PYTHON

TOKENS:

Smallest Individual unit of a program:

-> keywords -> Identifiers -> Literals

-> operators -> punctuators

-> literals

eg:

name = "Anya"

Identifier

print (name , "agency" , "coordinated")

keyword

punctuators

+ , - , % } -> operators

>> TEXT = 'hello world'

output :-

>> 'helloworld'

>> TEXT = 'helloworld IAM U'

output :-

>> 'helloworld IAM U'

operators :-

ARITHMETIC operators :-

-> + Addition

-> * multi

-> % modulus

-> ** power

-> // floor div

-> - sub

-> / div

Bitwise operators :-

& AND

^ XOR

| OR

shift operators

<< shift left

>> shift right

Identify operator :-

-> is is the identify same?

-> is not is the identify not same?

Relational operators	Logical operators	membership operators
<	and	in
>	or	not in
<=		
>=		
==		
!=		

* multiple assignments:-

```
a = b = c = 10

x, y, z = 10, 20, 30
```

* type function:-

```
>> a = 10
   type(a)
<class 'int'>

>> a = {10}
   type(a)
<class 'set'>
```

* type casting;

```
agi = 16.7      agi
mark = int(16.7)
print(mark)
```

output

```
16
```

* memory address:

```
P = 5
id(P)
1245766208
```

↳ address of P.

Is operators:

a=25	
b=25	
a is b	a is not b
True	False

but,

>> S1 = 3.5

>> k = float (input ("enter a value: "))
>> Enter a value : 3.5
>> k == 1

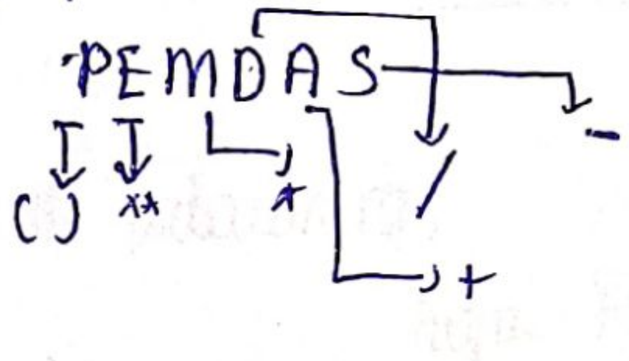
True

because, they not store in same memory location.

>> k is 1

False

[operator precedence]



NOTE:-

* if we use '/' divis operator always return the value in float

* modules :-

import <module name>

↓

at 1st line of the program,

Ex:- * math module
* random module
* statistics module etc...

math modulus :-

		return value
ceil,	math.ceil(num)	in float
sqrt,	math.sqrt(num)	
pow,	math.pow(base, exp)	
sin,	math.sin(num)	
etc....		

statistics.

example :- import statistics.

seq = [5, 6, 7, 5, 6, 5, 5, 9, 11, 12, 23, 5]

statistics.mean(seq) = same for median, mode

>> 8.25

Now,

FLOW of control :-

Syntax :-

```
if condition :
        statement 1
elif condition :
        statement 2
else :
        statement 3
```

[nested if also there like C program]

* Looping :-

(i) for i in range(start, end[exclusive], step):
        print(i)

[membership operators]

also, [variable] [String or may be list any other],

(ii) for i in 'calm':
        print(i)

* While Loop :-

while condition :
        statement
        incrementation

* break statement,
        helps to break or end the loop.

---

* New method of print:

a = [1, 2, 3, 4]
print(f"a = {a}")

format printing

String manipulation:- (Immutable)

single quotation marks, double quotation and triple quotation marks are known as string.

operations:-

"tea" + "pot" } string with only string in addition operators.

>> teapot

now, replication:-

>> "tea" * 2 } with number

>> teatea

* membership operators:-

>> "a" in "hya"

>> True

>> "123" not in "12345"

>> False

* Comparison operators:-

"abc" == "abc" True

"Abc" != "abc" True

now,

* character function:-

>> chr(65)

'A' } -> string type [result]

>>> chr(97)

'a'

String slicing:-

*

	0	1	2	3	4	5	6
word =	a	m	a	z	i	n	g
	-7	-6	-5	-4	-3	-2	-1

now, → exclusive

→ word [0:3] = 'ama'

→ word [-7:-3] = 'amaz'

also,

→ word [:7] = 'amazing'

→ word [5:] = 'ng'

now,

→ word [1:6:2] = 'mzn'

→ word [-7:-3:3] = 'az'

→ word [::-2] = 'giaa'

→ word [::-1] = 'gnizama'

reverse slicing,

if:-

```
S = "Hello"
print (S[4:8])
print (S[5:10])
```

} → one string is outstand it's boarder

output:-

0

↳ empty string for 2nd print ! [not raise any error]

String functions:-

1. len() [len(IDENTIFIER string)]

```
>> len("hello")
6
>>> name = "marie"
>>> len(name)
5 → integer as result
```

2. capitalize() [string.capitalize()]

```
>>> 'true'.capitalize()
True
>>> 'i love world'.capitalize()
'I love world'
↳ caps only 1st letter.
```

3. count() [string.count('___', ___, ___)]

- ___ (1st): required string
- ___ (2nd): initial index to
- ___ (3rd): final index
- characters

```
>>> 'abracadabra'.count('ab')
2
>>> 'abracadabra'.count('ab', 4, 8)
0
>>> 'abracadabra'.count('ab', 6)
1
```

(Start: I; end: I; onwards: I)

4. find() [<string>. find(sub[, start [, end]])

>>> string = 'it goes var - ringa ringa roses'

sub = 'ringa'

string. find(sub)

>> 13

string. find(sub, 15, 22)

>> -1 }(if the sub string not found)

5. index() [<string>. ~~find~~ index('sub', start, end)]

index..

>>> 'IAM b'. index('A', 0, 8)

>> 1

-X- [if the sub string not found it raise error]

6. isalnum, isalpha, isdigit. [<string>. is__()]

function

[result in Boolean value] [True / False]

alpha + num,

ex:-

>> '123abc'. isalnum()

>> True

>>> '123'. isalnum()

>> True

Gaph not allowed for all the True.

7. isdigit(), isupper(), isspace only, ;
   isdigit() isupper() replace()
   all must be lower same as lower
   [result in boolean value][True/False]

8. lower(), upper() [string.lower()]
   change all alphabet in lower.

```
>>> 'Abc123'.lower()
'abc123'
```

   change all alphabet in upper()

```
>>> 'abA123'.upper()
'ABA123'
```

9. [.strip(); .lstrip() .rstrip() [<string>.strip()]
   remove space both left & right
   remove space in left
   remove space in right.
   return copy of a string.

```
>>> " IAm ".strip()
'IAm'
>>> " IAm ".lstrip()
'IAm '
```

10.) startswith( ), endwith( ) [<string>.startswith()]

[returns boolean value] [True / False]

-> "abcd".startswith("ab")

-> True

-> "abcd".endwith("cd")

=> True

11.) title( ) :- , istitle( )

(i) return a string in titlecased, i.e the first character in caps and remaining character in lower -case.

>> "COMPUTER SCIENCE".title()

>>> "Computer Science"

(ii) istitle( ) [returns in Boolean value],

12.) replace( ) :- [<string>.replace(old, new)]

>>> 'abcabc'.replace('ab', 'sp')

>> 'spcaspc'

13.) split():- [<string>.split(sub)]
                    ↳ not necessary,

>>> "I Love python".split("o")

>> ['I L', 've Pyth', 'n']

[the occurence of 'o' is removed]
↳ result in list

14.) partition():- [<string>.partition("sub")]
↳ result in tuple and always the resulted length is 3,,

txt = " I enjoy working in PYTHON"

x = txt.partition("working")

print(x)

output:-

('I enjoy ', 'working', ' in PYTHON')

* list manipulation: (mutable)
→ changeable

elements stored in square brackets '[ ]'.

example:-

```
L = [1, 2, 'IAM']
```

* nested list

```
L = [[1,2], 'IAM', ['U', 'A']]
```

* list():-

```
l1 = list('hello')
>> ['h', 'e', 'l', 'l', 'o']
```

* getting input from user:-

```
l1 = list(input("Enter list elements:"))
Enter elements: 2345
>> [2,3,4,5]
```

also,

```
l1 = eval(input("enter:"))
>> Enter: [67, 78, 46, 23]
l1
>> [67, 78, 46, 23]
```

Note eval() is used to evaluate the input from user,

eg:

```
eval('3+5')
>> 8
eval('3*10')
```

Comparing operations:-

L1 = [1,2,3]

L2 = [1,2,3]

L3 = [1,2]

>> L1 == L2

True

>> L2 == L3

False

also,

1 < 9

[1,2,3,9] < [9,1] = True

[1,2,9,4] < [1,2,8,6] = False

9 ≱ 8

Check: Sequentially;;

* addition:-

[1,2,5] + [3,4,5]

>> [1,2,5,3,4,5]

* repeating:-

[1,2,3] * 2

* Slicing operations :-

seq = L[start:stop]

lst = [10, 24, 14, 20, 22, 24, 30]

>> seq = lst [3:-3]

>> seq

[20, 22, 24]

also,

Seq[1] = 28

>> [20, 28, 24]

if L[10:20]

↳ out of bounds

return []

↳ empty list instead of error.

* True copy of a string.

L = [1, 2, 3]

b = L

L

>> [1, 2, 3]

b

>> [1, 2, 3]

also,

L[1] = 5

>> [1, 5, 3] = L

>> [1, 5, 3] = b } changes

True copy

but

a = [1, 2, 3]

b = a.copy()

a

[1, 2, 3]

a[0] = 5

[5, 2, 3] = a

[1, 2, 3] = b } not changes

not true copy.

Slicing modification :-

* L = ["one", "two", "THREE"]

L[0:2] = [0,1]

L

[0, 1, "THREE"]

Now,

* L[0:2] = "a"

L

["a", "THREE"]

* L[2:] = "604"

L = [1,2,3]

[1, 2, '6', '0', '4']

* List functions :-

1) len() :-

L = [1,2,3]

>> len(L)

>> 3

2) list() :-

>> list((1,2,3))

[1,2,3]

>> list({'a':10, 'b':102})

['a', 'b']

form a list from keys

3) Index() :-

list.Index(<item>)

>> L1 = [13, 18, 11, 16]

L1.Index(18)

1

[raise error if the item not found]

4. append() :-

List.append(<item>)

```
c = ['red', 'grun', 'blu']
c.append('black')
c
>> ['red', 'grun', 'blu', 'black']
```

5. extend() :-

list.extend(<list>)

```
l1 = ['a', 'b', 'c']
l2 = ['d', 'e']
l1.extend(l2)
l1 = ['a', 'b', 'c', 'd', 'e']
```

but,

```
l2 = ['d', 'e'] -> remains same.
```

6. insert() :-

list.insert(<ind>, <item>)

```
l1 = ['a', 'e', 'u']
l1.insert(2, 'i')
l1
['a', 'e', 'i', 'u']
```

moved to index 3

inserted at index 2

[valid for negative index]

7. pop() :-

List.pop(<Index>):

*L1 = [1, 2, 3]

L1.pop()

L1

[1, 2]

*L1.pop(0)

L1

[2]

8. remove() :- [List.remove(<item>)]

L1 = [1, 2, 3, 4]

L1.remove(4)

L1

>> [1, 2, 3]

[remove first ocurrence]

[if item not found raise error]

9. clear() : [L.clear()]

L1 = [1, 2, 3]

L1.clear()

>> L1

>> [] = empty list

10. count() :- [List.count(<item>)]

L1 = [13, 18, 20, 18]

L1.count(18)

>> 2

L1.count(28)

>> 0

11. reverse() :- [List.reverse()]

t3 = L.reverse()

not save in t3

but, L is reversed,

12. sort(), sorted() :-

L.sort(reverse = F/T)

t = sorted(L, reverse = F/T)

can save by secondary identifier

does not save in any identifier.

14.7 min(), max(), sum() :-

L1 = [17, 24, 15, 30]

min(L1) = 15

max(L1) = 30

sum(L1) = 86

Comments in python:-

* Single line comments :- # is used for it
* double line comments :- ''' ___ ''' triple quots is used for multiline comments.

Tuples manipulation :- [Immutable] [Non-changeable]

```
T= (1, 2, "one")
type (T)
<class tuple>
```

also,

```
>>T= 3,
>> T
(3, )
```

↳ form tuple

```
>>T=(3)
>>T
 3
```

↳ not form tuple just integer.

nested tuples:-

```
t1= (1, 2, (3, 4))
```

* tuple function:-

```
>> t1= tuple ('hello')
>> t1
('h', 'e', 'l', 'l', 'o')
```

[use eval() to enter the perfect tuple as user convinent]

accessing tuples:-

→ access tuples by index:-

-5	-4	-3	-2	-1
a	e	i	o	u
0	1	2	3	4

Slicing same as string & ~~tuple~~ list..

Tuple operations:-
(same as list)

Tuple functions:-

* len()

T= (2, 3, 4)

len(T)

>> 3

* max()

T= (2, 3, 4)

max(T)

>> 4

* min()

T= (2, 3, 4)

min()

>> 2

* sum()

L= (27, 34, 25, 40)

sum(L)

>> 126

* index()

L= (3, 4, 5, 6)

L.index(5)

>> 2

* count()

t= (2, 4, 2, 5, 7, 2)

>> t.count(2)

>> 3

* sorted() [sorted('val', reverse=T/F)]

t= (27, 34, 25, 40)

val = sorted(t)

>> val

[27, 34, 25, 40] -) returning list

Dictionaries:-

dict1 = { 1:"A", 2:"B", 3:"C" }

were, dict 3 = {[2,3], "abc"}

It raise unhashable Type : 'list'

↓

type error

∴, The key only contain tuple like immutable types.

not mutable types.

keys:- [can be]

* string,
* number,
* tuple

Accessing elements of a dictionary:-

* d = {"vowel1":"a", "vowel2":"e", "vowel3":"i", "vowel4":
"o", "vowel5":"u"}

>> d["vowel1"]

'a'

Characteristic of a Dictionary:-

1. unordered set:-

A dictionary is a unordered set of key: value pairs.

2. Not a Sequence:-

A dictionary is not a sequence. because its unordered set of elements.

3. Indexed by keys:-

not like string, list or tuples.

4. Keys must be unique:-

5. Dictionaries are mutable:-

[It can changable]

Create a dictionary:-

(i) E = dict (name = 'John', salary = 10000, age = 24)

↑ key not enclosed by quotes

>> E

{'salary': 10000, 'age': 24, 'name': 'John'}

(ii) E = dict (zip(('name', 'salary', 'age'), ('John', 10000, 24)))

↑ keys

↑ values

>> E

{'salary': 10000, 'age': 24, 'name': 'John'}

(iii) E = dict ([['name', 'John'], ['salary', 10000], ['age', 24]])

>> E

{'salary': 10000, 'age': 24, 'name': 'John'}

* Nested dictionary:-

E = {'John': {'age': 25, 'salary': 20000}, 'Riya': {'age': 35}}

>> E['John']['age']

>> 25

* checking in dictionary:-

empl = {'salary': 1000, 'age': 2, 'name': 'baby'}

>> 'age' in empl

True

>> 'age' not in empl

False

* pretty printing in Dictionary. [import module Json]

>> print (json.dumps (winners, indent = 2))

indentation = 2

Dict name

2 spaces

{

"Nihar": 3,

"Rohan": 5,

"Zebra": 3,

"James": 5

}

dictionary functions :-

* get() [<dict>.get(key, [default])]

empl = {'salary': 10000, 'dept': 'sales', 'age': 24}

>> empl.get('dept')

'Sales' = value as output,,

[note: if the key is not in dict then, it raise error]

So,

>> empl.get('dsisg', "Not yet")

'Not yet'

if not present
['default' values]

* items2() [<dict>.items2()]

empl = {'name': 'John', 'salary': 10000, 'age': 24}

myList = empl.items2()

for i in mydict:
    print(x)

[out:-
('salary', 10000)
('age', 24)
('name', 'John')]

[note:-
output in dict of tuples with key value pair]

* keys2():- [<dict>.keys()] values():- [<dict>.values()]

emp.keys2()

['salary', 'dept', 'age'] -> return in list

emp.values()

['10000, 'sales', 24]

* fromkeys() :- [dict.fromkeys(<key-seq>, [<value>])] *

nd = dict.fromkeys([2,4,5,6], 100)

{2:100, 4:100, 5:100, 6:100}

also,

dict.fromkeys((2,3))

{2: None, 3: None}

note:-
only tuple or list not integer
↑ [keys]

also;

dict.fromkeys((3,))

{3: None}

* setdefault() :- [dict.setdefault(<key>, <value>)]

>> m = {1:100, 2:200, 3:300}

m.setdefault(4, 400) = [return the value], 400

{1:100, 2:200, 3:300, 4:400}

m.setdefault(4, 420)

400

{1:100, 2:200, 3:300, 4:400}

[it not change because it already exist.]

m.setdefault(5) ← only key

{1:100, 2:200, 3:300, 4:400, 5: None}

↳ None as assigned.

* The update() :- [<vdict>.update(<other-dict>)]

e1 = {'name': 'John', 'salary': 1000, 'age': 24}

e2 = {'name': 'Diya', 'salary': 54000, 'dept': 'sales'}

e1.update(e2)

>>e1

{'salary': 54000, 'dept': 'sales', 'name': 'Diya', 'age': 24}

>>e2

[ ] not change.

* copy [shallow copy, deep copy] : [<dict>.copy()]

dict = {1:100, 2:200, 3:300}

dict2 = dict.copy()

dict3 = dict

} two types of copy,

[It not change if also, the dict is change.,]

[It change if dict is change]

[note:-

for more over dict copy() logic also,

[over.. pg 470 11th book]]

* Deleting elements from dict :- [clear(), pop(), popitem(), del]

1. pop() :- [<dict>.pop(key, <value>)]

>> stu

{1: 'Neha', 2: 'saima', 3: 'Avnit'}

>> stu.pop(3)

>> 'Avnit' [return value]

>>> stu.pop(6, "Not there") ← if not found the default value,

'Not here' ← because it's not in dict()

2. popitem() method: [<dict>.popitem()]

>> stu

{1: 'Neha', 2: 'saima', 3: 'Avnit'}

stu.popitem()

>> (3, 'Avnit') => value returned in tuple, (key, value),

[Note:-
last key: value removed and returned]

3. clear() method:- [<dict>.clear()]

E = {1: 100, 2: 200}

E.clear()

>> E

{} → dict remains.

del <dict>

E = {1: 100, 2: 200}

del E

E

[raise error as
E is not defined]

* sorted() :- [sorted(<dict>, [reverse = False])]

stu={41: 'Neha', 12: 'saima', 32: 'Arnit'}

```
>>> lst = sorted(stu)
->lst
[12, 32, 41]
```

, remain unaffected.

* maximum, minimum and sum():-

[max(dict), min(<dict>), sum(<dict>)]

d3 = {41: 'N', 12: 'S', 32: 'A', 24: 'C'}

max(d3)

>> 41

min(d3)

>> 12

} maximum and minimum key

sum(d3)

, applicable for dictionary contain integer or float as a key

FUNCTIONS:-

* <dy> <functionname>(____, ____): ← parameters

    # statements

    return _value_ or [print (value)],,

* Python function types:-

    * Buil-in function :- [like, sum(), print()]

    * Functions defined in modules :- [like, math.sin()]

    * user defined functions :- [dy by the programmer]

* example:-

    [function name] ←

    dy sum(x, y): ← formal parameters

    [keyword]

        S = X + y

        return S

        [return the value]

main program.

print (sum (5, 10))

actual parameters.

four type:-

* no return type, no parameter
* no return type, with parameter
* NO parameter, with return type
* ~~return~~ parameter, with return type with

dy interest (prin, cc, time = 2, rate = 0.09):
    return prin * time * rate

Function call statement — legal / illegal

* interest (prin = 3000, cc = 5) — legal
* interest (rate = 0.12, prin = 5000, cc = 4) — legal
* interest (cc = 4, rate = 0.12, prin = 5000) — legal
* interest (5000, 3, rate = 0.05) — legal
* interest (rate = 0.05, 5000, 3) — illegal
* interest (5000, prin = 300, cc = 2) — illegal
* interest (5000, prinamt = 300, cc = 2) — illegal
  ↳ undefined
* interest (300, time = 2, rate = 0.05) — illegal..

* returning value :-

```
dy sum(x, y):
    return x+y
```

```
—main—
b = sum(5,3)
└——, it contain 8 ,,
```

```
-> dy greet():
       print("hello z")

a = greet()
print(a)
```

---

hello z
none
└, unexpected output.
} output.

* scope of variables:-

-> global scope:-

Is a variable defined in the 'main' program (__main__section). such variable are global scope.

-> Local scope:-

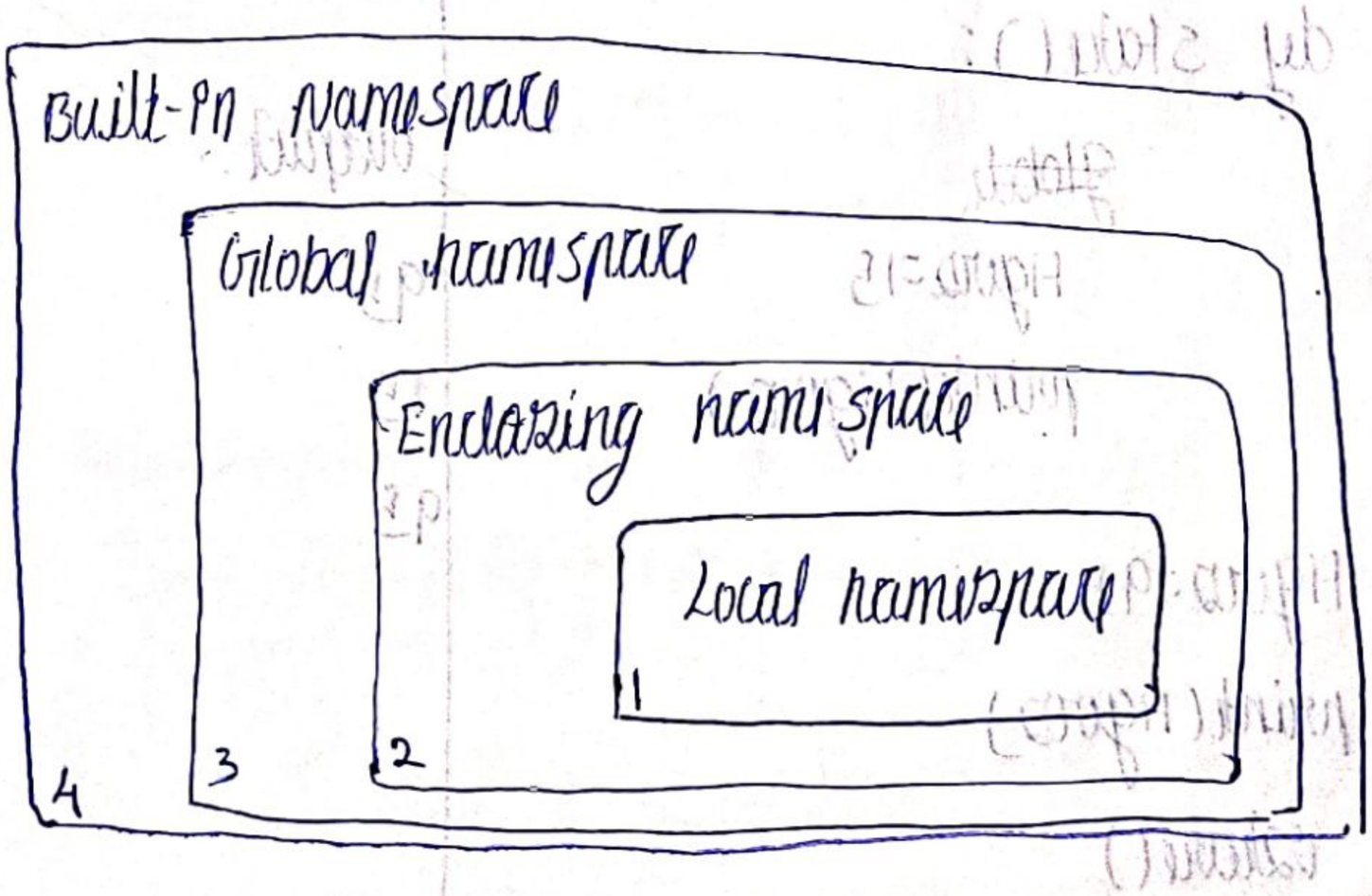
Is a variable defined within a function.

such variables are said to have local scope.

example:-

```
def calcsum(x,y):
    z = x+y } -> local scope
    return z

num1 = int(input("enter the num:"))
num2 = " "  "  "  } = global scope
sum = calcsum(num1, num2)
print("sum:", sum)
```

commonly:-



```
def calcsum(x,y):
    print(num1)  —> it will print because, it's global scope
    return s

num1, num2 = eval(input("enter num:"))
print("sum:", calcsum(num1, num2))
```

if we use: [global <variable num>]

1.
```
def state():
    global figure
    figure2=15
    print(figure2)

figure2=95
print(figure2)
state()
print(figure) —> the value changed,,
```

output:-

95
15
15

2.
```
def state():
    ~~global~~
    figure2=15
    print(figure2)

figure2=95
print(figure)
state()
print(figure) —> not change..
```

output:-

95
15
95

* dy num(a):

    b=5

    a=b

    return a

x = int(input("enter num:"))

print(x)

print(num(x))

print(x) -> not changed..

```
enter num: 5
6
5
6
```

* dy num(a):

    a=a+2

x = int(input("enter num:"))

print(x)

print(num(a x))

print(x) -> value changed..

```
enter num: 5
5
7
7
```

File handling:

A file in itself is a bunch of bytes stored on some storage device like hard-disk, thumb-drive etc...

* Data files:-

- -> Text files = store in form of ASCII values
- -> Binary files = store in form of stream of bytes

-) two types :-

-> regular text files

-> delimited text files

-(*) Tab separated values (TSV

->(*) comma separated values (CSV

opening and closing files:-

-) * <file_object name> = open (<file-name>, <mode>)

- <file_object name> -> file object.
- <file-name> -> path to file
- <mode> -> default, read mode

example:-

* file2 = open ("data.txt", "r")

also, * file2 = open (r"c:\temp\data.txt", "r") = legal

* file2 = open ("c:\temp\data.txt", "r") = ~~now~~ wrong..

-> prefix, r which is means raw string which ignore the special meaning of characters

-, were, it result, it because no prefix r ..

-) * <file_handle name>.close()

eg:

file2.close() ..

Access modes:-

Txt file	Binary file	
'r'	'rb'	——→ raises I/O error if file not exist
'w'	'wb'	——→ create file, if not and if exist, delete all
'a'	'ab'	——→ create file, if not and returned any appn at last.
'r+'	'rb+' (or) 'r+b'	same but including reading as common.
'w+'	'wb+', 'w+b'	
'a+'	'ab+', 'a+b'	

* WORKING WITH TEXT FILE:-

1. read()

<File handle>.read([n])

how many bytes

if n is no specified read entire file and store in the form of string.

2. readline()

<file handle>.readline([n])

how many bytes in the line

if n is specified read almost n bytes in the line.

result in form of string with '\n' at the end.

3. readlines()

<file handle>.readlines()

it store in form of list of strings every line with '\n' at the end.

using function strip() we can remove leading and trailing spaces along with new line character.

for program examples:- refer 12th summita vareera

pg: 190 - 193.

* writing onto a text files:-

1. write()

<file-object>.write('Stri')

2. writelines()

<write_lines>.writelines()
↳ list of lines.

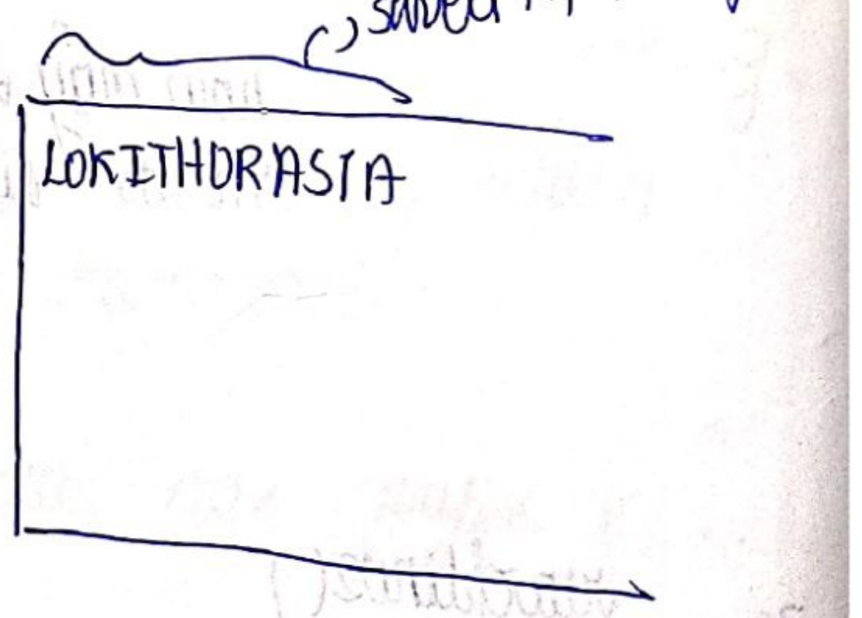
file object.

example:-

```
f = open("student.txt", "w")
for i in range(5):
    name = input("enter name: ")
    f.write(name)
f.close()
```

output;

```
enter name: LOKI
" " : THOR
" " : ASIA
```



close() is must with out that it not write or modify the txt in files.

flush():- [<file handle>.flush]

if you want to force python to write the contents of buffer onto storage, you can use flush().

with open():- statement:-

-> with open('<fname>.txt','r') as <file_object>:

* WORKING WITH BINARY FILES:-

-> serialization (pickling);

-> unpickling;

* import pickle

dump(), load()

File = open("stu.dat","wb+")

File.close()

* writing onto a Binary file:- [pickling]

pickle.dump(<object-to-be-written>,<file handle ->)

eg:- pickle.dump(list1, fp)

list1 -> what

fp -> any data type

example:-

```
import pickle
stu = { }
stufile = open('stu.dat', 'wb')
ans = 'y'
while ans == 'y':
    rno = int(input("enter roll:"))
    name = input("enter name:")
    mark = float(input("enter mark:"))
    stu['rno'] = rno
    stu['name'] = name
    stu['marks'] = marks
    pickle.dump(stu, stufile)
    ans = input("want to enter more rec? (y/n):")
stufile.close()
```

* Reading from binary file [unpickling]:-

```
<object> = pickle.load(<file name>)
```

eof, try & except :-

or

with open :-

* Import pickle

```
FP = open("namu.dat", "rb")
stu = { }
Try:
    while True:
        stu = FP.load(FP)
        print(stu)
except EOFError:
    FP.close()
```

* by with open():-

```
with open("namu.dat", "rb") as FP:
    stu={}
    while True:
        stu=pickle.load(FP)
        print(stu)
```

* tell() & tell()

<File object>.tell()
└→ return the current position of the cursor.

<file_seek>.seek(n,n)
└→ no. of movement
└→ at position

mode,

└→ default 0

fp.seek(30) ⟹ move first 30 bytes

fp.seek(30,1) ⟹ move 30 bytes from current position

fp.seek(-30,2) ⇒ move backword 30 bytes from the end

fp.seek(-30,1) ⇒ move backword from current position

Working with CSV files;

21
16
7

Fp = open("nam1.csv", "w", newline = " ")

↳ additional argument for csv to avoid EOL Error.

* writing into a csv file:-

csv.writer() ——→ writer_object to be declared

<writer_object>.writerow() ——→ write one row onto w_ob

<writer_object>.writerows() ——→ writes multiple rows of data onto w_ob

eg:- import csv
fh = open("student.csv", "w")
w_o = csv.writer(fh)
w_o.writerow(['IAM', 'A', 'Boy'])

PROGRAM 1:-

```
import csv
fp = open("nam.csv", "w")
f_o = csv.writer(fp)
f_o.writerow(['name', 'Rollno', 'mark'])
ans = 'y', l = []
while (ans == 'y'):
    n_a = input("enter the name")
    R_N = int(input("enter the Rollno:"))
    M_RK = int(input("enter the mark:"))
    l = [n_a, R_N, M_RK]
    f_o.writerow(l)
    ans = input("enter y/N to continue:")
fp.close()
```

like writerows :-

line 1 line 2 line 3 ....

L = [['IAm'], ['Be'], ['The'], ['Best']]

1

store each line

L.append([name, roll, mark])

1

Inside while loop.

E0.writerows(L)

FP.close()

} changes from above programs for writerows()

* Reading a csv file :-

csv.reader() -> file_handle as argument

* used to read the csv file, we can get each line by using loop.

PROGRAM 1 :-

```
import csv
with csv("employee.csv", "r") as fh:
    ereader = csv.reader(fh)
    for i in ereader:
        print(i)
```

output :-

['Iam']

class and object

common — different objects with same class.

Syntax:

```
* class <name>:
    drink = " "
    def party(self):
        print("Let's party")
    def beach(self):
        print("Enjoying the beach")
```

```
ramesh = goa()
suresh = goa()

ramesh.party()
suresh.beach()
```

also,

```
* class goa:
    name = " "
    drink = " "

ramesh = goa()
ramesh.name = "Ramesh"
print(ramesh.name)
```

output:

Ramesh.

```
*
class goa:
    name = "ik"

print(goa().name)
```

output:

>> ik

*

* class laptop :

    price = 0

    processor = " "

    Ram = " "

HP = laptop()

Dell = laptop()

Lenovo = laptop()

HP. price = 60000

~~class~~ HP. processor = "i5"

HP. ~~new~~ Ram = "8Gb"

* class kodiaknal:

    def name = " "

    car = " "

    def boat(self):

        print("enjo
        boating")

    def food(self):

        print("eating

constructor and self keyword:

class laptop:

    def __init__(self):
        print("demo")

    def display(self):
        print("Display")

hp = laptop()

constructor = [is a unique function that gets called automatically when an object is created of a class.]

output

>> demo } but we not called any function from the object, only established the object of class

- construction also allocate memory for the class which is created.

* using constructor, dying a variable :-

```
class laptop:
    dy __init__(self):
        self.price = 0
        self.ram = ""

    dy .display(self):
        print("display")
```

```
hp = laptop()
hp.price = 50000
hp.ram = "16gb"
print(hp.price)
print(hp.ram)
```

output:

50,000

16gb

```
dy display(self):
    print("ram:", self.ram)
```

```
#main,
hp.display()
```

output:

16gb

self = denoting current object is called self or refering object into class.

class demo: } -> empty class
pass

class and object example:-

passing parameter to object:

class fruit :
    def __init__(self):
        self.color = "black"

apple = fruit()
print(apple.color) } #1

#1

#2

def __init__(self, color):
    self.color = color

(or)

def __init__(self, cor):
    self.color = cor

#2

also,

apple = fruit("red")
print(apple.colour) } #2

example:-

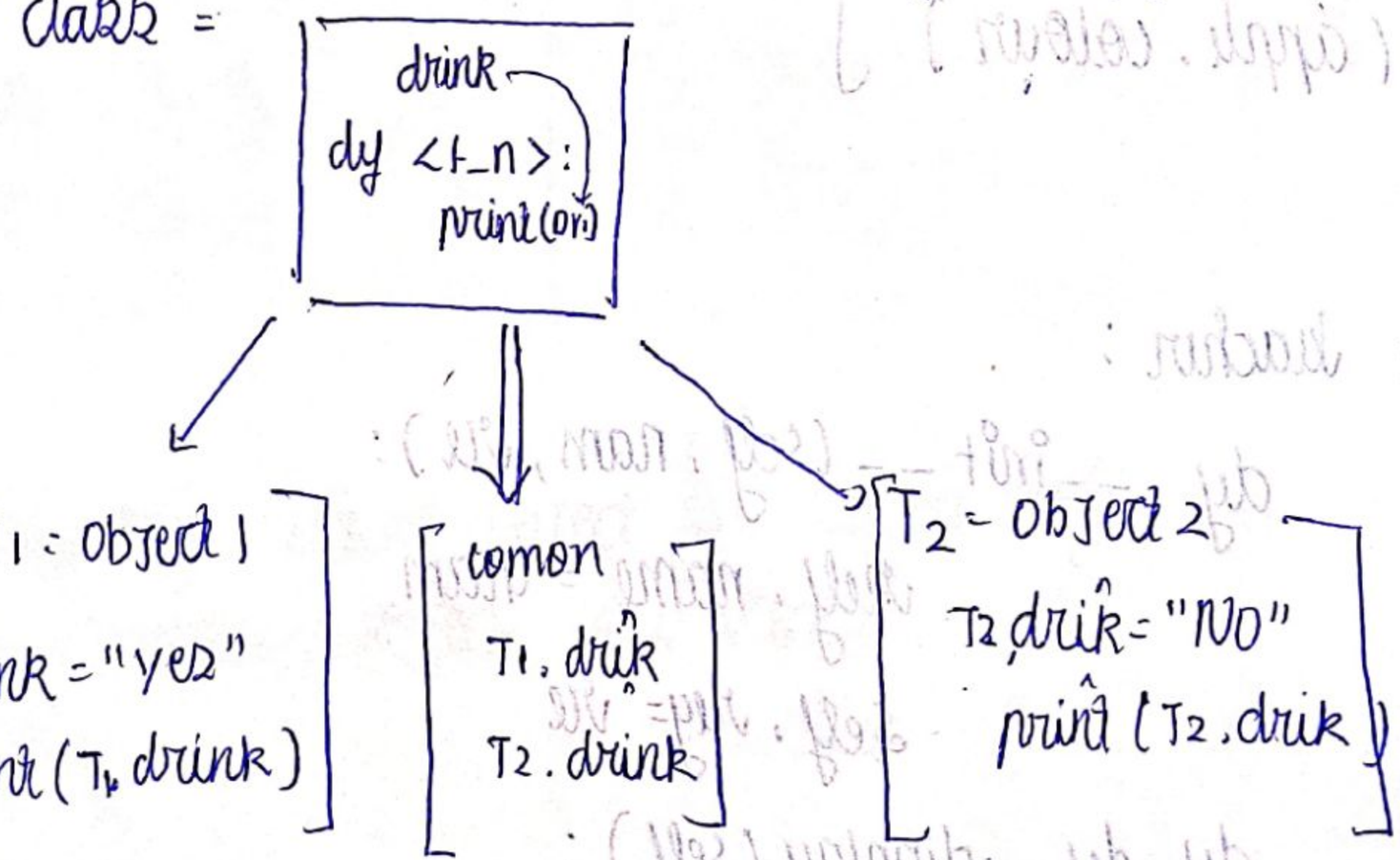
class teacher:
    def __init__(self, nam, re):
        self.name = nam
        self.reg = re
    def display(self):
        print("name:", self.name)
        print("reg:", self.reg)

TE = teacher("LOKI", "10203041")
TE.display()

example : 2 :-

```
clazz calculator:
    dy __init__(sely, a, b):
        sely.num1 = a
        sely.num2 = b
    dy add(sely):
        print("sum:", sely.num1 + sely.num2)
    dy multi(sely):
        print("mult:", sely.num1 * sely.num2)
T2 = calculator(2, 3)
T2.add()
T2.multi()
```

*flows of clazz and objects :-



constructor flow : & self flow :-

```
class oops:
    df __init__(self):          → tcr
tcr ← self.name = ""
        self.age = 0

    df display(self):           → tcr
        print(self.name)
        print(self.age)
                  ↓
                 tcr.

tcr = oops2( )
tcr.display( )

self = tcr
  ↓
referring_object
```

* instance variable :-

```
class phone:
    def __init__(self, name, Roll):
        self.name = name1
        self.Roll = Roll1
    def display(self):
        print("name:", self.name)
        print("Roll no:", self.Roll)

ob = phone("samsung", 2)
ob2 = phone("VIVO", 3)
```

{ instance variable, because, it save values according to object.

{ not common values }

[Note:- variables in constructor are instance variables.

* class variables:-

common variables are used for all objects is called class variables.

* class scope:

```
name = ""
ROll = 0
def display(self):
    print(self.name)
    print(self.ROll)

ob = scope()
ob.display()
```

* different types of class method:-

```
class laptop():
    chargertype="C TYPE"
    def __init__(self):
        self.brand=""
        self.price=34

    def setprice(self,price):
        self.price=price

    def getprice(self):
        print(self.price)
```

#1 instance method.
changing variable value by (self.)

```
hp=laptop()
hp.setprice(20000)
hp.getprice()
```

output:-
>>>20000

#2 .class method.

```
def setchargertype(self):
    chargertype="B TYPE"

def changechargertype(cls):
    cls.chargertype="B TYPE"
    print("charger.type changed to B")
```

value
class

laptop().changechargertype()
laptop

also, if we use @classmethod above function definition we commond just laptop().changechargertype(),

* static method:

```
def info():
    print("this is dayton class")
```

A. function without any class variable and instance variable

hp.info()

↳ raise error.

[if we use '@staticmethod' above the specific function.]

then,

hp.info()

↳ don't raise error,

print the following statement in the function

note:-

```
class A1:
    def display(self):
        print("name")
ob = A1()
A1.display(ob)
```

(work finely)

↳ object

↳ class name

: Inheritance :-

accessing a one class by the object of another one class;

normally :-

```
class dad():
    dy phone(self):
        print("Dad's phone")

class son():
    dy laptop(self):
        print("son's laptop")

ram = son()
ram.phone()
```

→ raise error as `no attribute of phone in son class'

;o,

linking son → dad.

we dad as :-

```
class son(dad):
    another class dad by son class object;
```

output → [after this]

→)

"Dad's phone".

* multiple Inheritance:-

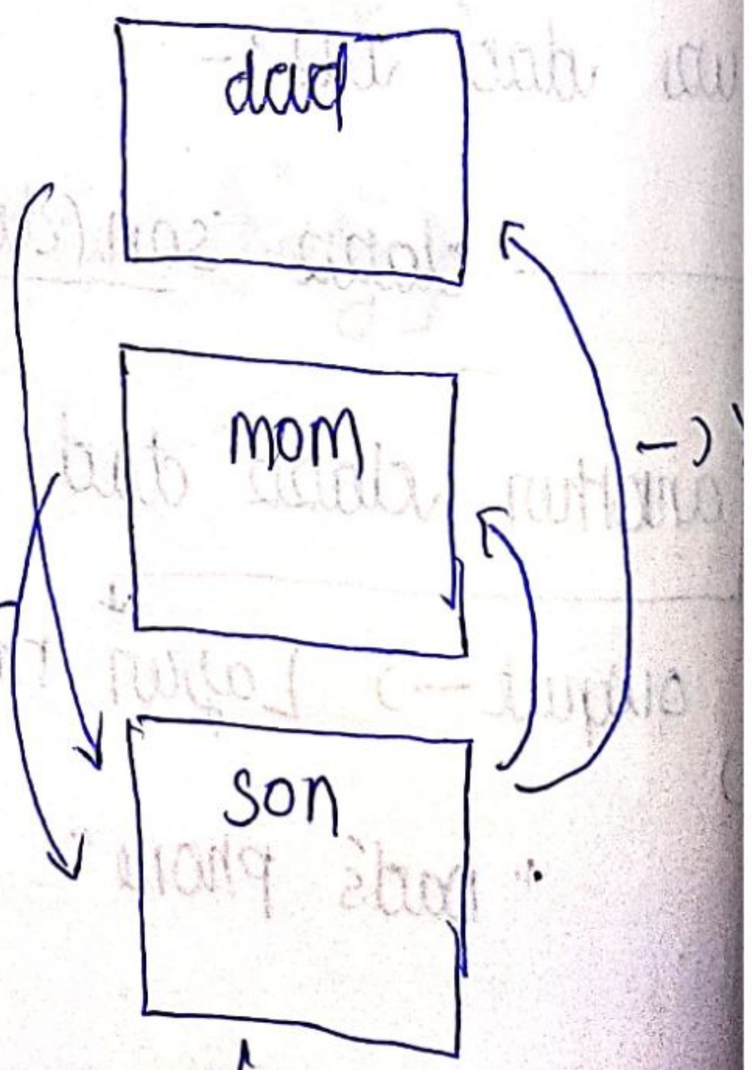
one class can acces two classes called multiple Inheritance.

example 1 # Inheritance [multiple]

```
class dad():
    dy phone(sely):
        print(" Dads phone ")

class mom():
    dy sweet(sely):
        print(" moms sweet ")

class son(dad, mom):
    dy laptop(sely):
        print(" son's laptop ")

ram = son()
ram.phone()
ram.sweet()
```

output :-

Dad's Phone
mom's sweet



1.1

example 2: # multilevel inheritance

```
class grandpa():
    def phone(self):
        print("Our grandpha phone")
grandpa
class dad(grandpa):
    def money(self):
        print("dads money")
```

dad

```
class son(dad):
    def laptop(self):
        print("son's laptop")
```

```
ram = son()
ram.money()
ram.phone()

dad = dad()
dad.phone()
```

output:-

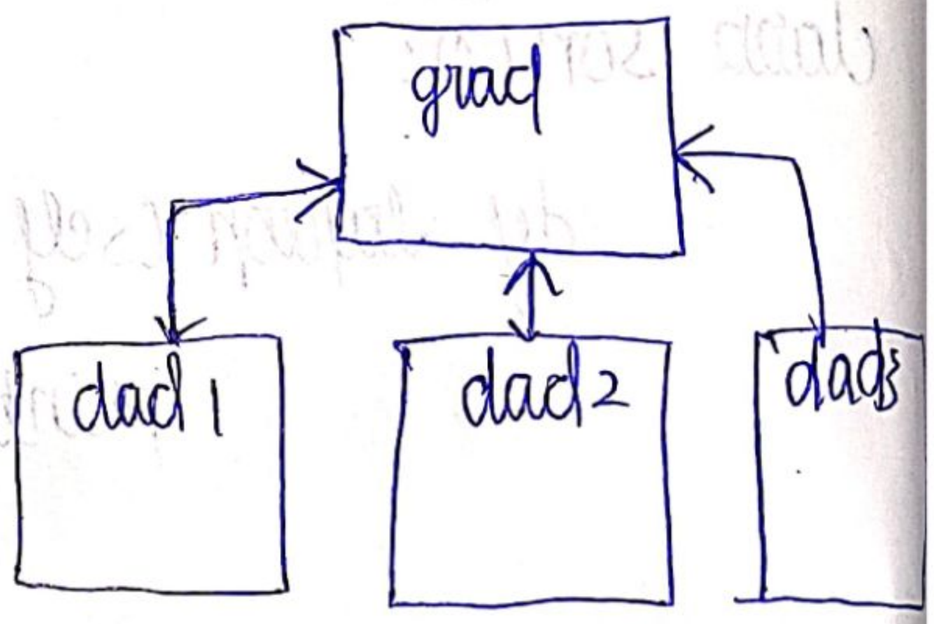
```
pad's money
grandpha phone
grandpha phone
```


son, can access grand

if dad and grand related & son and dad related.

example 3: [#

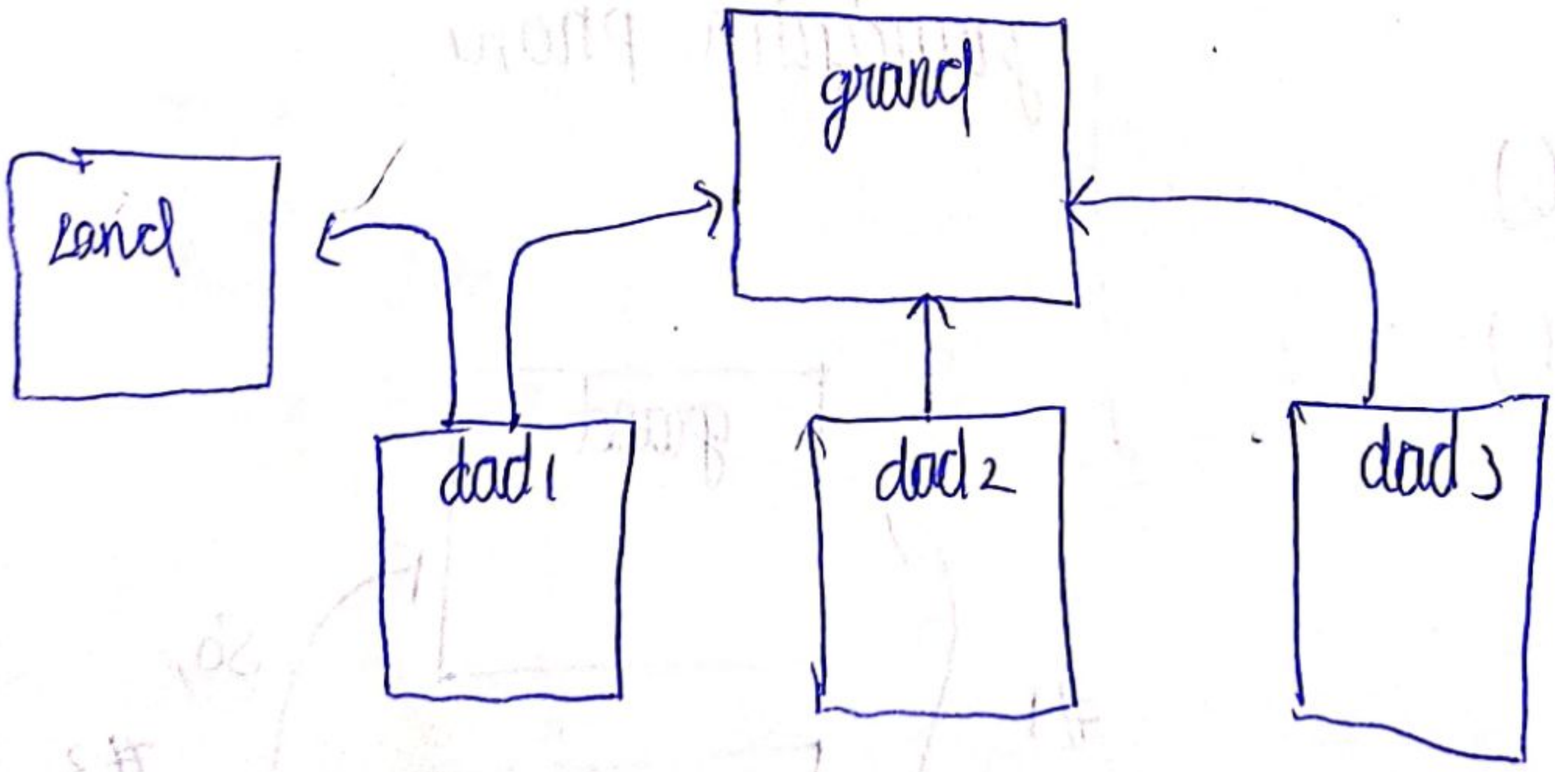
```
class grandpa():
    dy phone(sey):
        print("grandpa's phone")
class dad1(grandpa):
    dy money(sey):
        print("dad's money")
class dad2(T: grandpa
    dy pass
class dad3(grandpa):
    pass.
```



The Thru dad can access the grand,

example 4:-

if. flow is [# hybrid Inheritence]



super keyword ():- [use to use a function of main class @ in sub class which have same name]

```
class a:
    def __init__(self):
        print("main class:")

class b(a):
    def __init__(self): ——→ super().__init__()
        print("sub class:")
```

now,

```
* ob = b()
```

output:

sub class:

* if no constructor in b

So,

```
ob = b()
```

output

main class:

↳ main class constructor is called because, b inherit a

* if two constructor want to display

↳ add [super().__init__()]

So,

```
ob = b()
```

output:-

main class

sub class

↓

in body of b class constructor.

* Polymorphism:- [having many forms. In programming, polymorphism means the same function name (but different signature) being used for different types. The key difference is the data types and number of arguments used in function].

#1 :-

```
class Animal:
    def sound(self):
        print("Animal sound:")

class dog(Animal):
    def sound(self):
        print("Dog Barks:")
```

* ~~class~~

```
ob = dog()
ob.sound()
```

~~Out~~ output:-

Dog Barks:

* if no "def sound:" function in dog then,

```
ob = dog()
ob.sound()
```

output:-

Animal sound.

↳ main class

* So, By using "super().sound()" in sound() of dog class

So,

```
ob = dog()
ob.sound()
```

output:-

Animal sound
Dog barks.

example: #2:-

create a base class called "person" with a constructor that takes a name as a parameter. create a derived class "student" that inherits from person and has a constructor that takes a parameter called "grade". write a method in "student" to display the name and grade of the student. use super keyword to achieve this.

sol:-

```
class person():
    def __init__(self, name):
        self.name = name

class student(person):
    def __init__(self, name, grade):
        super().__init__(name)
        self.grade = grade

    def display(self):
        print(self.name, self.grade)

s1 = student("John", "A")
s1.display()
```

output:- John A

encapsulation :-

↓

[public, private, protected]

class do:

def __init__(self):

self.name = "google" — public [use by any where]

self._name1 = "facebok" — > protected [can't change]

self.__name2 = "Bank" — > private [can't change, & can't print]

ob = do()

print(ob.name) — > google

print(ob._name1) — > Facebook

print(ob.__name2) — > Error '__name2 not defined'

[ob.name)

[, [print only by class function

* modulus

add.py

```
dy addor(a,b):
    return a+b
```

calc.py

```
dy addor(a,b):
    return a+b
dy sub(a,b):
    return a-b
```

1° main.py (in same folder)

```
import add
dy print(add.addor(5,6))
```

or

2° main.py (same folder)

```
From add import addor
print(addor(5,6))
```

3° main.py (same folder)

```
from calc import addor, sub
print(addor(5,6))
print(sub(6,5))
```

or 4° main.py (same folder)

```
from calc import *
print(addor(5,6))
print(sub(6,5))
```

* all function
all variable
all classes