04-05-2025

Inheritance

Decorators

method Overriding :—

```
class A
    def write(self, ....):
        ...

class B(A)      → parent
    def write(self, ....):
        ...
```

B() → obj → write of B

Suppose Below is Java/C++

```
def add(a, b):
    a+b
```

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

add(2,3) → def add(a, b)

add(2,3,1) → def add(a, b, c)

Overloading

def booktickat ([name], [age])

def book ticket (name , age)

booktickat ([kinal] , [30])

list list

bookticket (kinal , age)

str str

c++ , Java

→ void add (int a , int b)

1000 line

→ void add (double a, double b)

~~vm~~

compiler

object for execution

Java → data type

c++

void (int, int)

void (double /doubl)

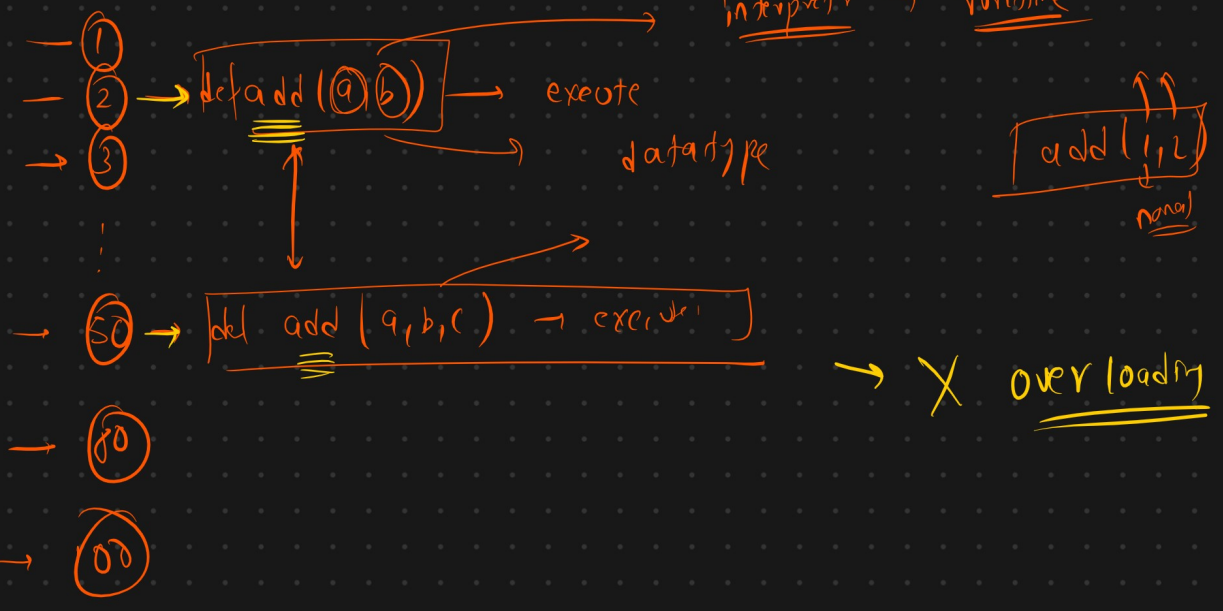
python

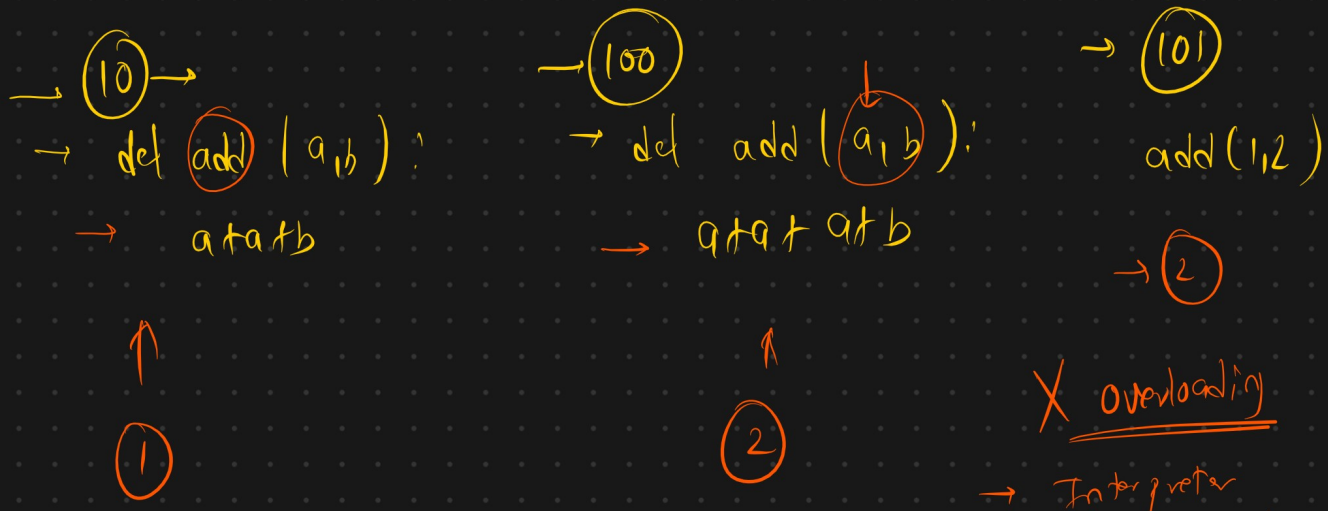
→ def add (a, b):

a+b

def add(a,b)

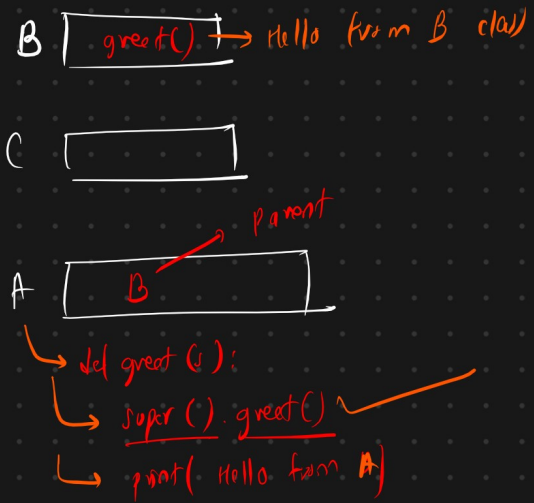
a+b


interpreter → runtime
1
2
3
def add(a,b)
execute
datatype
add(1,2)
manual
50
def add(a,b,c) → execute
→ X overloading
80
00


10
def add(a,b):
a+a+b
1
100
def add(a,b):
a+a+a+b
2
101
add(1,2)
2
X overloading
→ Interpreter
→ dynamic datatype allocation.

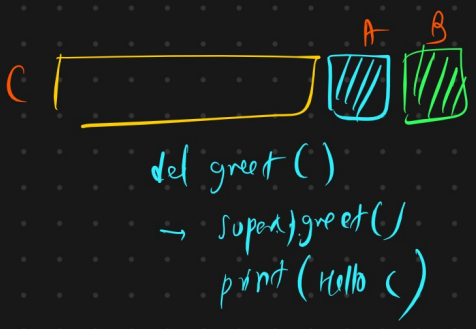
Inheritance

```
class B:
    def greet(self):
        print("Hello from B")

class C:
    def greet(self):
        print("Hello from C")

class A(B):
    def greet(self):
        super().greet()
        print("Hello from A")
```




```
a = A()
a.greet()
```

→ ① Hello from B
→ ② Hello from A.

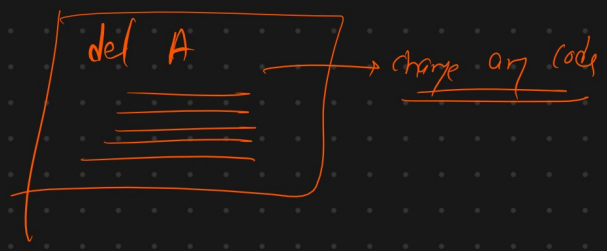


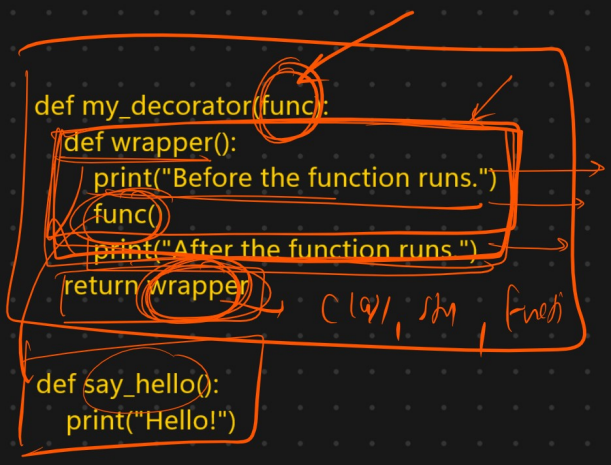
MRO → Method Resolution Order

↓

Determines which method to call.

special functions





① create a function which does something with another function

② create a function that accepts function & pass to step ①

abc = mydecorator(say_hello)

```
def add(a,b)
    c = a+b
    return c

d = add(1,2)
```

@time_it
func A

@time_it
func B

@
func C

func D