,2,3,4,5], int)
```

Date : / /
Page :

Different ways of creating array in numpy

```
from numpy import *
arr = array([1,2,3,4,5.0])
print(arr.dtype)
print(arr)
```

=

float 64
[1. 2. 3. 4. 5.]
↳ all float

```
arr = array([1,2,3,4,5], int)
print(arr.dtype)
```

↓
int 32

```
arr = array([1,2,3,4,5], float)
print(arr.dtype)
print(arr)
```

↓
float 64
[1. 2. 3. 4. 5.]

=

```
from numpy import *
arr = linspace(0, 15, 16)
print(arr)
```

start, stop (included), No. of divisions in range.

Date : / /
Page :

[0. 1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 13. 14. 15.]

↳ float value because range is being divided, so, we might get float value.

eg

arr = linspace(0, 15)

↳ it will by default divide range into 50 parts.

arr = ~~array~~ arange(1, 15, 2)

start stop ↳ step

print(arr)

[1 3 5 7 9 11 13]

arr = logspace(1, 40, 5)

↳ 5 parts division

10^1 10^{40}

print('%.2f' % arr[0])

↳ 10.0

arr = zeroes(5)

↳ create array of size 5 each element 0. If ↳ float

You want work with int then

arr = zeroes(5, int).

Same with ones()

arr = ones(5) → ones(5, int) → [1 1 1 1 1]

↳ [1. 1. 1. 1. 1.]

Date : __/__/__
Page : _____

```
from numpy import *
arr = array([1,2,3,4,5])
arr = arr + 5        → add 5 to each element.
print(arr)
```

↳ [6, 7, 8, 9, 10]

Vectorised operation

```
arr1 = array([1,2,3,4,5])
arr2 = array([6,7,8,9,10])
arr3 = arr1 + arr2
print(arr3)          ↳ add both
```

↳ [7, 9, 11, 13, 15]

```
arr1 = array([1,2,3,4,5])
print(sin(arr1))
```

⇒ [0.84147098 0.90929 ———]

```
print(cos(arr1))
print(log(arr1))
print(sqrt(arr1))
print(sum(arr1))      = 15
```

↳ sum of all elements of arr

```
print(min(arr1))      → min of elements of array.
```

↳ 1

```
print(max(arr1))      → 5
print(sort(arr1))
```

```
arr1 = array([1,2,3,4,5])
arr2 = array([3,7,8,9,10])
print(concatenate[arr1, arr2])
```

↓

[1, 2, 3, 4, 5, 3, 7, 8, 9, 10]

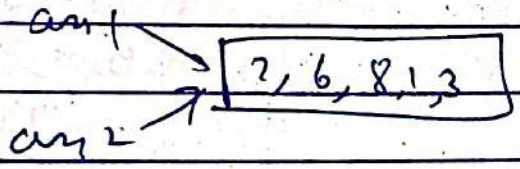
How to copy an array ???

```
arr1 = array([2,6,8,1,3])
arr2 = arr1
print(arr1)
print(arr2)
```

[2, 6, 8, 1, 3]
[2, 6, 8, 1, 3] > but we have only one array in our memory.

```
print(id(arr1))
print(id(arr2))
```

> same address



← Aliasing
both array pointing to same address.

If

```
arr2 = arr1.view()
```

↳ same array is copied at new address.

Date : / /
Page :

Shallow copy & Deep copy

```
arr1 = arr([2,6,8,1,3])
arr2 = arr1.view()
arr[1] = 7
print(arr1)
print(arr2)
```

shallow copy ⇒ [2 7 8 1 3]
[2 7 8 1 3]

But
if arr2 = arr1.copy()
then output

[2 7 8 1 3]
[2 6 8 1 3]

Changes only in 1st array

```
arr1 = array([
            [1,2,3],
            [4,5,6]
            ])
```

print(arr1) → [[1,2,3]
[4,5,6]]
print(arr1.dtype) → int32
print(arr1.ndim) → 2 (no. of dimensions)
print(arr1.shape) → (2,3)
↓ ↓
rows columns
print(arr1.size) → 6

Convert 2-D array into 1-D array

```
arr2 = arr1.flatten()
print(arr2)          → [1 2 3 4 5 6]
```

- Now,

```
arr1 = array([
                [1, 2, 3, 6, 2, 9],
                [4, 5, 6, 7, 5, 2]
                ])
arr2 = arr1.flatten()
arr3 = arr2.reshape(3, 4)
                    rows  columns
print(arr3)
[ [1 2 3 6]
  [2 9 4 5]
  [6 7 5 3] ]
```

Now if

```
arr3 = arr2.reshape(2, 2, 3)
print(arr3)
```

↓

```
[ [ [1 2 5]
    [6 2 9] ]
  [ [4 5 6]
    [7 5 3] ] ]
```

⇒ one big 3D array in which 2 2-D array are present

Date : __/__/____
Page : _____

```
arr1 = array([
        [1,2,3,6],
        [4,5,6,7]
        ])
m = matrix(arr1)
print(m)
```

```
[[1 2 3 6]
 [4 5 6 7]]
```
→ output is same but here you can perform more operations.

or

```
m = matrix('1 2 3 6; 4 5 6 7')
print(m)
```
→
```
[[1 2 3 6]
 [4 5 6 7]]
```

Suppose

```
m = matrix('1 2 3; 6 4 5; 1 6 7')
print(diagonal(m)) → [1 4 7]
print(m.min()) → 1
print(m.max()) → 7
```

If two matrix

m_1

m_2

$m_3 = m_1 + m_2$ (matrix adding)

or $m_3 = m_1 * m_2$ (matrix multiplying)

print("x = ", 2)

x = 2

Date : / /

Page :

Functions

```
def greet():
    print("Hello")
    print("good morning")

greet()
```

output:

Hello

Good morning

```
def add(x,y):
    c = x+y
    print(c) / return c

add(5,4)        ⟶ 9
```

In Python we don't use call by reference & call by value. In Python int & strings are immutable so we create a new copy at new address and so original value don't gets changed. But when we talk about list which is mutable, we can change its value in its actual address. so, its actual value gets changed.

#Refer Vidio 33

```
def add(a,b):
    c = a+b
    print(c)
```

→ Formal arguments

```
add(5,6)
```

→ actual arguments

Actual Arguments

① Keyword Argument

```
def person(name, age):
    print(name)
    print(age-5)

person(age=28, name='Ayush')
```

This helps when we don't know actual sequence and take help of keywords (here name & age).

② Default argument

If not passing age, by default it should be 18 (etc.).

```
def person(name, age=18):
    print(name)
    print(age)
```

→ try not passing age

person ('navin', 28)

output : navin
28 → overwrites.

Variable length argument

```
def sum (a, *b):
    c = a
    for i in b:
        c = c+i
    print (c)
sum (5, 6, 34, 78)
```

go in b as tuple.

or

```
def sum (*b):
    c = 0
    for i in b:
        c = c + i
    print (c)
```

Keyword variable length argument.

```
def person(name, **data):
    print(name)
    for i, j in data.items():
        print (i, j)
```

two arguments

person ('Ayush', age = 28, city = 'Mumbai', mob = 123)

Date : __/__/____
Page : _____

Scope

```
a = 10  (global 'a')
def something ():
    a = 15
    print("in fun", a)
```

} This 'a' local to func

```
something()
print("outside", a)
```

output
in fun 15
outside 10

But

```
a = 10
def something():
    print(a)

print(a)
```

| we can use value of 'a' (global variable) in func.

```
10
10
```

```
a = 10
def something()
    global a
    a = 15
    print("a")
something()
print(a)
```

→ 15
15

Date : / /
Page :

```
a = 10
print (id(a))
def something():
    a = 9
    x = globals()['a']
    print(id(x))
    print("in fun", a)
    globals()['a'] = 15

something()
print("outside", a)
```

```
2659598424
2659598424
in fun 9
outside 15
```

passing list in func

```
def count(lst):
    even = 0
    odd = 0
    for i in lst:
        if i%2==0
            even+=1
        else:
            odd+=1
    return even, odd
lst = [10, 25, 14, 19, 16, 24, 28, 47, 26]
```

Date : __/__/____
Page : _____

even, odd = count(lst)
print("Even: {} and odd: {}".format(even, odd))

Even: 6 and odd: 3

Fibonacci:

```
def fib(n):
    a = 0
    b = 1
    print(a)
    print(b)
    for i in range(2, n):
        c = a + b
        a = b
        b = c
        print(c)
```

fib(5)

0
1
1
2
3

Factorial

```
def fact(n):
    f = 1
    for i in range(1, n+1):
        f = f * i
```

Date : / /
Page :

return f

```
x = 4
result = fact(x)
print(result).
```

Recursion in Python

```
import sys
print(sys.getrecursionlimit())
```

1000 (output)

put any number for change limit.

Python supports
1. Functional programming
2. OOP
3. procedure oriented programming

Function in OOPs called 'method'.
class → like blueprint of object.

~~Class~~

Object & Class

Attributes → variables
Behaviours → methods (functions)

Date : __/__/____
Page : _____

```
class computers:

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

com1 = computers()
computer.config(com1)
```

output

i5, 16gb, 1TB

Q)

```
com1 = computers()
com2 = computers()
computer.config(com1)
computer.config(com2)
com1.config()
com2.config()
```

i5, 16gb, 1TB
" " "
" " "
" " "

```
class computers:
    def __init__(self, cpu, ram):
        self.cpu = cpu
        self.ram = ram

    def config(self):
        print("config is", self.cpu, self.ram)
```

Cpu = i5
Ram = 16
com1 (object)

Cpu = Ryzen
Ram = 8
com2 (object)

com1 = computer('i5', 16)
com2 = computer('Ryzen 3', 8)

com1.config()
com2.config()

config is i5 16
config is Ryzen 3 8

Heap memory → contains all objects
Every time you create an object, it is allocated to new space.
Size of object depends on no. of variables & size of each variable. constructor allocates size of object.

Anonymous Functions

As we pass value to func, we can also pass func to func as func are also object in Python.

```
def square(a):
    return a*a
```

} we have to write def, etc for single line.

```
result = square(5)
print(result)
```

25 (output)

Date : / /
Page :

f = lambda a : a * a
result = f(5)
print(result)

f = lambda a, b : a+b
result = f(5,6)
print(result)

we can pass n arguments but it should end up in one statement (i.e, here a+b)

Methods

class method
static method
Instance methods
} same in variable but diff in method.

class student:

    school = 'Telusko'
    def __init__(self, m1, m2, m3):
        self.m1 = m1
        self.m2 = m2
        self.m3 = m3
    def avg(self):
        return (self.m1 + self.m2 + self.m3)/3

s1 = student(34, 45, 34)
s2 = student(89, 92, 12)

Instance → accessors (fetch value of inst. var)
use self keyword → mutators (change value of inst. var.)

class → 'cls' use keyword

Print (s.avg())

```
def get_m1(self):
    return self.m1

def set_m1(self, value):
    self.m1 = value

def info(cls):
    return cls.school

print(student.info())
```

→ to get output we have to pass 'cls' but we don't want to do so,

```
@classmethod
def info(cls):
    return cls.school
```

How it works.

```
@staticmethod
def info():
    print("This")
```

→ pass nothing & for static variable.

```
student.info() => # This
```

$11 \times 23 I_2 + 23 I_3 = -60$

$23 \times 11 I_2 + 20 I_3 = 110$

Class in class

$253 I_2 + 460 I_3 = -660$

$253 I_2 + 40 I_3 = 220$

```
class student :
    def __init__(self, name, rollno):
        self.name = name
        self.rollno = rollno
        self.lap = self.laptop()
    def show(self):
        print(self.name, self.rollno)

class laptop:
    def __init__(self):
        self.brand = 'hp'
        self.CPU = 'i5'
        self.ram = 8

s1 = student('Nancy', 2)
s2 = student('Jenny', 3)
```

$213 I_3 = 880$

$I_3 = \frac{880}{213}$

$I_3 = 4.13$

$2 I_2 + 23 \times 4.13 = -60$

-77.498

s1.lap → different ids

You can create object of inner class inside the outer class

OR

You can create object of inner class outside the outer class provided you use outer class name to call it.

$4I_2 - 3I_3 - 2\left[\frac{3I_2+10I_3-6}{2}\right] I_1 = 40$

$4I_2 - 3I_3 - 6I_2 - 20I_3 + 12 = 40$

$-2I_2 - 23I_3 = 60$

$\frac{11}{2} I_2$

Inheritance

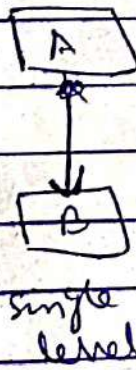
```
class A:
    def feature 1 (self):
        print ("Ayush")

class B:
    def feature 2 (self):
        print ("shaurya")

class C:
    def feature 3 (self):
        print ("Isha")
```

C is child class of A & B.

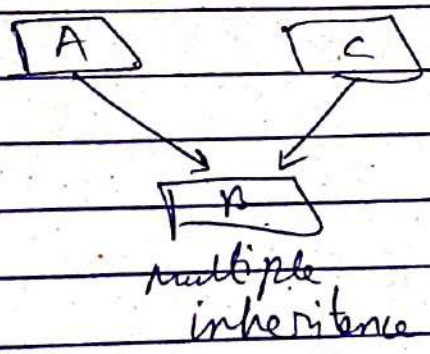
c1 = C()
c1. any feature

=> If class C(A, B):
    inherits all feature from super class
If class B(A):
    inherits all features from 'A'.

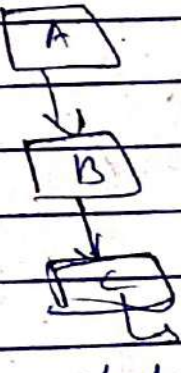
child inherits all features from parents.



single level



multiple inheritance



C has feature of both 'a' & 'b'.

Multi level inheritance.

super class can't access features from sub class as parent can't from children but child can.

Date : __/__/____
Page : _____

Constructor in Inheritance

```
class A:
    def __init__(self):
        print("in A init")
    def feature1(self):
        print("Ayush")
class B(A):
    def feature2(self):
        print("Ajay")
a1 = A()
b1 = B()
```

Output

in A init
in A int

Now if 'init' func in B then, output
in A init
in B init.

So, firstly 'init' of B would be searched, then if not found, then init of 'A' would be printed.

Date : __/__/____
Page : _____

But, if we want to get output of init of both A & B then,

```
class B(A):
    def __init__(self):
        super().__init__()
        print("in B init")

    def feature2(self):
        print("Shambhya")

b1 = B()
```

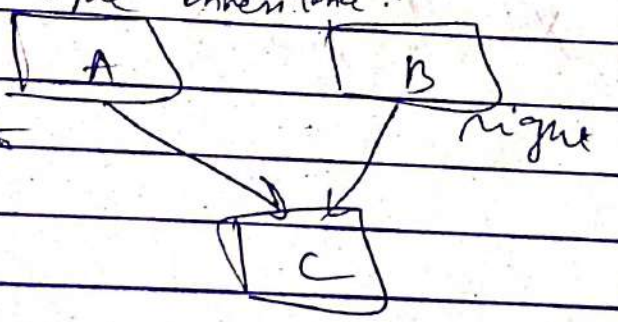
'init' gets feature of superclass.

output

in A init
in B init

(A, B)

Now if C's class also there and we call super().__init__ then init of A would be taken as the concept of MRO (Method resolution order) is from left to right in case of multiple inheritance.



Date : / /
Page :

Polymorphism

poly → many
morph → form
ism

It means that one thing can take multiple forms.

Polymorphism

- Duck Typing
- operator overloading
- method overloading
- method overriding

Duck Typing

```
class ayush:
    def king(self):
        print("Ayush shaurya she is Gangster bandi")
class tha:
    def x(self, g):
        g.king()
g = ayush()
t = tha()
t.x(g)
```

Output

Ayush shaurya she is gangster bandi

operands

5 + 6

operator

Date : / /

Page

Operators overloading :-

a = 5

b = 6

print (int.__add__(a, b)) → This is called when we do (a+b).

print (a+b)

||

||

% a = '5' and b = '6' in same code

output

'56'

'56'

class student :

def __init__ (self, m1, m2):

self.m1 = m1

self.m2 = m2

def __add__ (self, other):

m1 = self.m1 + other.m1

m2 = self.m2 + other.m2

s3 = Student(m1, m2)

return s3

s1 = student(58, 69)

s2 = student(60, 65)

s3 = s1 + s2

print(s3.m1)

118 (output)

Iterators

Date : __/__/____
Page : _____

```
nums = [7, 8, 9, 5]
it = iter(nums)
print(it.__next__())
print(it.__next__())
print(it.__next__())
```

7
8
9

if we do print(next(it)) after this then we will get

7
8
9

Lets create our own iterators.

```
class Topten():
    def __init__(self):          (as initialization)
        self.num = 1
    def __iter__(self):          (as a counter)
        return self
    def __next__(self):          (as a next element)
        if self.num <= 10:
            val = self.num
            self.num += 1
            return val
        else:
            raise StopIteration
values = Topten()
print(next(values))
for i in values:
    print(i)
```

1
2
3
4
5
6
7
8
9
10

Date : / /
Page :

Generators → gives Iterators

```
def topTens():
    yield 5

values = topTens()
print(values)
```

<generator object topTens at ...>

return 5 instead of yield 5 then output is 5.

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

Now output :-

```
yield 1
yield 2
yield 3
yield 4
```

Print(values.__next__()) only one so, output 1.

Generator → iteration without iterator

```
def toptens():
    n = 1
    while n <= 10:
        sq = n * n
        yield sq
        n += 1
Values = toptens()
for i in Values:
    print(i)
```

1
4
9
16
|
|
|
100

Error

- Syntactical error (syntax error)
- Logical error (wrong output)
- Runtime error (eg: dividing by zero)
- mistake by user

```
a = 5
b = 0
try:
    print(a/b)
except Exception:
    print("No division by zero")
print("Bye")
```

→ It will give no errors as I am due to 'try'.

Date : __/__/____
Page : _____

```
try
a = 6
b = 0
print(a/b)
except Exception as e:
    print("my a", e)
```

Output

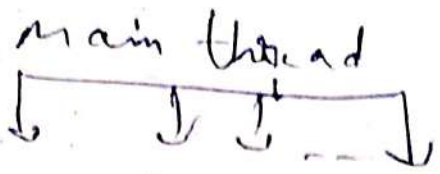
My division by zero

Compiler message of error

```
a = 5
b = 2
try:
    print("resource open")
    print(a/b)
    k = int(input("enter"))
    print(k)
except ZeroDivisionError as e:
    print("my")
except ValueError as e:
    print("invalid input")
except exception as e:
    print("something")
finally:
    print("resource closed")
```

Threading

using multiple thread to make our system work fast.



Date : / /
Page :

```
from time import sleep
from threading import *

class Hello(Thread):
    def run(self):
        for i in range(5):
            print("Hello")
            sleep(1)
class Hi(Thread):
    def run(self):
        for i in range(5):
            print("Hi")
            sleep(1)
t1 = Hello()
t2 = Hi()
t1.start()
sleep(0.2)
t2.start()
t1.join()
t2.join()
print("bye")
```

Hello
Hi
Hello
Hi
Hello
Hi
Hello
Hi
Hello
Hi
bye

File Handling

```
f = open('my data', 'r')
f1 = open('abc', 'w')
f2 = open('xyz', 'a')
```

Date : __/__/____
Page : _____

```
for data in f:
    print data
```

output

Ayush
Corporate Trainer
Abc.

we want to read from mydata and add to abc

```
for data in f:
    f1.write(data)
```

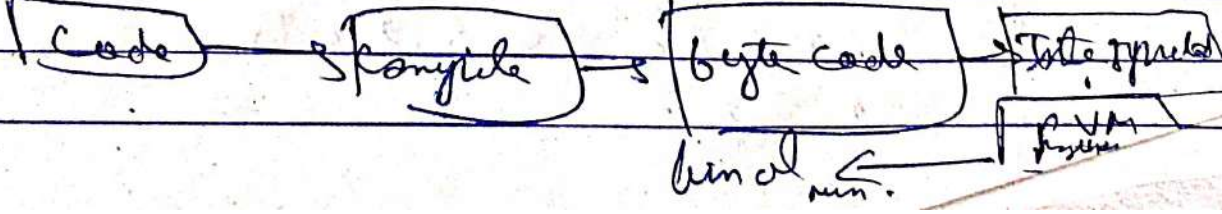
So all data from 'mydata' will be written in file 'abc'.

for image copying

```
f = open('Img-6309.jpg','rb')
f1 = open('my-pic.jpg','wb')
for i in f:
    f1.write(i)
```

For comment we use '#', if multiline then use # on every line. we can do by """ also. (triple quotes).
use # only.

python → compiled & Interpreter both.



Date : / /
Page :

Linear search (values can be in any order)

```
def search(list, n):
    i=0
    while i < len(list):
        if list[i] == n:
            return True
        i=i+1
    return False

list = [5, 8, 4, 9, 2]
n = 9
if search(list, n):
    print("Found")
else:
    print("Not Found")
```

~~Line on Zero~~

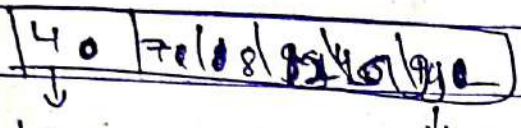
```
pos = -1

def search(list, n):
    i=0
    while i < len(list):
        if list[i] == n:
            globals()['pos'] = i
            return True
        i=i+1
    return False
list = [5, 8, 4, 6, 9, 2]
n = 9
```

to make it global

Date : __/__/____
Page : _____

```
if search(list, n):
    print("Found at", pos)
else:
    print("Not found")
```

Binary search
(values must be sorted)



mid of index = (L+U)/2
= 0(0+5)/2
= 2

mid value = 8

Check if mid value = element search

```
def search(list, n):
    l = 0
    u = len(list) - 1
    while l <= u:
        mid = (l+u)//2
        if list[mid] == n:
            globals()['pos'] = mid
            return True
        else:
            if list[mid] < n:
                l = mid + 1
            else:
                u = mid - 1
    return False
```

Date : / /
Page :

Bubble sort

back order

```
def Sort(nums):
    for i in range(len(nums)-1, 0, -1):
        for j in range(i):
            if nums[j] > nums[j+1]:
                temp = nums[j]
                nums[j] = nums[j+1]
                nums[j+1] = temp
```

zip function

```
names = ("Navin", "kiran", "harsh")
comps = ("Dell", "Apple", "MS")
zipped = list(zip(names, comps))
print(zipped)
```

↓

```
[('Navin', 'Dell'), ('kiran', 'Apple'),
 ('harsh', 'MS')]
```

We can use

```
zipped = set(zip(names, comps))
```

or dict

↙ but we guarantee that output will maintain order but if we have repeated values in names & comps then also it will give some value.

Date : / /
Page :

or

```
for (a, b) in zipped:
    print(a, b)
```

Same output

Socket programming

TCP (Transmission control protocol) & UDP (User datagram protocol)

TCP:
- ↓ connection oriented protocol
- ↓ First has to create connection
- Main benefit is that we can get sure that whether packet has reached there or not.

UDP:
- ↓ Don't have to get connection
- ↓ Just have to send the packets. Based on network address it will simply reach to particular destination.

See photos in phone.