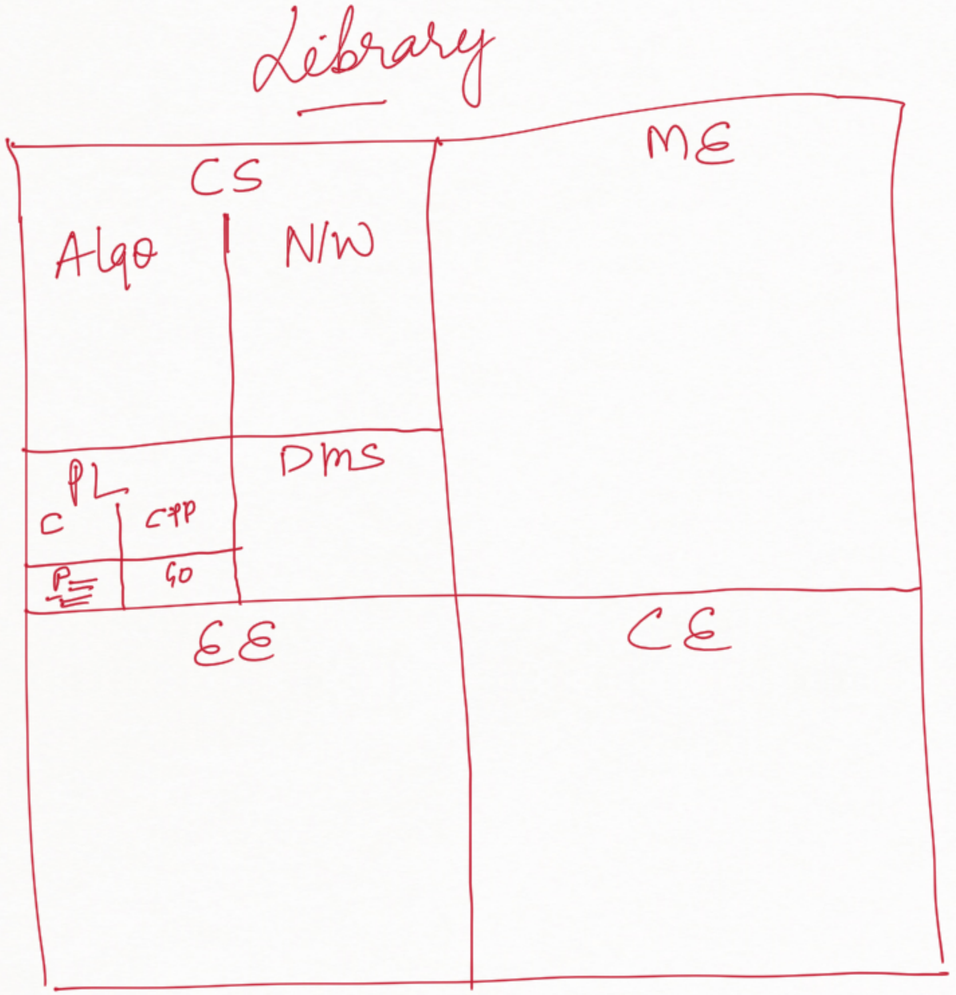
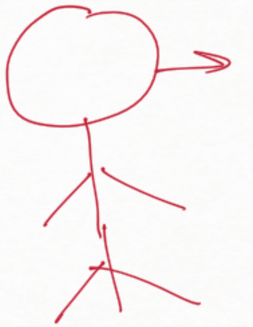




Book store

→ Extra cost

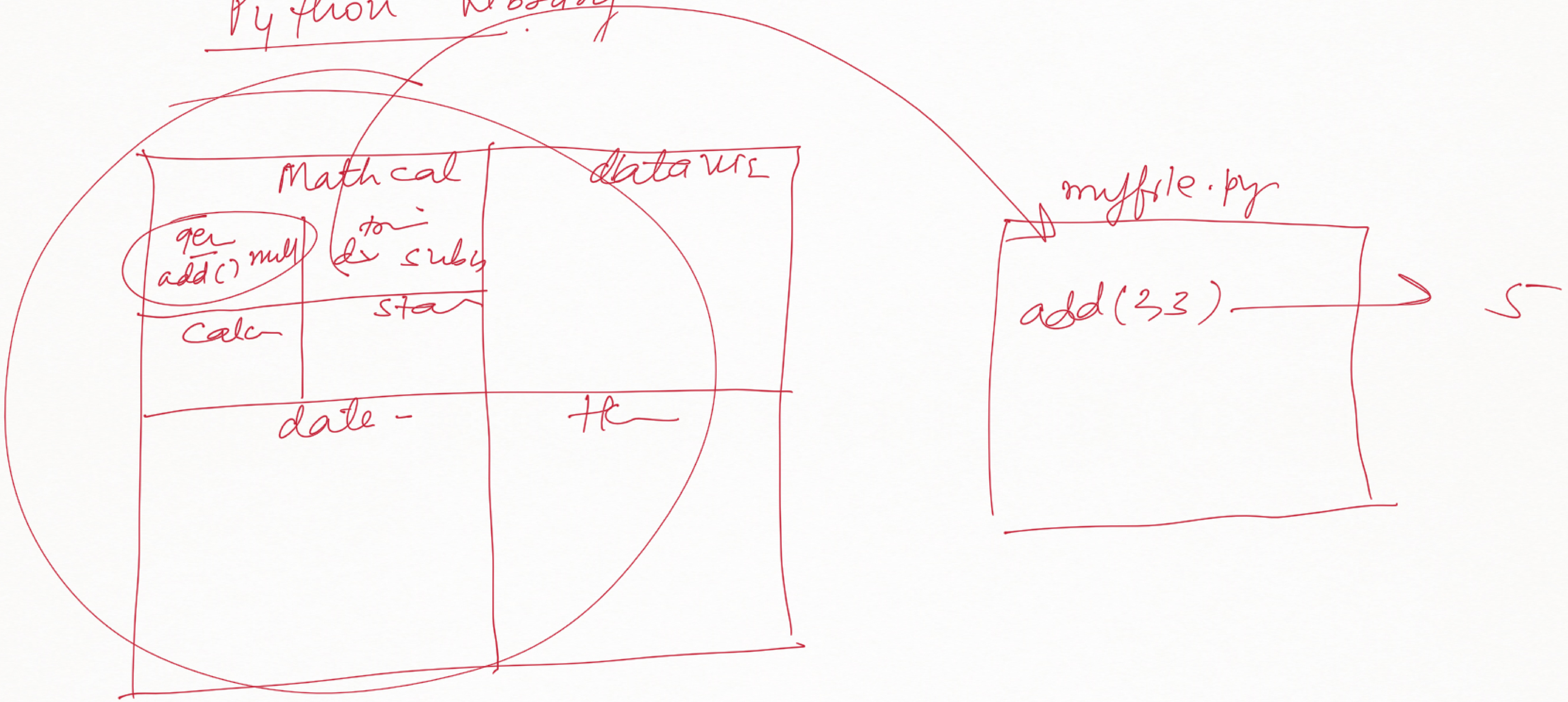
→ Extra term

dq libs

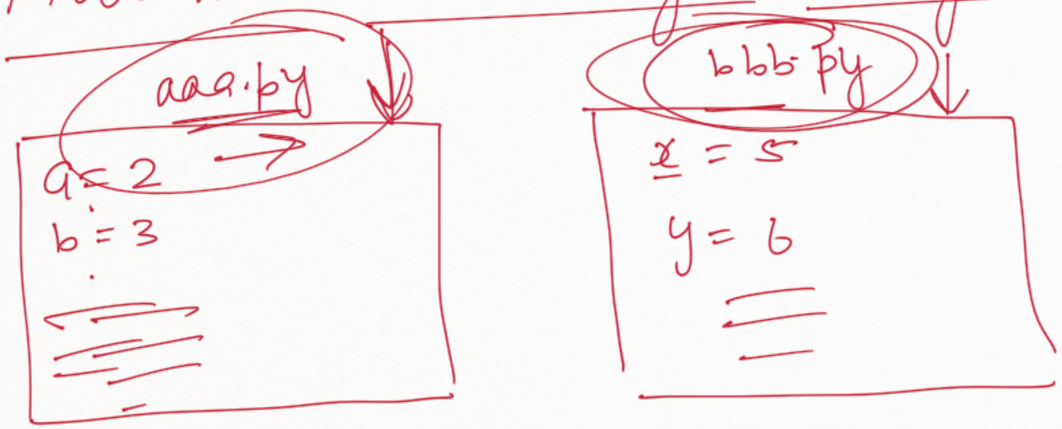
→ No extra cost

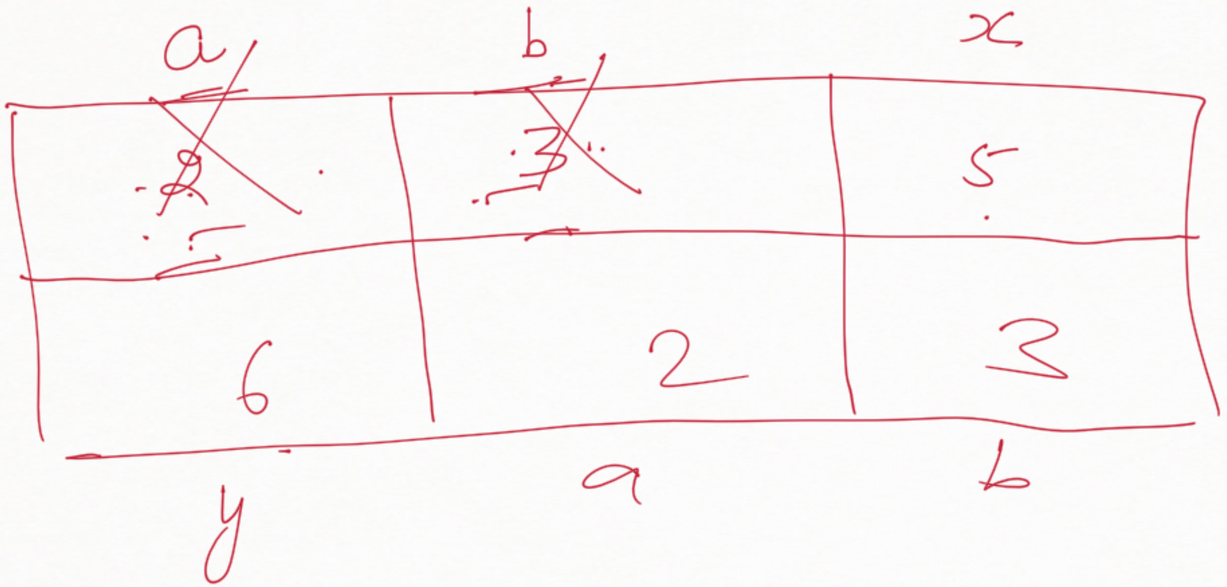
→ No extra ti

Python Library



Automatic memory management & Garbage collector :



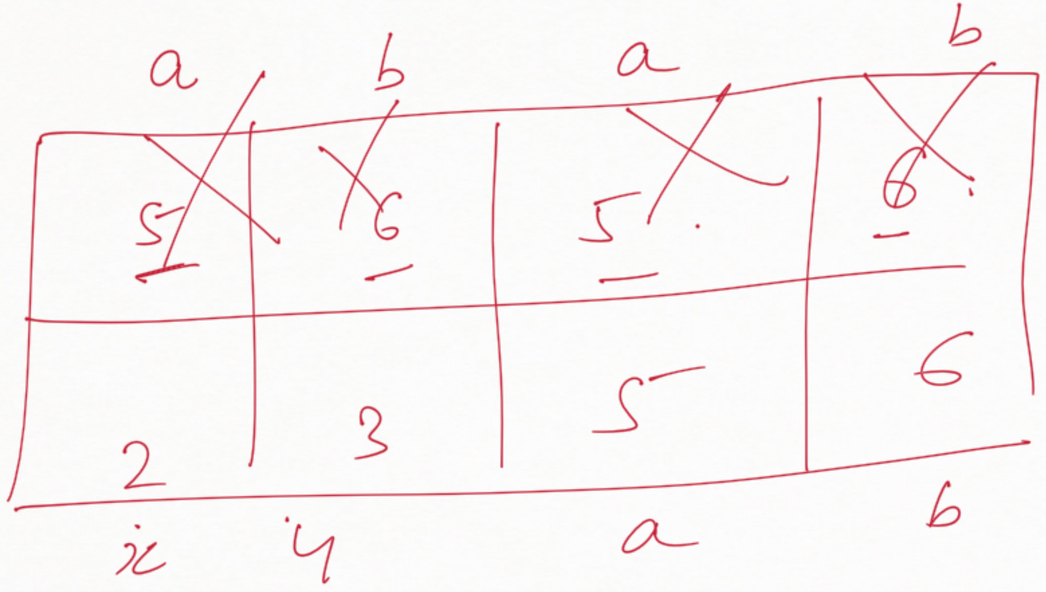


aaa.py

```
a = 5
b = 6
```

bbb.py

```
x = 2
y = 3
```

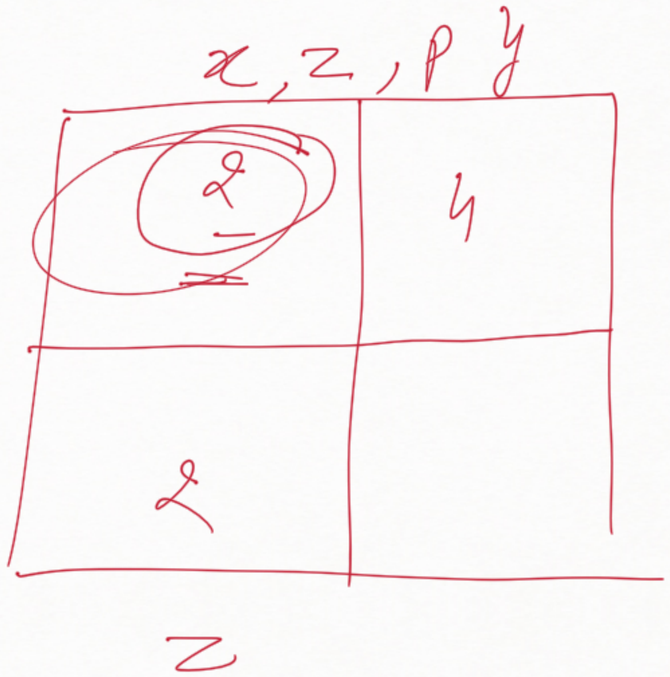


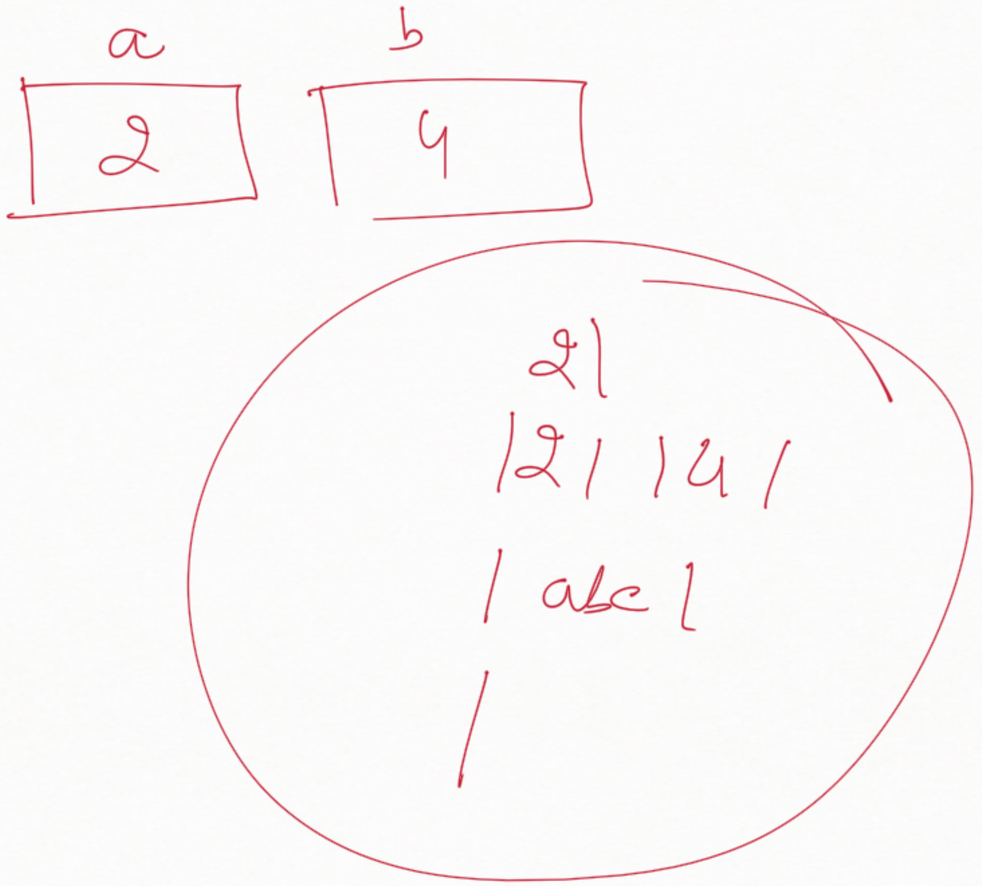
$x = 2$

$y = 4$

$z = 2$

$p = 2$





→ a = 2

→ b = 4

→ print (a)

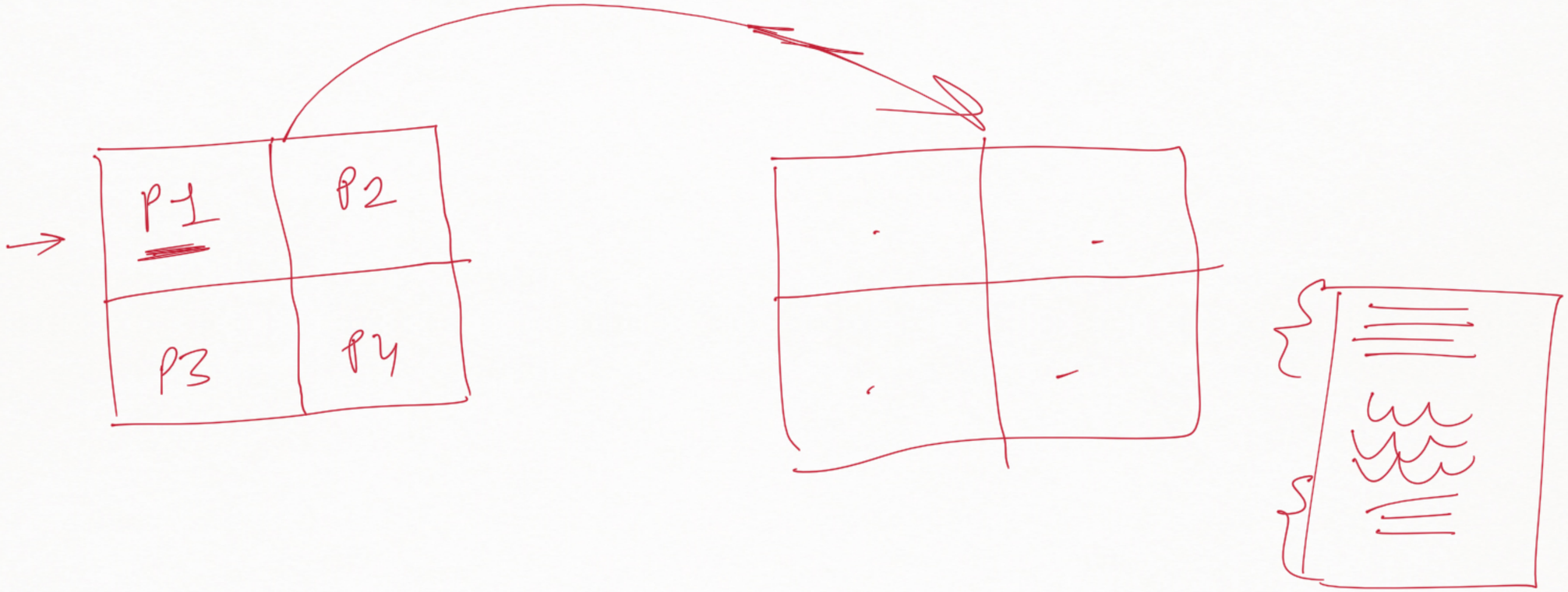
→ print (a, b)

→ print ('abc')

→

sep = '_' end = '\n'

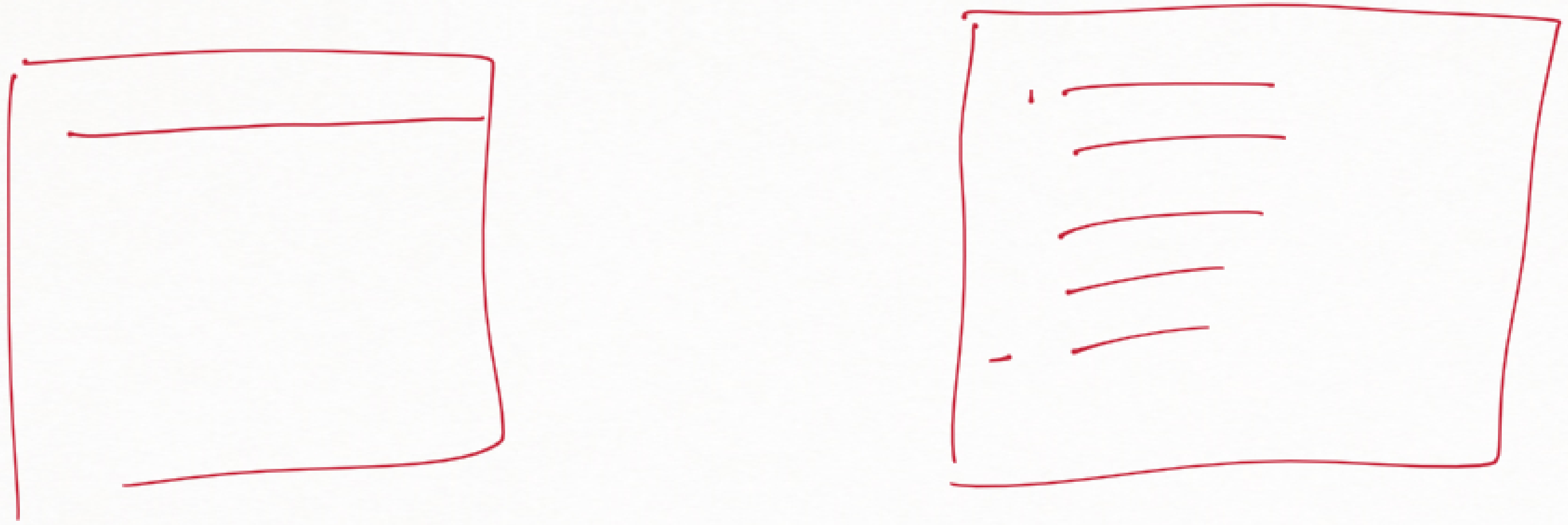
Comments in Python :-



Comment

Single line | multiline comment

#



''' ≡ '''

""" = """

Data type in Python:

Number

int → -n to +n

float → -n.n to +n.n

complex → 2+9j

j → $\sqrt{-1}$.

2 → real

9 → imag

String :

'2'

"2"

'''2'''

"""2"""

collection of characters.

'abc' → string

'a' → string

x = 'abc'

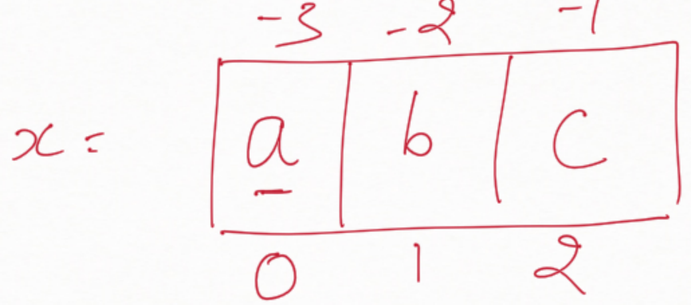
x ↵

x[0] or x[-3]

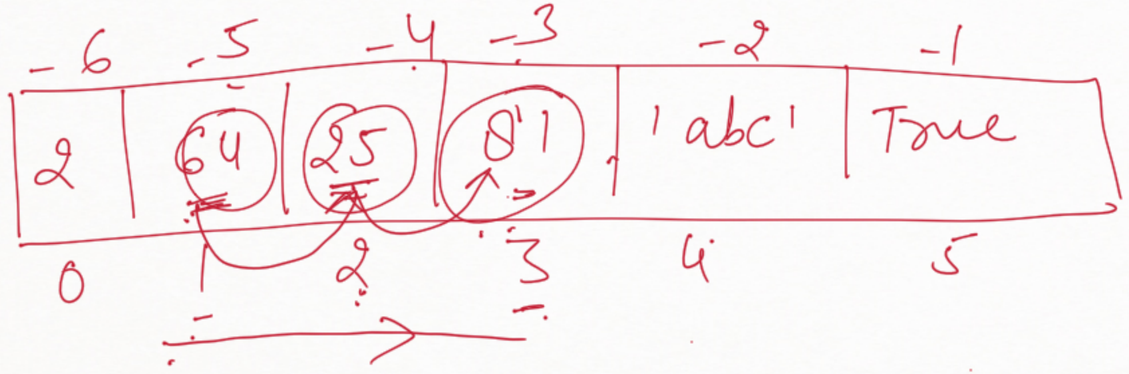
↪ 'a'

x[-2] or x[1] → 'b'

x[-1] or x[2] → 'c'



a = [2, 64, 25, 81, 'abc', True]



[64, 25, 81]

-4 + 5 = 1

2 - 1 = 1

3 - 2 = 1

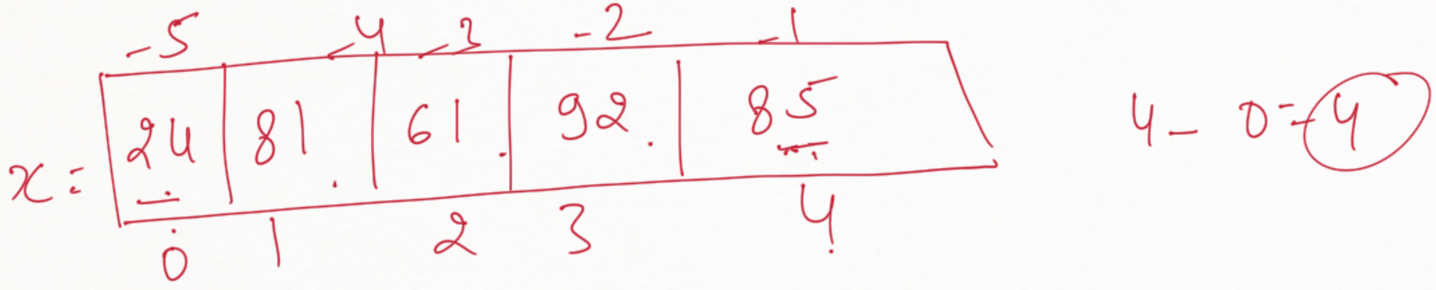
ln[start : stop : step]

a[-5 : -2 : 1]

a[1 : 4 : 1]

a[-5 : 4 : 1]

a[1 : -2 : 1]

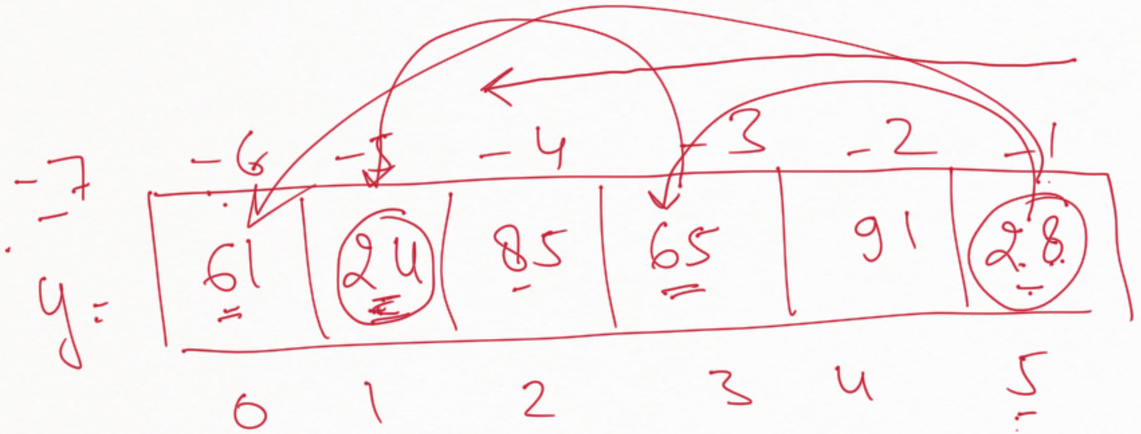


4 - 0 = 4

[24, 85]

x[0:5:4]

x[-5:5:4]



3 - 5 → -2

1 - 3 → -2

0 - 5 → -5

[28, 65, 24]

y[5 : -6 : -2]

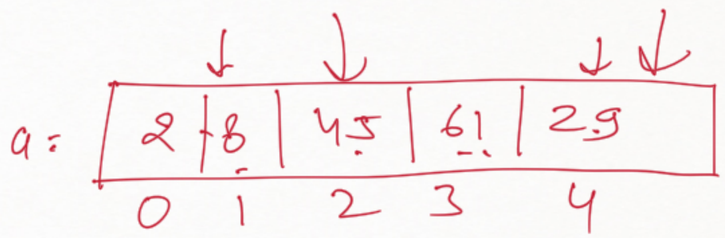
y[-1 : 0 : -2]

[28, 61]

y[5 : -7 : -5]

y[-1 : -7 : -5]

	Start	Stop	Step
M:N:L	M	N-1	L
M:N →	M	N-1	+1
M:	M	len(a) -1	+1
M::L	M	"	L
:N:L	0	N-1	L
:N:	0	N-1	+1
::L	0	len(a)-1	L
:	0	"	+1

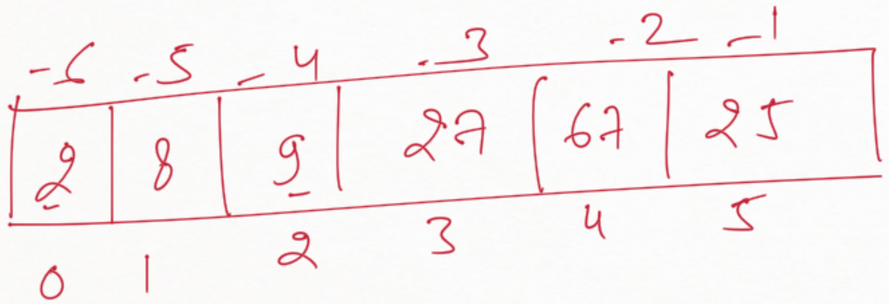


a[1:5:2] → [8, 61]

a[2:5] →

list_name [start : stop : step]

a:



[2, 8, 9)

a[0:3:1]

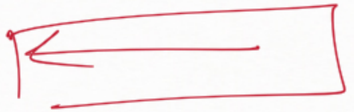
Slicing L → R

Start → 0

Stop → -1

Step → +1

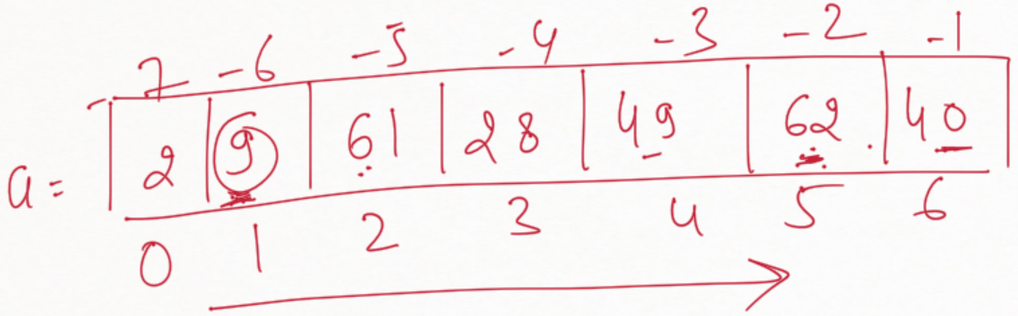
Slicing R to L



start → -1

stop → 0

step → +1



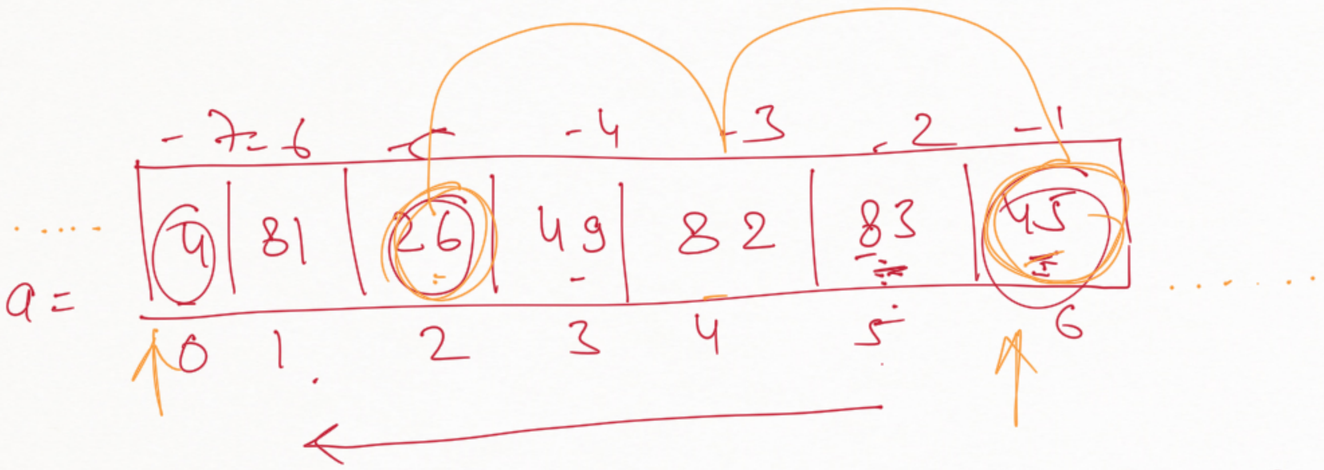
if step → + L to R

− R to L

a[1:6:3] → [9, 49]

a[1:6] → [9, 61, 28, 49, 62]

a[1:] → [9, 61, 28, 49, 62, 40]



a[:] → [4, 81, 26, 49, 82, 83, 45]

a[6: 2: -1] → [45, 83, 82, 49]

a[5: 1] → [ ]

a[5:] → [83, 45]

89

a[::-1] →

[45, 81, 82, 49, 26, 81, 4]

a[2: 89: 2] → [26, 82, 45]

a[:-24: -2] → [45, 82, 26, 4]

Adding elements

append() → add one element at last of the list

insert() → add one element at your desired location

extend() → add more than one element at last of the list

Removing elements

pop() → remove one element by using index from list

remove() → remove one element ——— value from list

del → remove one or more than one ———

clear() → remove all el—

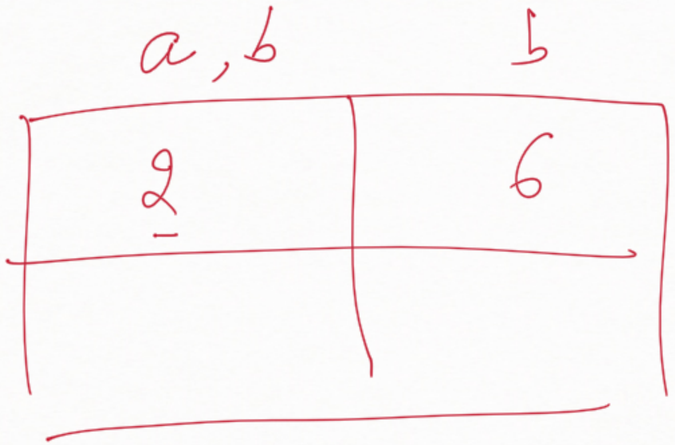
other methods

- index() → accessing index value
- count() → occu of any ele
- reverse() → reve the element
- sort() → sorting into ascend or descend order
- copy() →

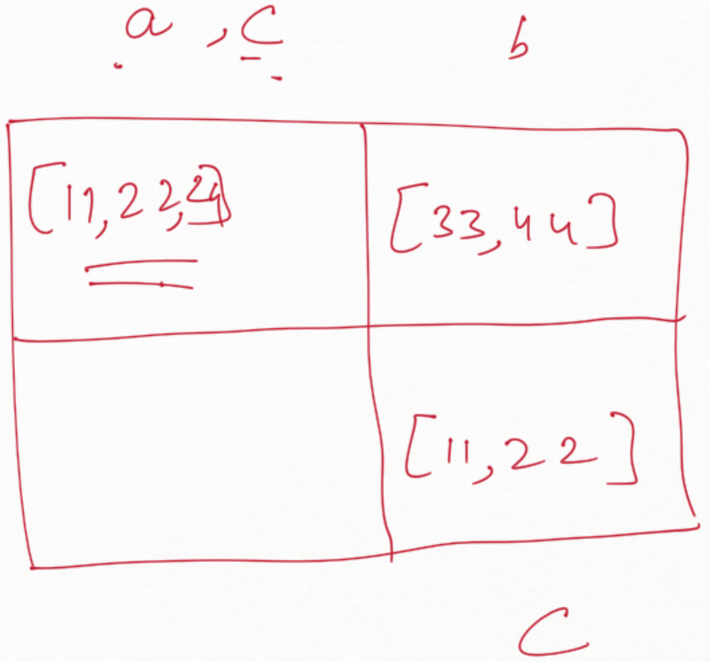
$a = 2$

$b = 6$

$c = 2$



```
a = [11, 22]

b = [33, 44]

c = a

a.append(24)

c = a.copy()
```



Tuple : collection of hetrogeneous type of data, indexed, ordered, immutable (unchangeable).

a[-1][1]

a = (2 , 2.5 , True , 3+9j , [2,5,6])

a[0] → 2

a[-5] → 2

a[-3] or a[2]

a[3].real → 3.0

a[-2].imag → 9.0

a[0:3:1]

-5	-4	-3	-2	-1
2	2.5	True	3+9j	[2,5,6]
0	1	2	3	4

-3	-2	-1
2	5	6
0	1	2

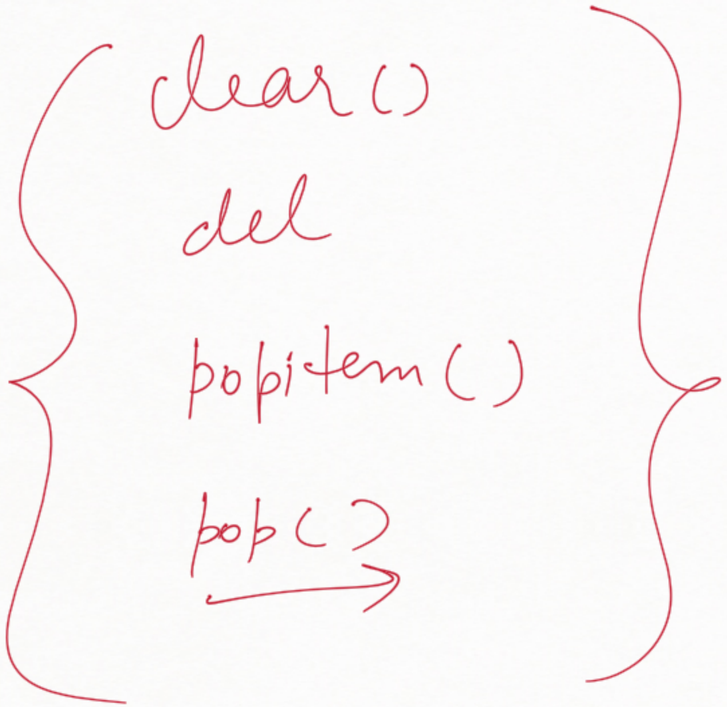
a[-1][::2] → [2,6]

Dictionary :- collection of hetrogeneous type of data, unindexed, ordered, mutable...

Key : value , Key unique ,... , { }

```
Student = { name : 'manoj',
            age : 12,
            sub : ['maths', 'chem', 'Python']
          }
```

Remove :



Nested dictionary :-

dictionary inside dictionary

Set :- collection of immutable type of data,

unique, mutable, { }, unordered, unindexed

a = { 8, 2.5, True, 2+3j, (2,5,6) }

a1 = {2, 5, 6, 2, 8, 1} error X

{2, 5, 6, 8, 1}

union

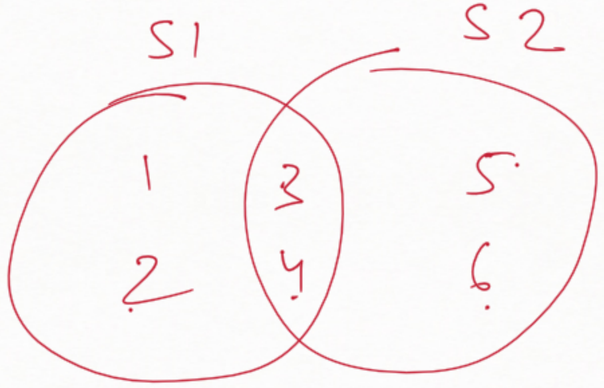
intersection

difference

symmetric

$S1 = \{1, 2, 3, 4\}$

$S2 = \{3, 4, 5, 6\}$



$S1 - S2 = \{1, 2\}$

$S2 - S1 = \{5, 6\}$

$S1 \cap S2 = \{3, 4\}$

$S1 \cup S2$

$\{1, 2, 3, 4, 3, 4, 5, 6\}$

$\{1, 2, 3, 4, 5, 6\}$

adding element in set :

add ⟶ 1 element

update ⟶ more than 1 el

delete

discard()

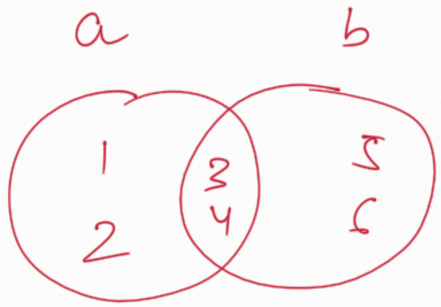
remove()

pop()

del

$a = \{1, 2, 3, 4\}$

$b = \{3, 4, 5, 6\}$



a.difference(b) = {1,2}

$a = \{1, 2, 3, 4\}$

$b = \{3, 4, 5, 6\}$

a.diff_update(b) = {1,2}

$a = \{1, 2\}$

$b = \{3, 4, 5, 6\}$

Boolean :- True , False

1 0

logical operator :- and, or, not, xor ----

i1	i2	and	or	xor
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

not $0 \rightarrow 1$

$1 \rightarrow 0$

and → logical operator

a = 20

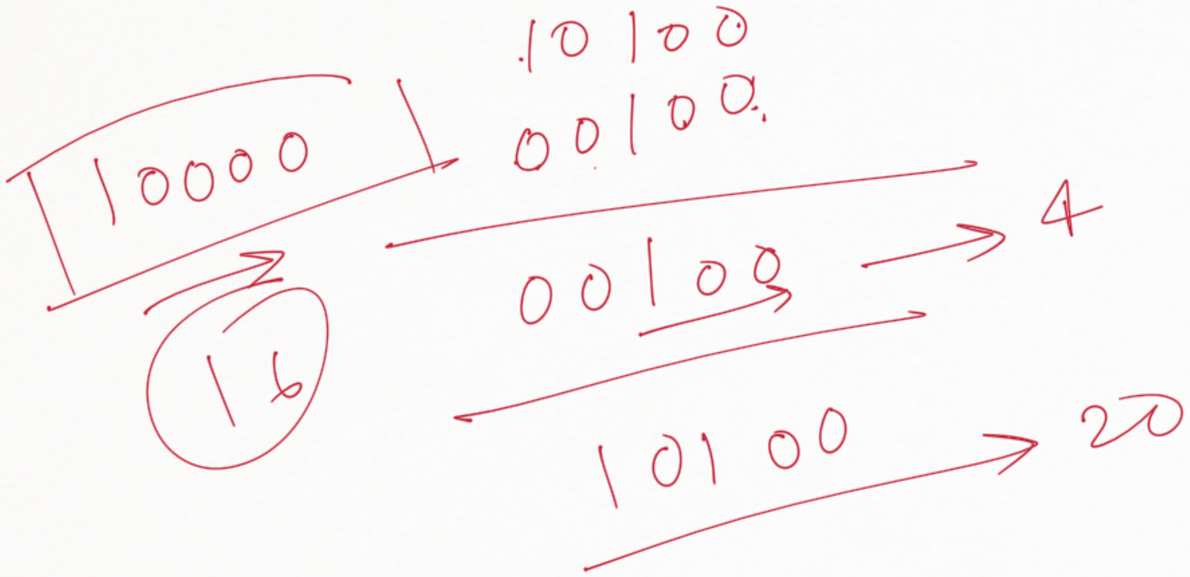
b = 4

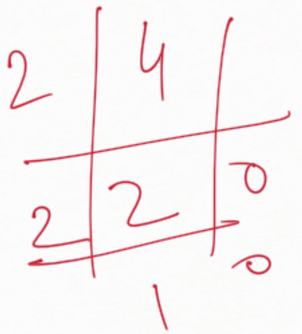
20 → 10100

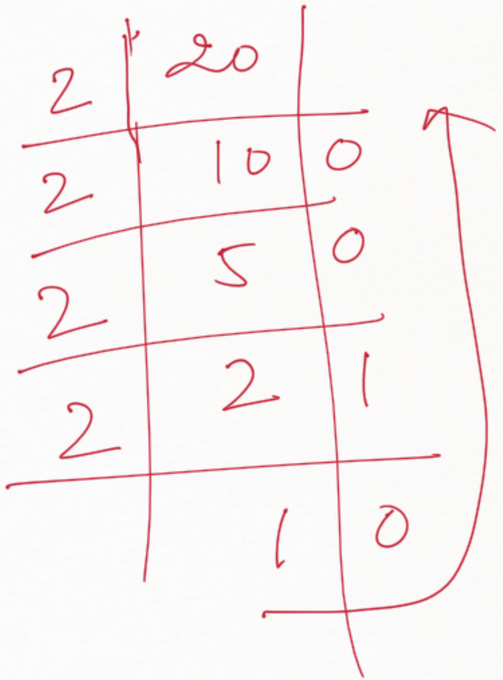
4 → 100

a and b → 4

10100







a = 2

b = 4

a+b → 6

input ( )

↓

string

---

str → num

→

process of converting one data type into another type of data -

str → int

int → float

typecasting

a = 'a'

a: '4' → ~~int~~ str

int(a)

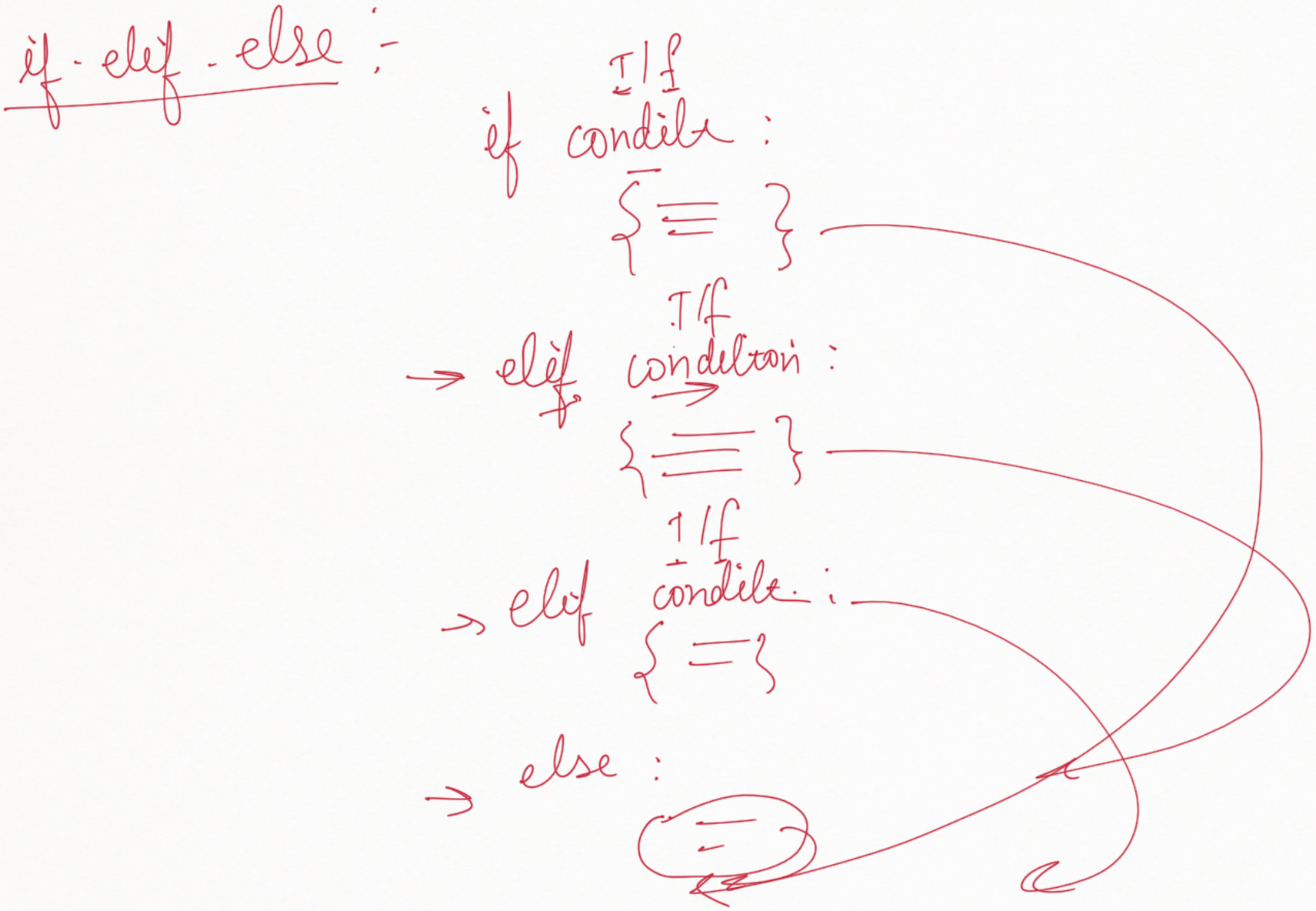
Control Statement :-

- if
- if - else
- if - elif - else
- nested - if

True / false

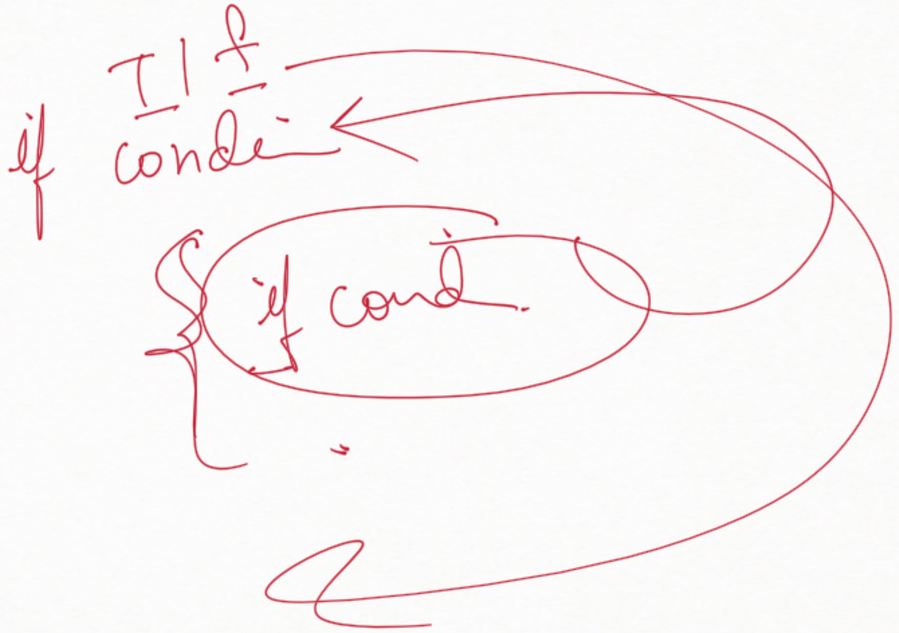
if condition :

if - elif - else :-



Nested if.

if inside if



a, b, c = 2, 5, 9

if (a >= b and a >= c):

    print('a is gr b and c')

2 >= 5 and 2 >= 9

f and f → f

Project - 1

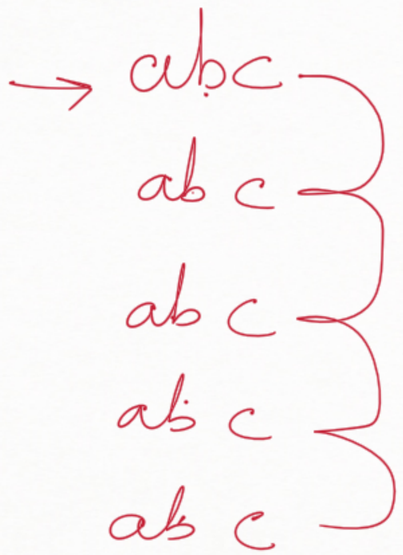
Quiz Game

Loop :- Repeatation of statement

while

for

While loop :-



initialization
condition
incr/decr

→ print('abc') → a = 1

→ While a < 6 :

print('abc')

a = a+1

a=1

1<6 T

a=2

2<6 T

a=3

3<6 T

a=4

4<6 T

a=5

5<6 T

a=6

6<6 f

abc
abc
abc
abc
abc

1

→ 1.
+1
2. +1
3. +1
4.
:
:
:
10.

a = 1

While a < 11:

{ print(a)

a = a + 1 or a += 1

/ 1
12
1

a = 1

1 < 11 T

2 < 11 T

3

→ 5-1
4-1
3-1
2-1
1

```
a=5
→ while a>0 :
    → print(a)
      a=a-1   or  a-=1
```

```
a=5
5>0 T
a=4
4>0 T
```

```
5
4
3
1
```

→ 2 + 2
4 + 2
6 + 2
8
10
12
14
16
18
→ 20

a = 2 + 2
a <= 20 T
a = 4
4 <= 20 T
a = 6
.
.
.

a = 2
while a <= 20 :
→ print(a)
a = a + 2
→

2
4
.

$2 \times 1 = 2$

$2 \times 2 = 4$

$2 \times 3 = 6$

.

.

.

.

$2 \times 10 = 20$

```
a = 1
while a<=10:
    print(ax2)
    a+=1
```

a = 1

K = 10T

1 x 2 = 2

a = 2

2 <= 10T

2 * 2 = 4

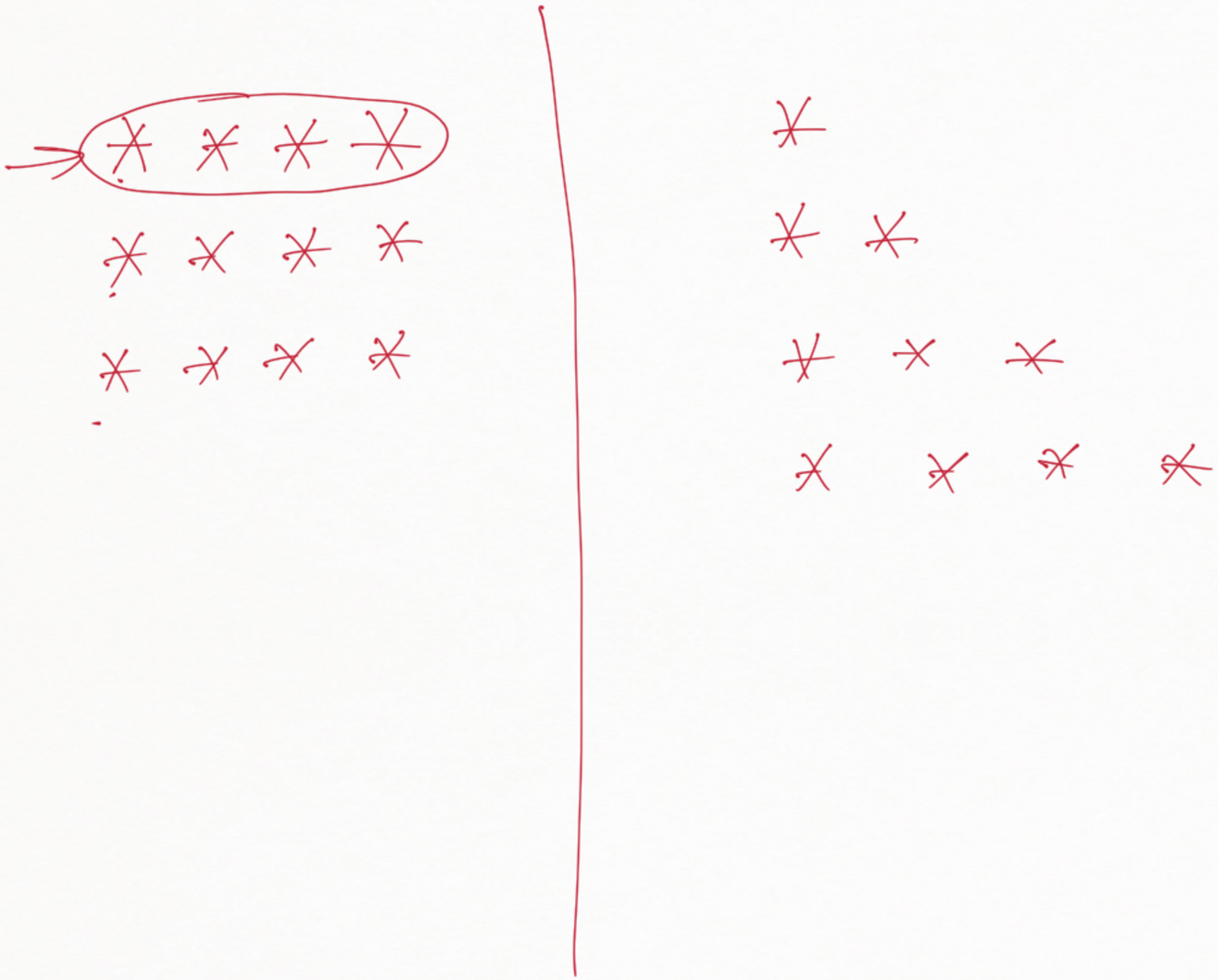
1
2
3
4
.
.
.
10

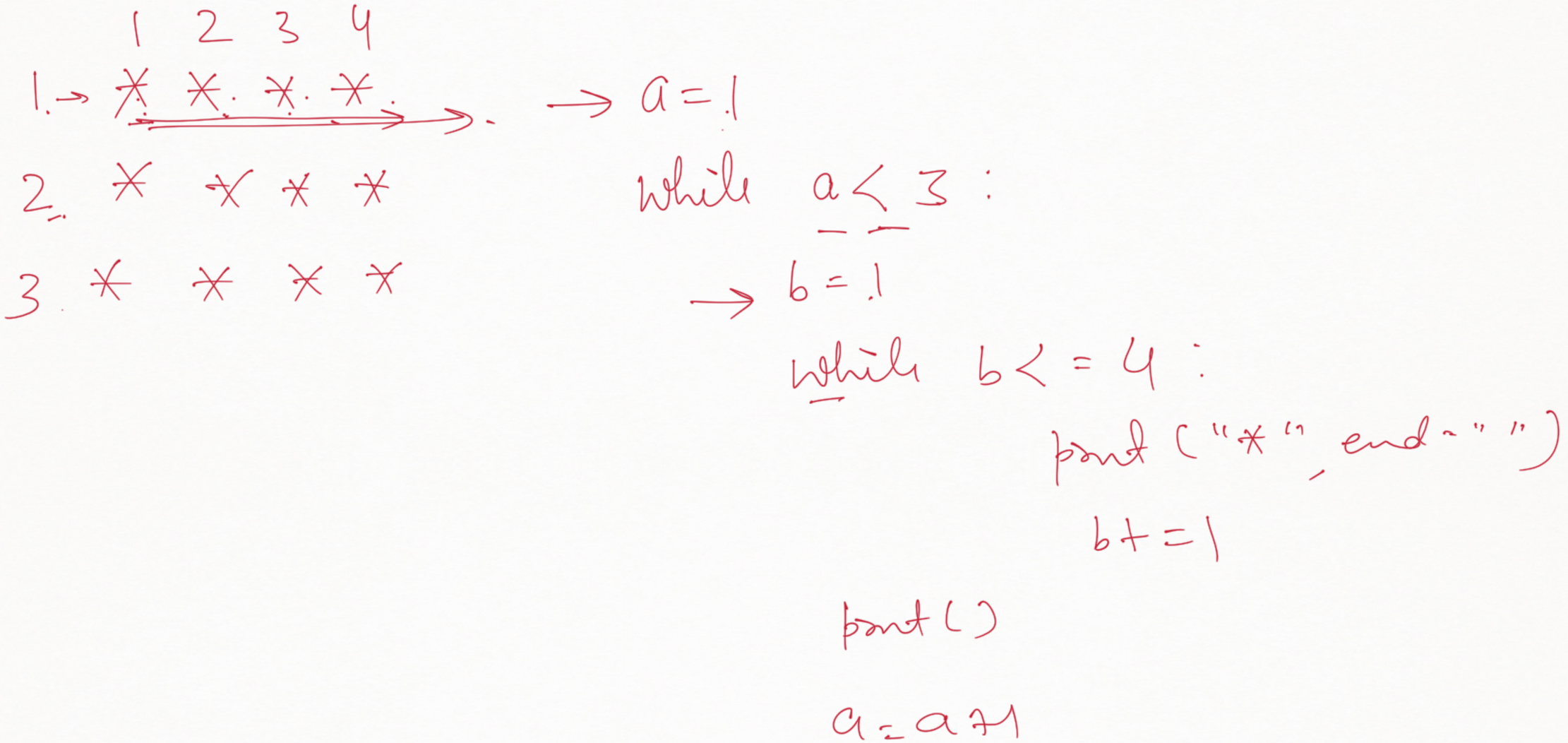
Nested while loop :

while loop inside while loop

.

```
while
    while
        whil  —
              —
              —
```




1 2 3 4
1.
2.
3
a = 1
while a < 3 :
b = 1
while b <= 4 :
print("*", end=" ")
b += 1
print()
a = a + 1

```
    1  2  3  4
1   *
2   *  *
3   *  *  *
4   *  *  *  *
```

```
row = 1
while row <= 4 :
    col = 1
    while row >= col :
        print ("*", end = " ")
        col += 1
    print()
    row += 1
```

row col

1 = 1

2 => 1 2

3 => 1 2 3

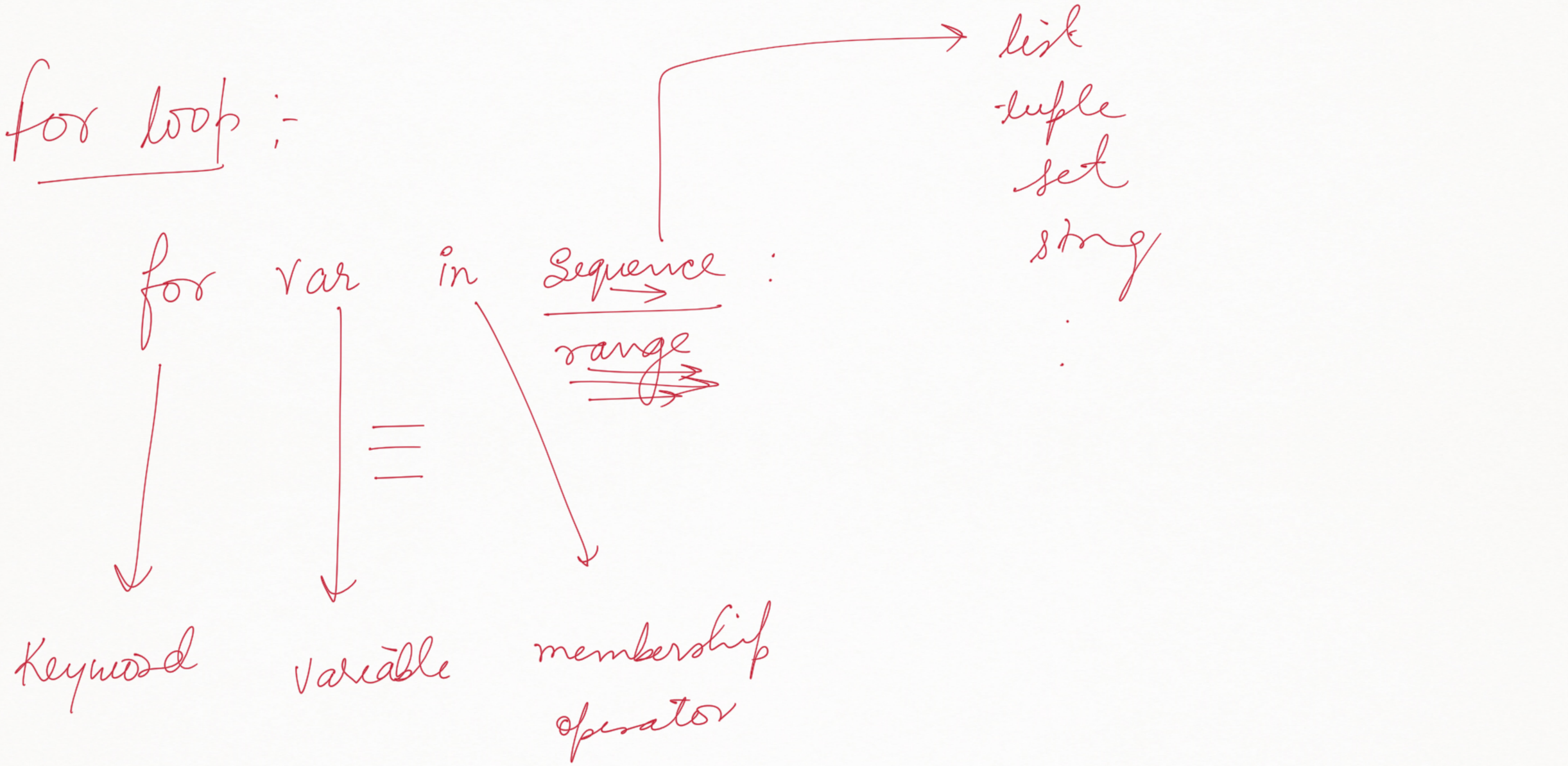
4 => 1 2 3 4

Try to do it by yourself

all the best

:)

for loop :-


for
var
in
Sequence
range
list
tuple
set
string
Keyword
variable
membership
operator

Membership operator :-

in	True
not in	false

x = [2, 5, 'abc', 29]

2 in x True

'abc' in x True

21 in x false

5 not in x false

29 not in x false

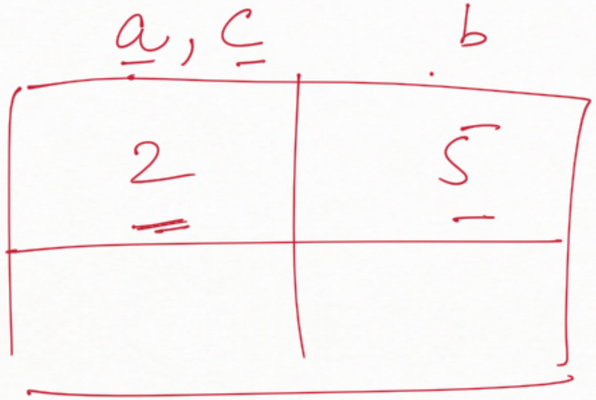
50 not in x True

Identity Operator : { is / is not } true / false

a = 2

b = 5

c = 2



a is b false

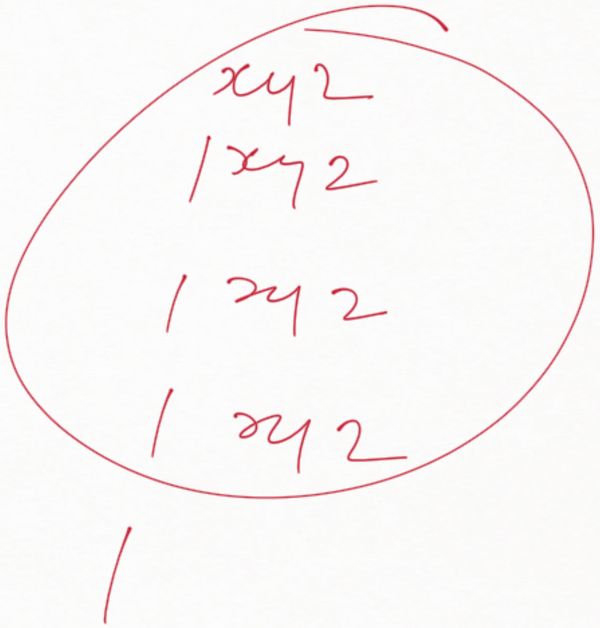
id(a) =/= id(b)

a is c True

a is not c false

a is not b True

```
for x in [2, 8, 5, 'abc']:
    {print('xyz')
```



2 in [2, 8, 5, 'abc'] True

8 in " True

5 in " True

'abc' in " True

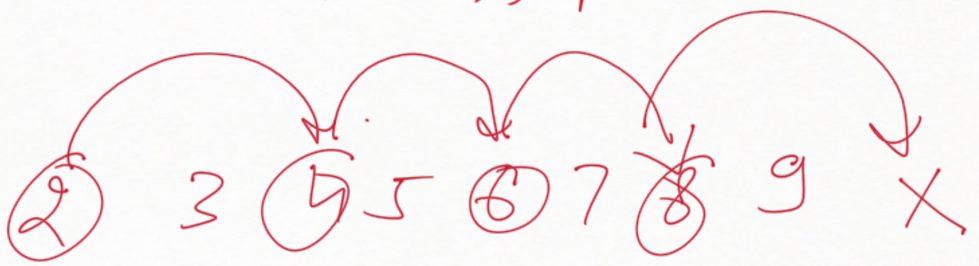
range(start, stop, step)

start: 0
stop: X
step: +1 -1

range(0, 5, 1)

0 1 2 3 4

range(2, 10, 2)



(2, 4, 6, 8)

range(10, 2, -1)

10 9 8 7 6 5 4 3

1 X 2

2

3 X 5

4

.

.

.

10

```
for i in range(1,11):
    print(i*2)
```

2

4

6

8

.

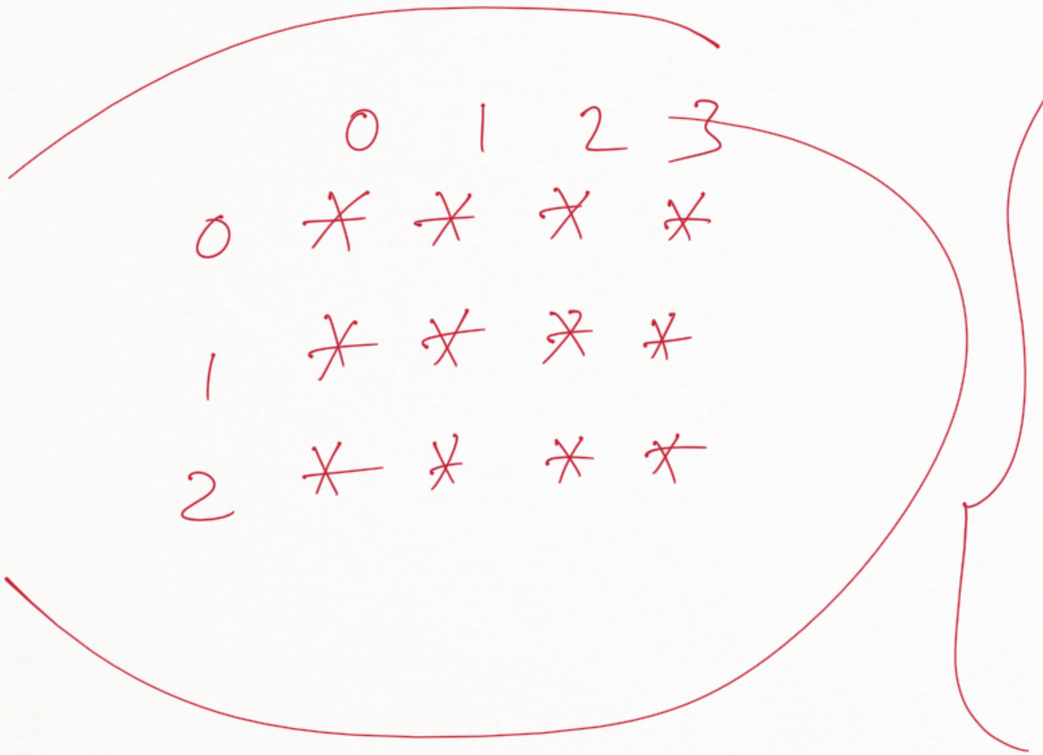
.

.

20

Nested for loop :-


```
for i in range(3):
    * * * *
```




```
for row in range(3):
    for col in range(4):
        print("*", end="")
    print()
```

0 → 1 *

1 2 * *

2 3 * * *

3 4 * * * *

1 2 3 4

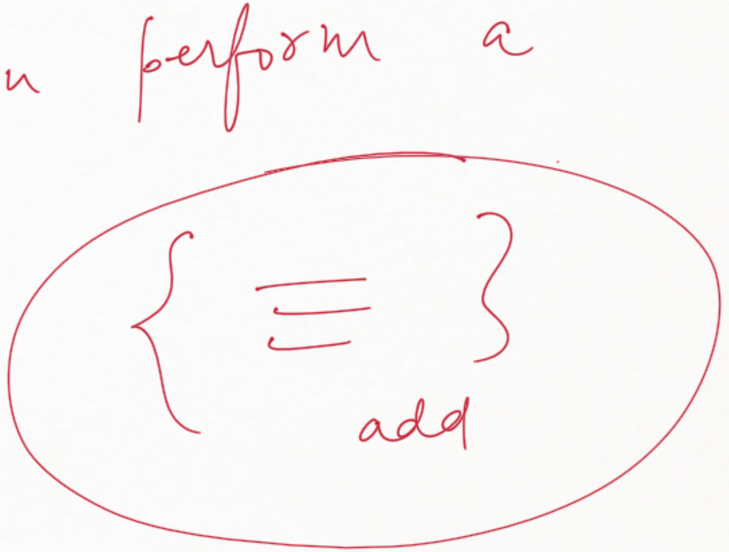
```
for row in range(1,5):
    for col in range(row):
        print(*, end)
    print()
```

File Handling :-

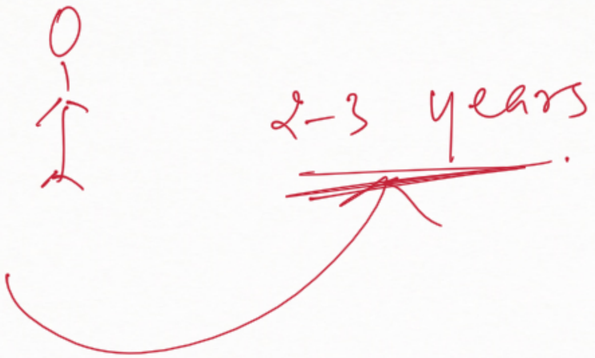
	mode
open()	'r' → read
write()	'w' → write / create
read()	'a' → append
close()	'rb'
	'wb'

Function in Python :-

Block of code, by which you can perform a specific task -



plumber →



defining a function → { ≡ }

calling a functn →

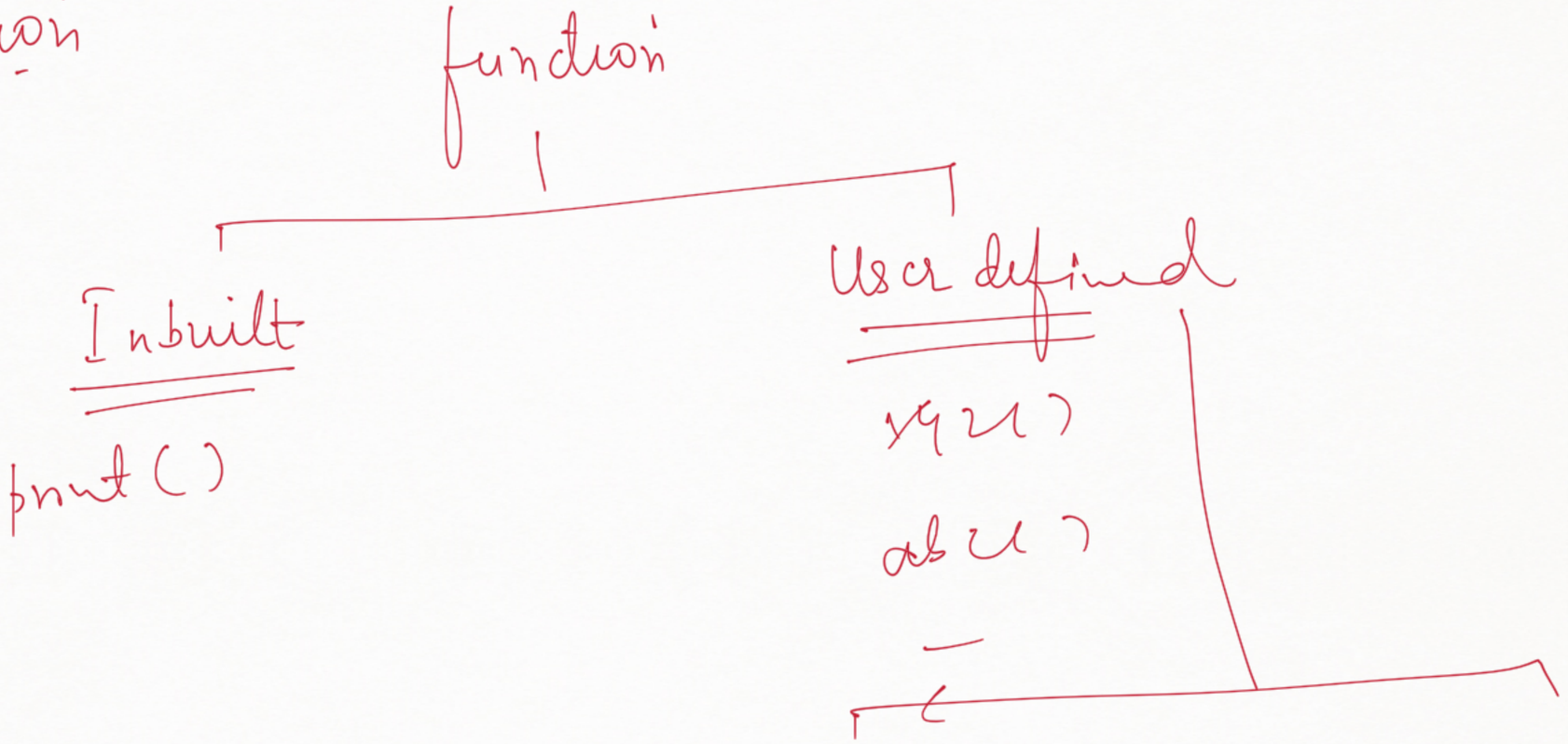
def function_name( ):

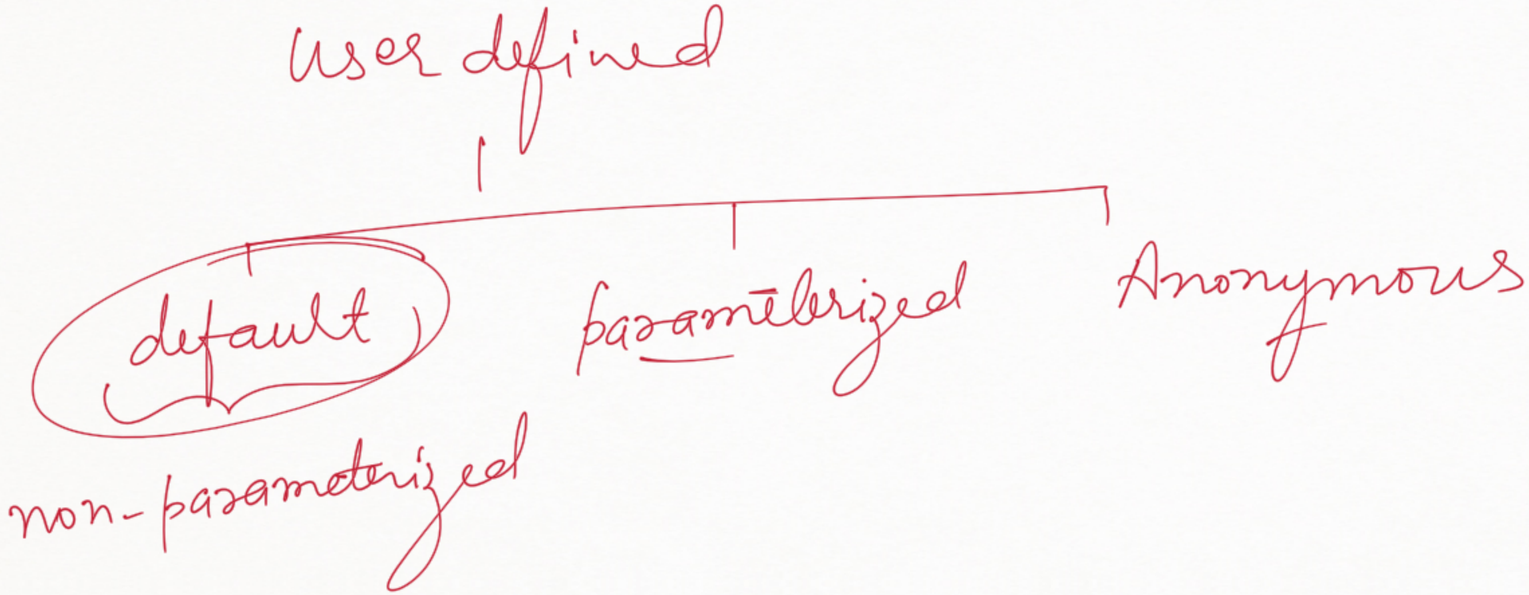
function_name( )

```
def name():
    print('hello')
    print("My name is Malleka")
    print('Bye')
```

```
name()
name()
hello
|
|
```

Types of function




User defined
default
parameterized
Anonymous
non-parameterized

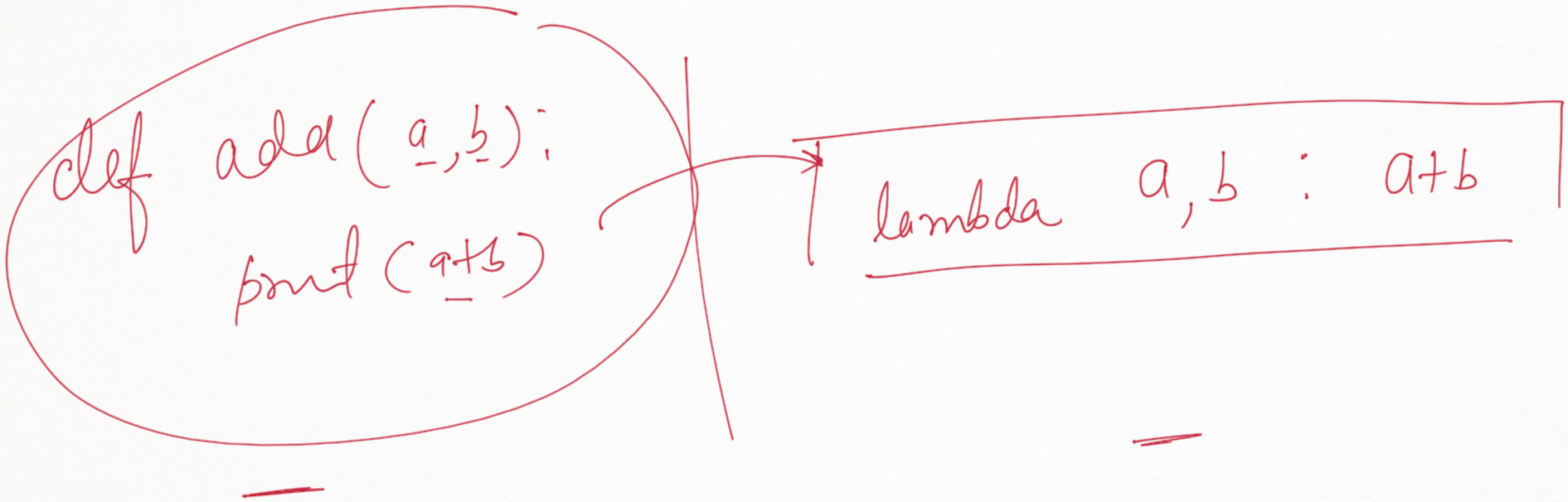
Anonymous function:-

function without name

or

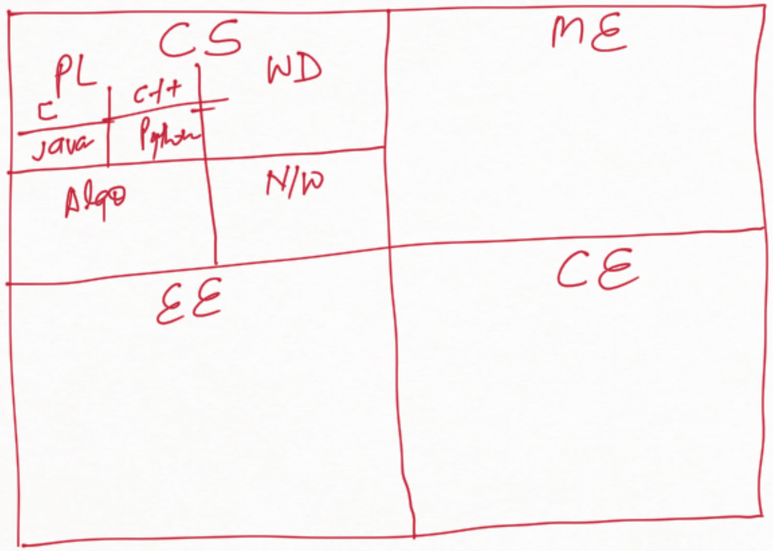
Nameless funct

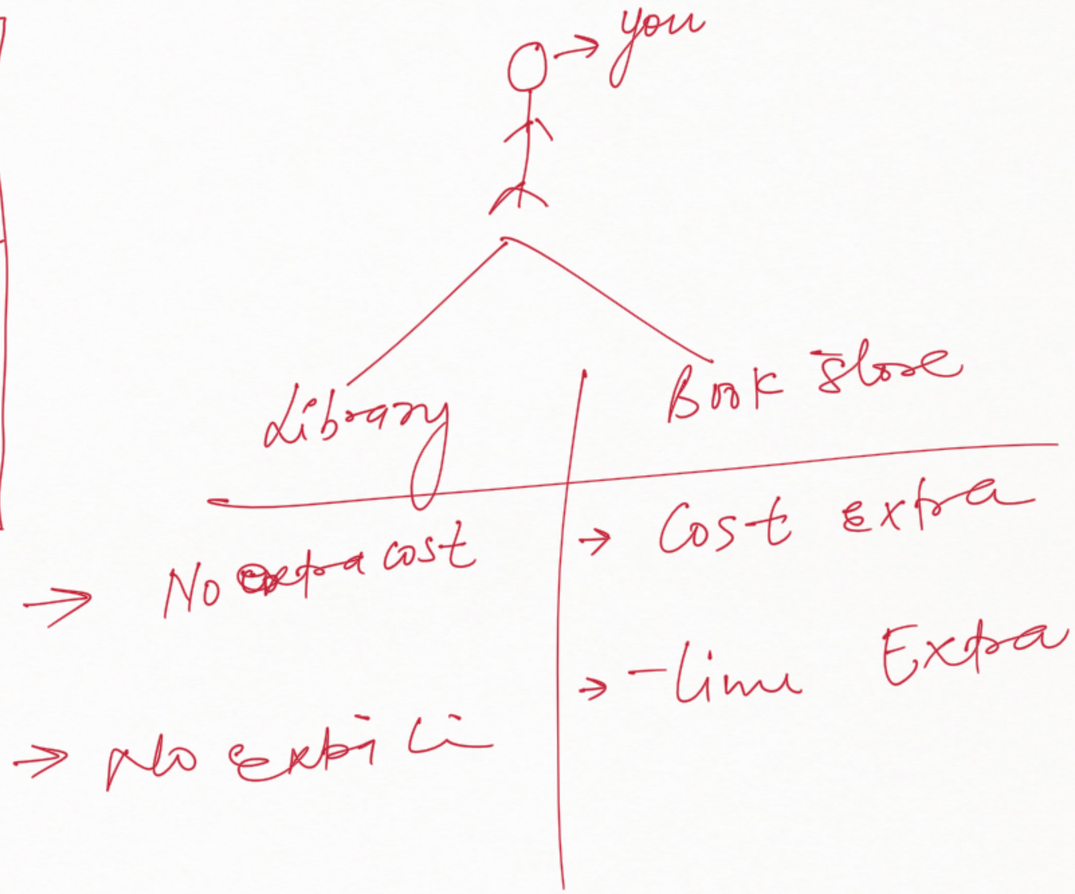
lambda

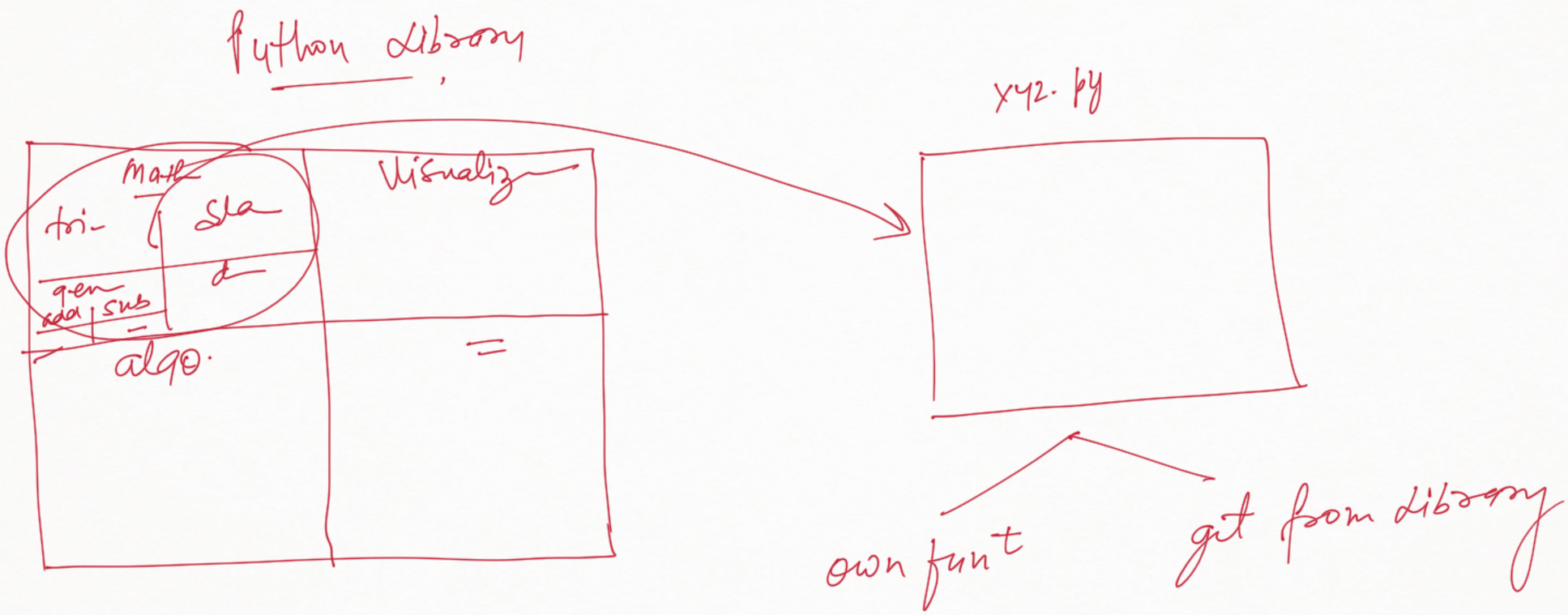

def add(a,b):
print(a+b)
lambda a,b : a+b

Packages and Modules :-

College Library






Python Library
Math
tri-
sta
d
gen
add | sub
algo.
Visualiz
=
xyz.py
own funct
get from library

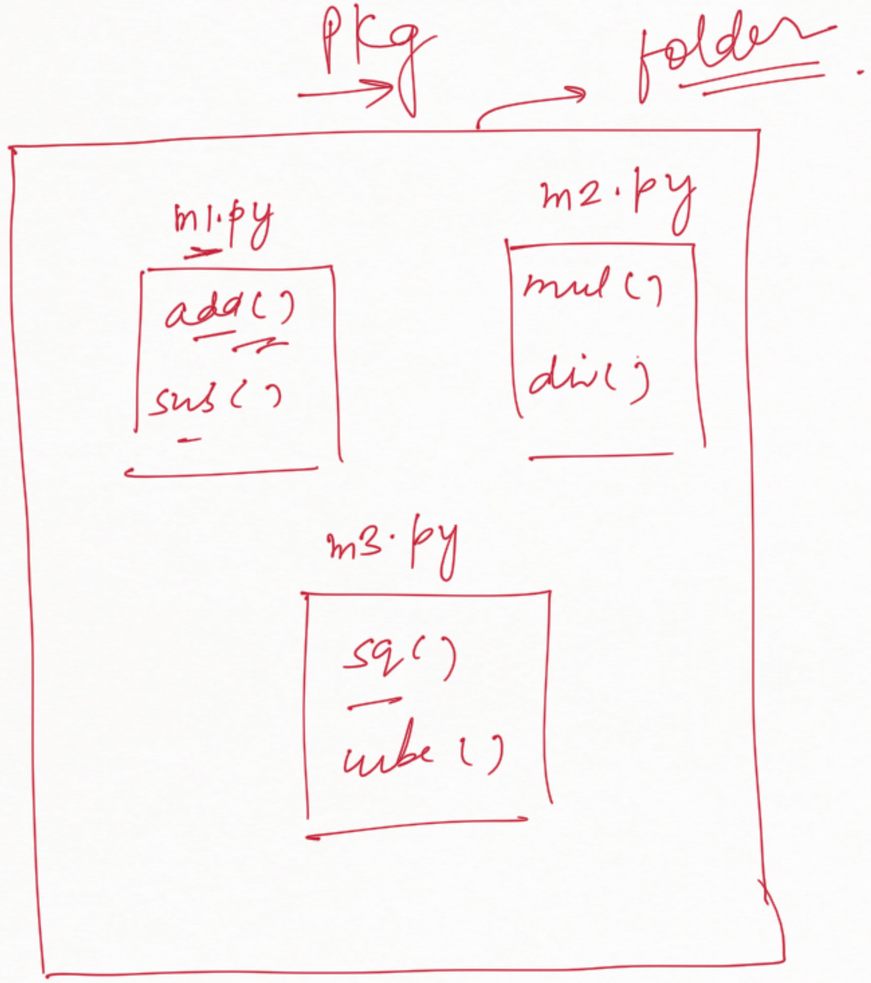
Library :- collection of packages →

package :- collection of modules

Module :- collection of function or

.py file

function :- _____ .

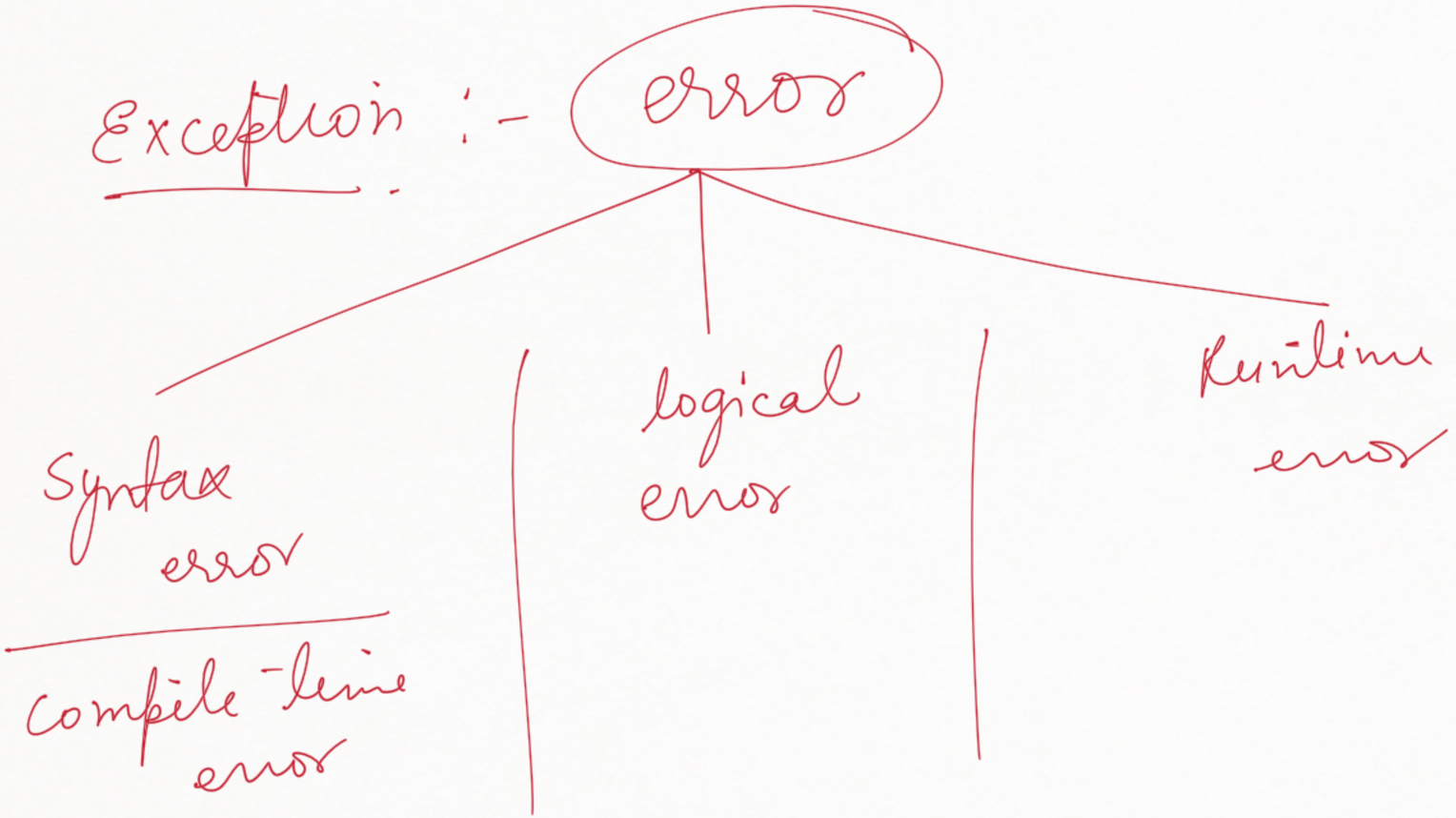


myfile.py

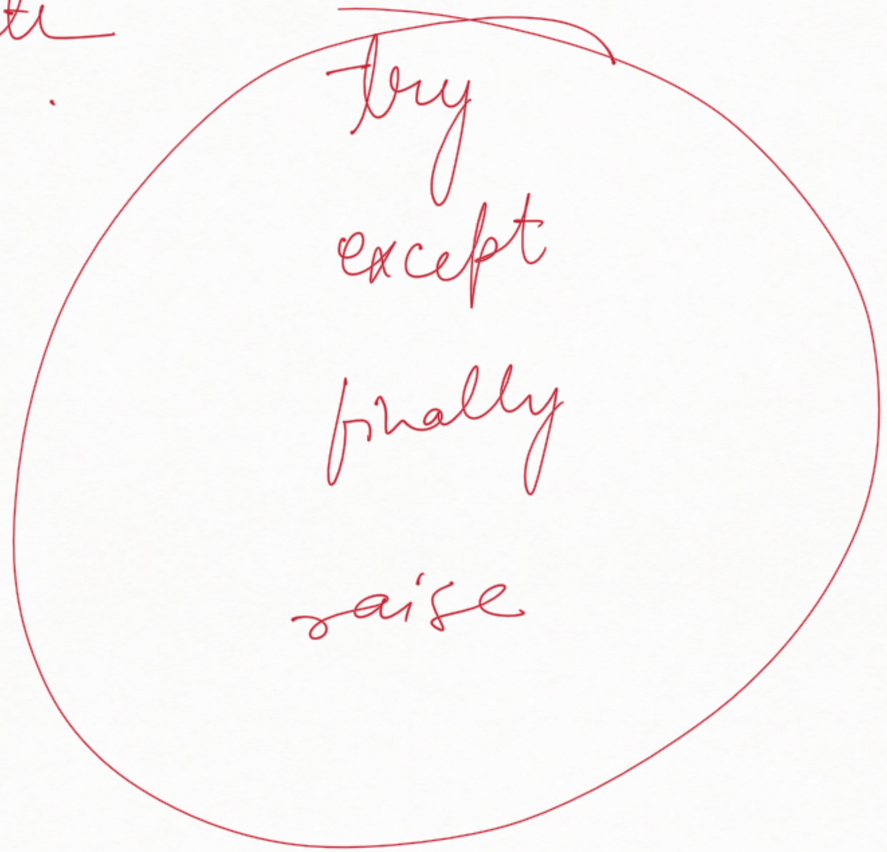
```
from pkg import m1,m3

m1.add()
m1.sub()
m3.sq()
```

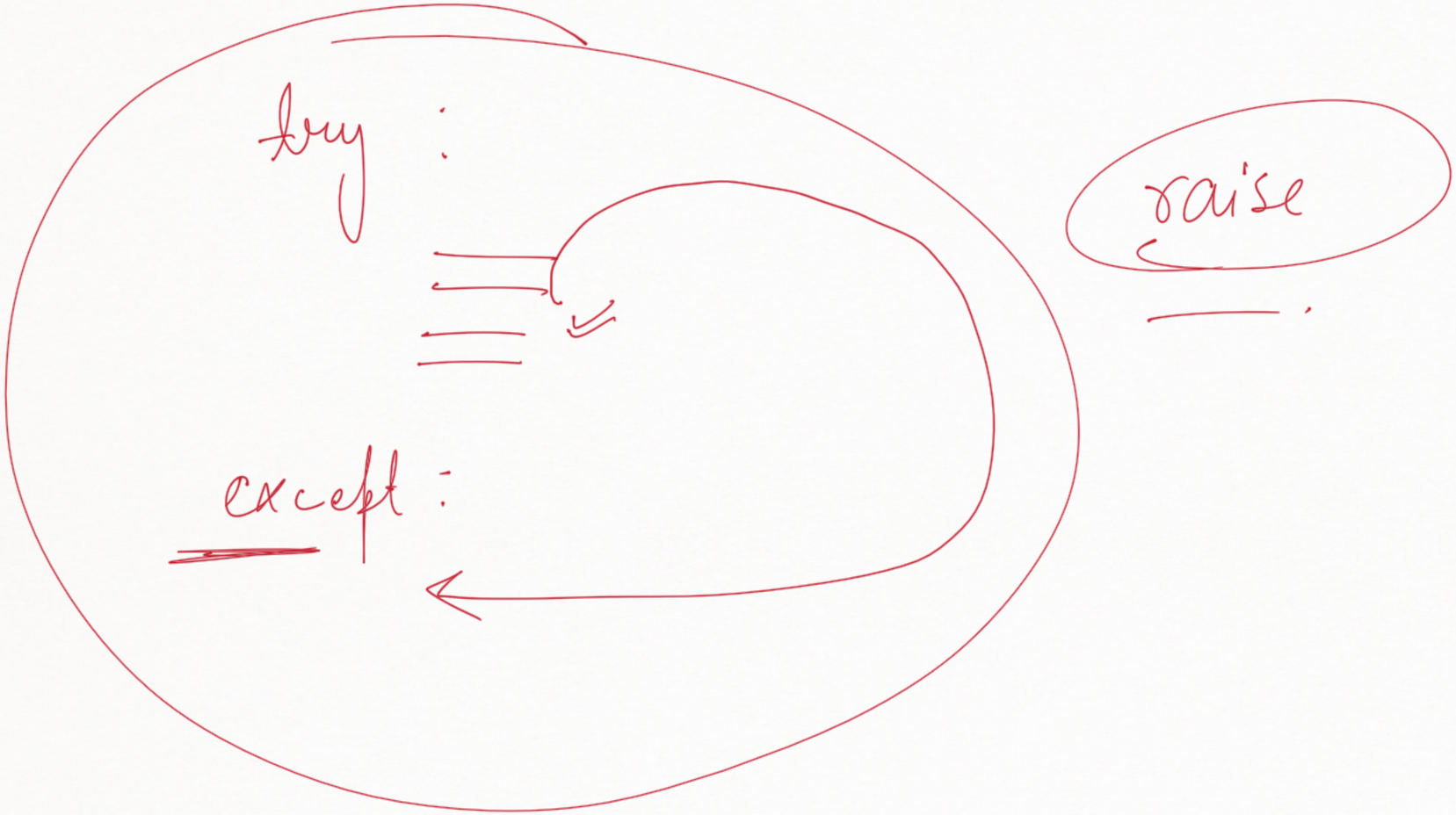
Exception handling :-

Exception :- error



Exception handling.





raise

Object Oriented programming s/m:-

OOPs :- Way of programming real world concepts

1. object
2. class
3. Abstraction
4. Encapsulation
5. Polymorphism
6. Inheritance

Object - real world entity / instance of class

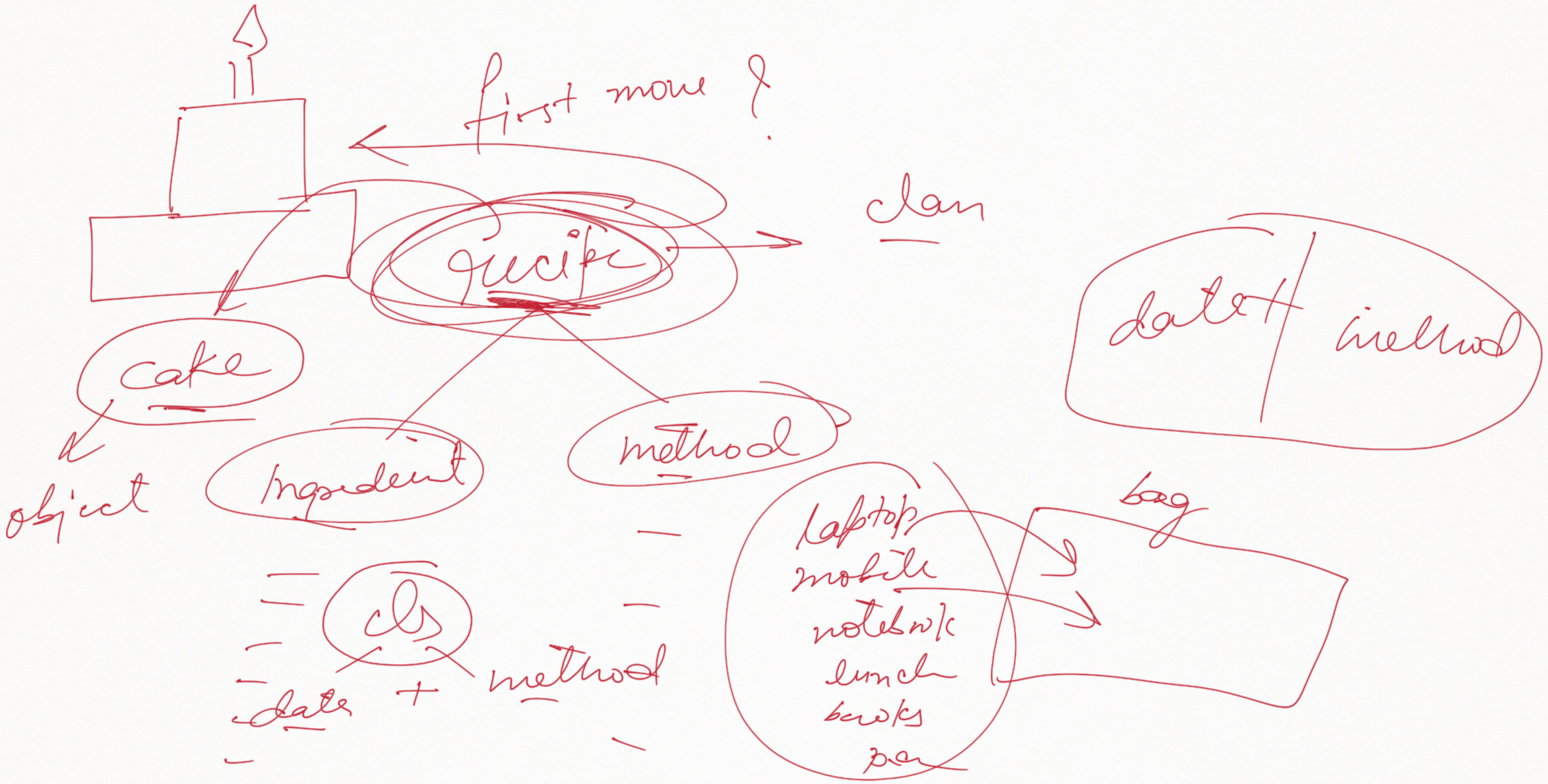
class - blueprint of object

Abstraction - Hide complexity, show essential

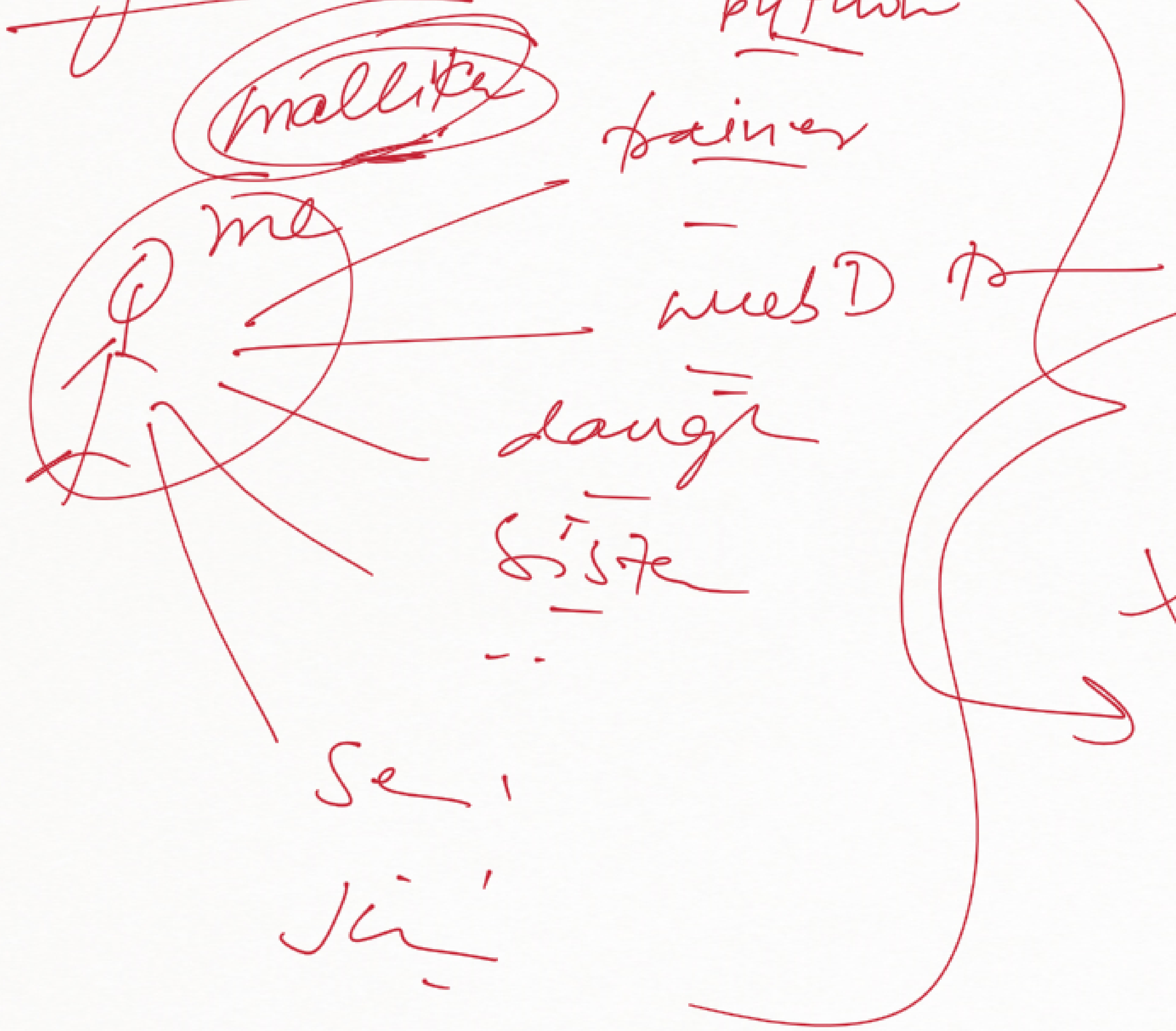
Encapsulation: Wrapping of data & method into a single unit

Polymorphism: Same name, many form / Emissional change, behav change

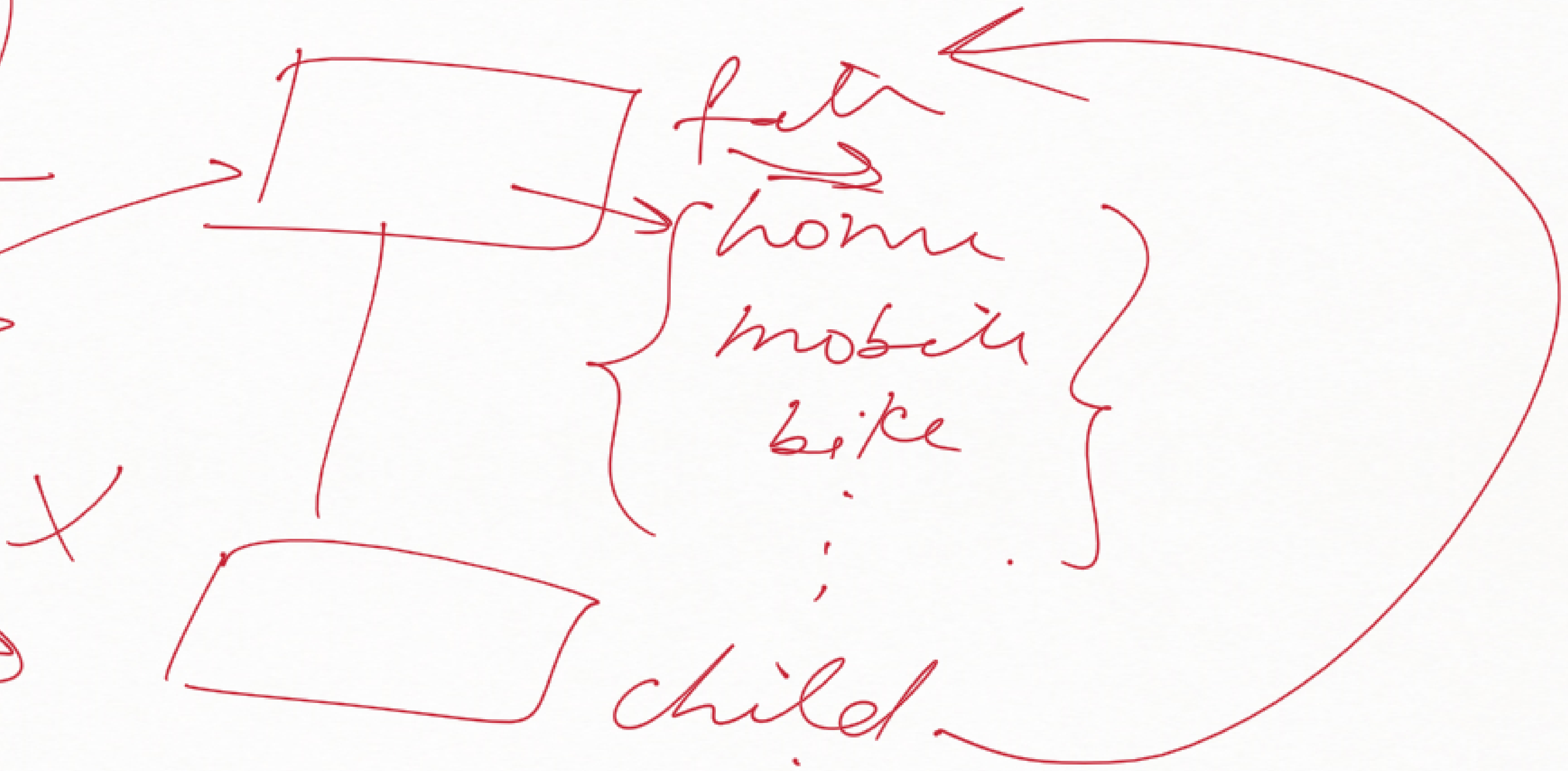
Inheritance: Reusability


first move
recipe
can
cake
object
ingredient
method
data + method
laptop
mobile
notebook
lunch
books
pen
bag
data
method

Polymorphism :-



Inheritance :-



class and object.

Constructor :- function (special function)
initialize the object.

__init__()

→ default

→ parameterized

__init__(a, b, ...)

class A:

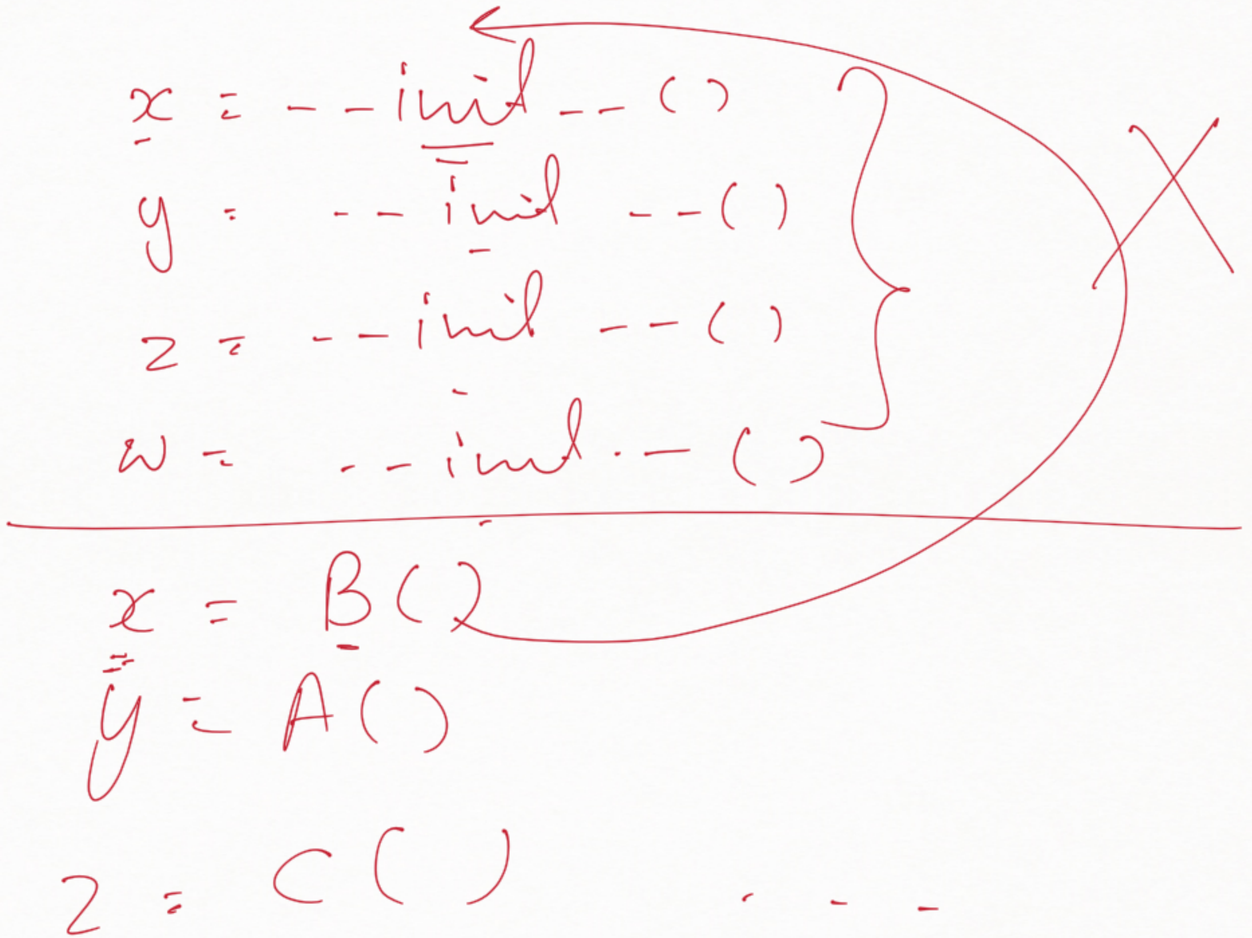
    ==

cls B:

    ==

cls C:

    ==



self → variable
point your current object.

Types of method :- 3

- instance method → instance var (self)
- cls method → cls var
- static method → instance or cls var

Abstraction & Encapsulation :-

↳ Hide the complexity, show the essential

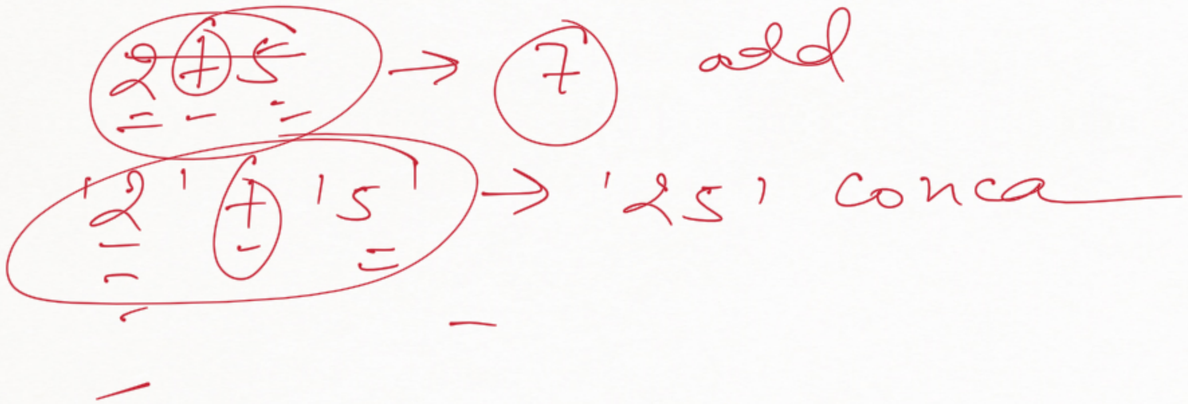
2 + 3

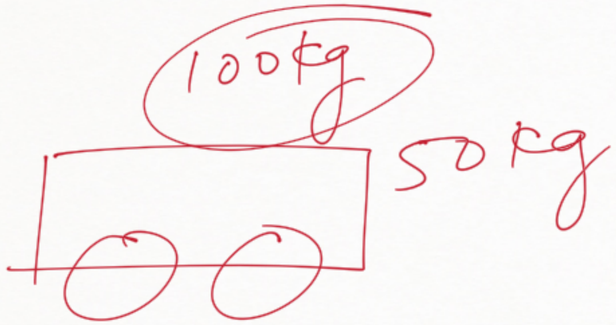
⟶

Polymorphism

Polymorphism :

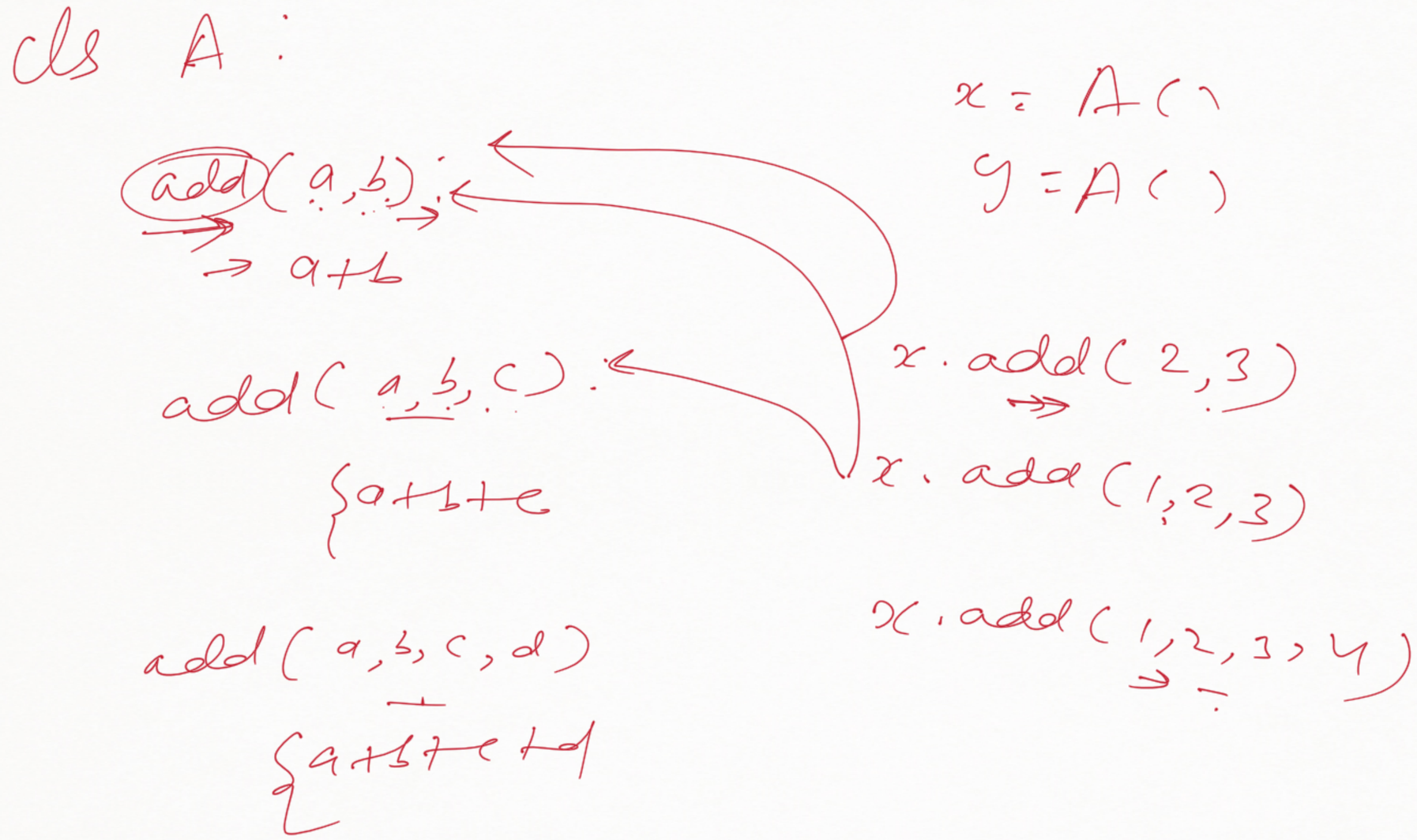
Same name, many forms.





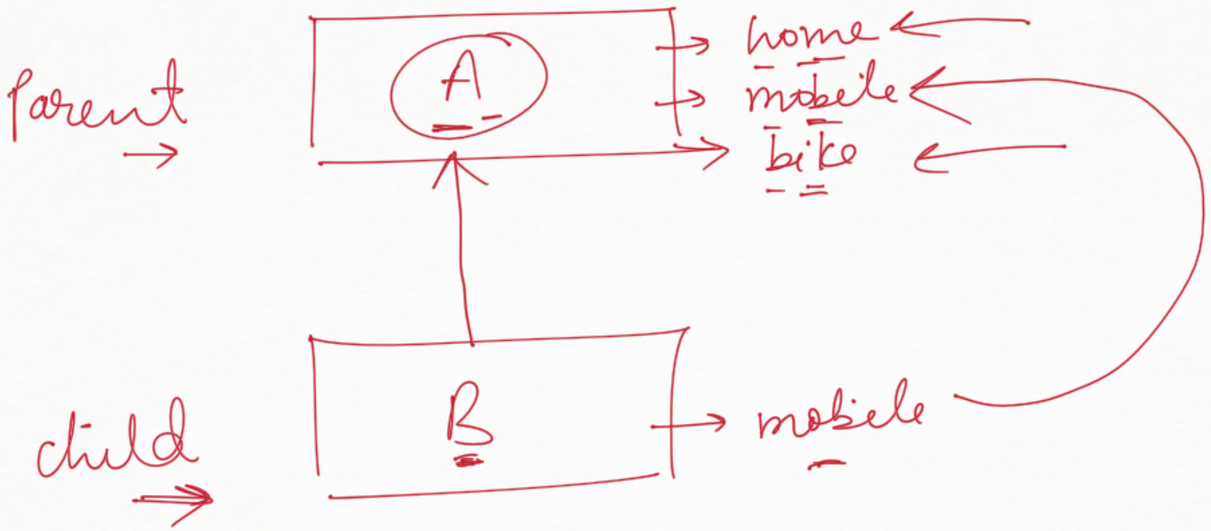
Polymorphism

- Overloading
  - operator
    - \+
    - \+
  - method
- Override
  - method


cls A :
add(a,b):
→ a+b
add(a,b,c):
{a+b+c
add(a,b,c,d)
{a+b+c+d
x = A()
y = A()
x.add(2,3)
x.add(1,2,3)
x.add(1,2,3,4)

Inheritance :- Reusability →

Accessing, inheriting or reusing the property of parent class.



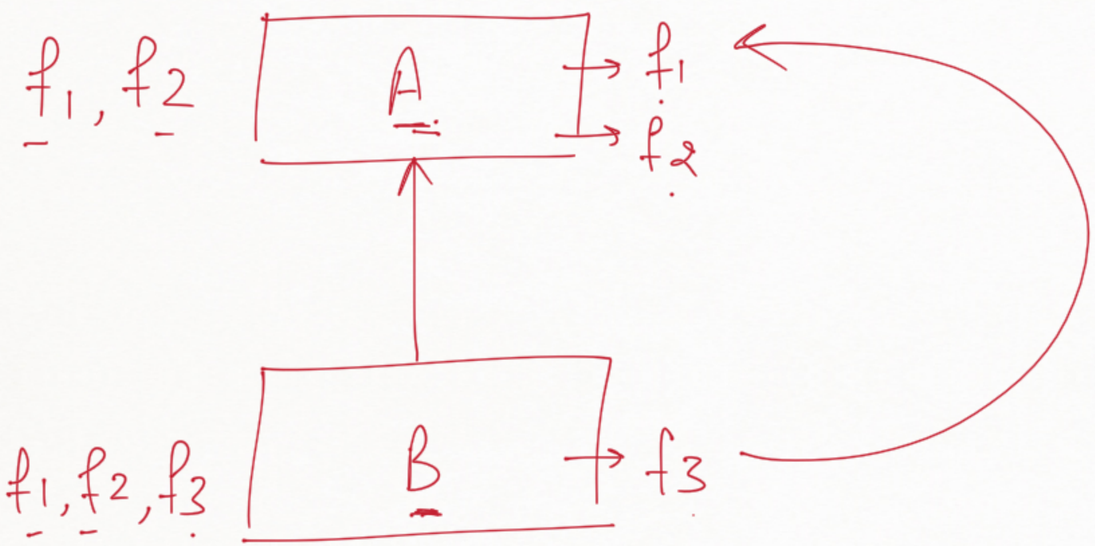
Note. child class can access all the property of parent cls but parent class can't access any property of child class.

Parent cls / child cls or Base cls / derived cls or Super cls / Sub cls

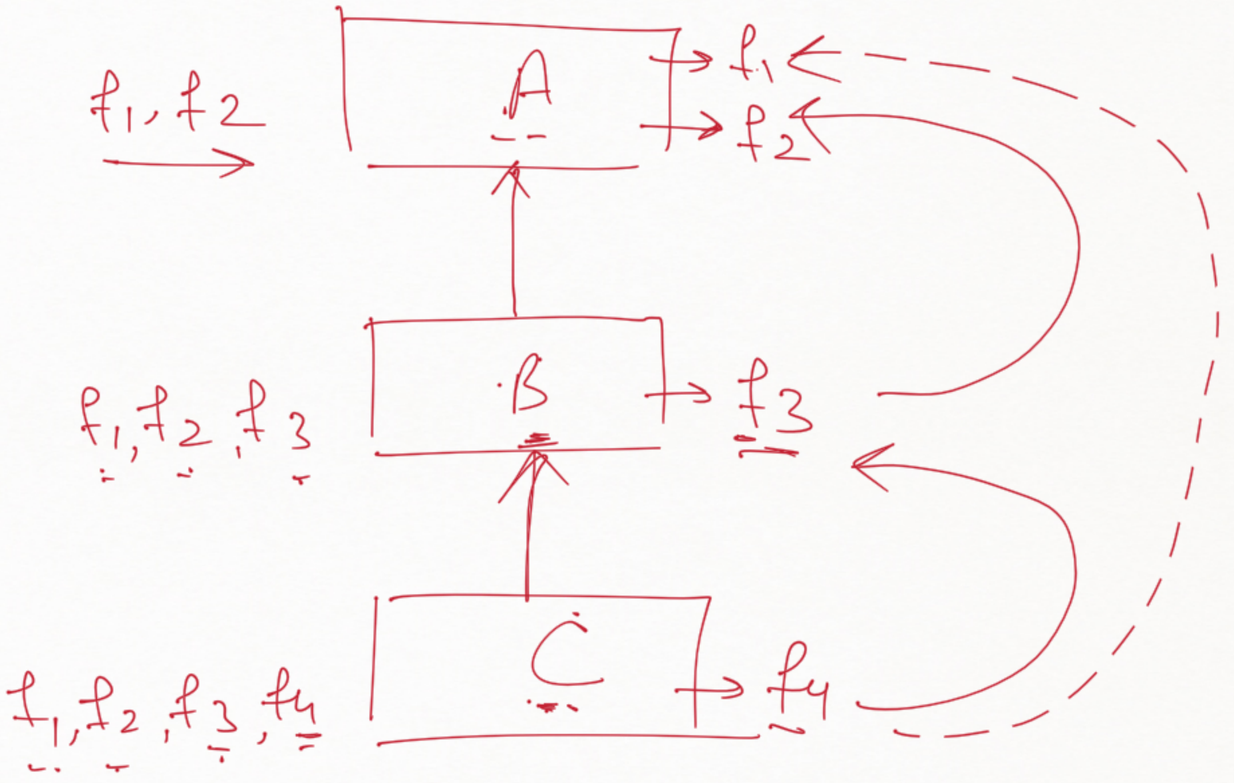
5 types of inheritance :

1. single level
2. Multi° level
3. Hierarchical
4. Multiple
5. Hybrid

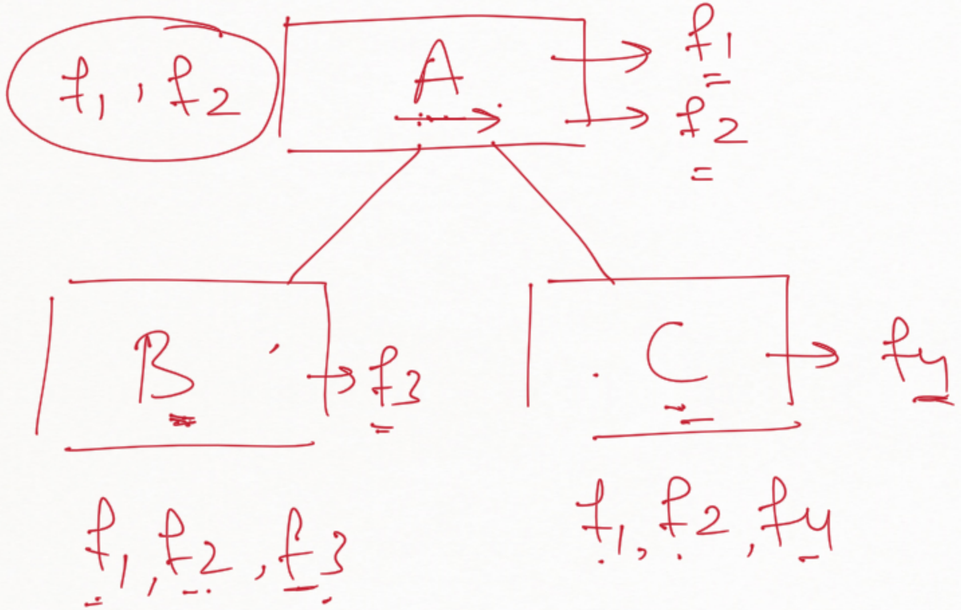
Single level



Multi level

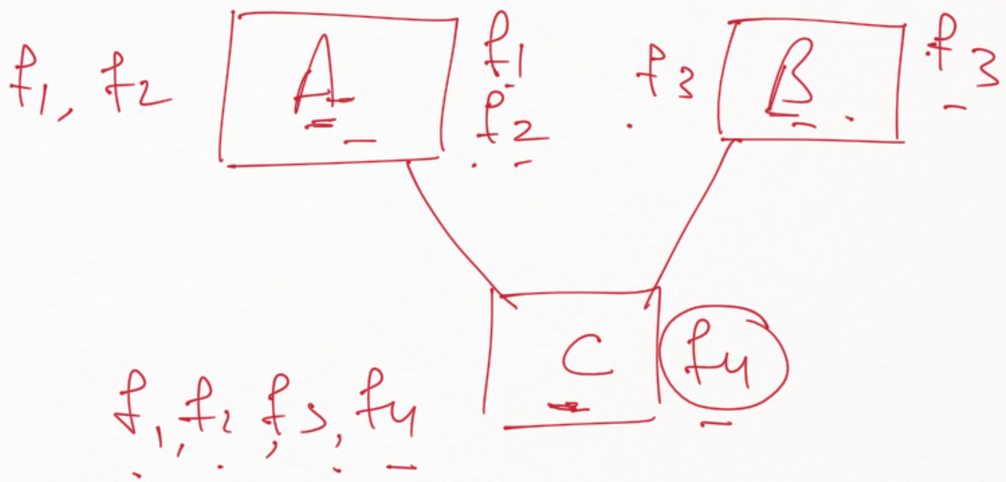


Hierarchical



only one parent and multiple child

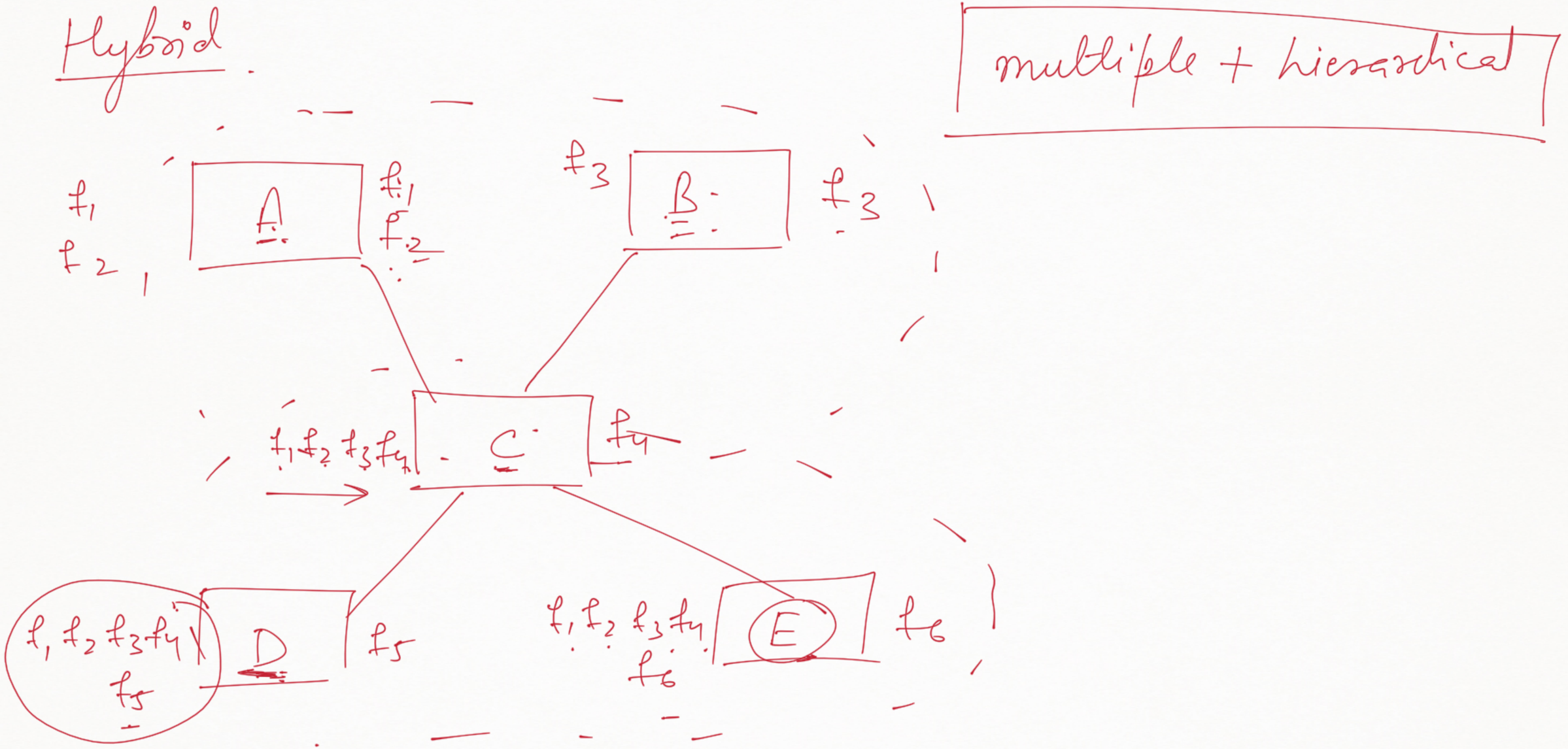
Multiple



more than one parent and only one child

Hybrid

multiple + hierarchical



Method Overriding :-

```
cls A:
    f1()
    =

cls B:
    f1()
    -
```