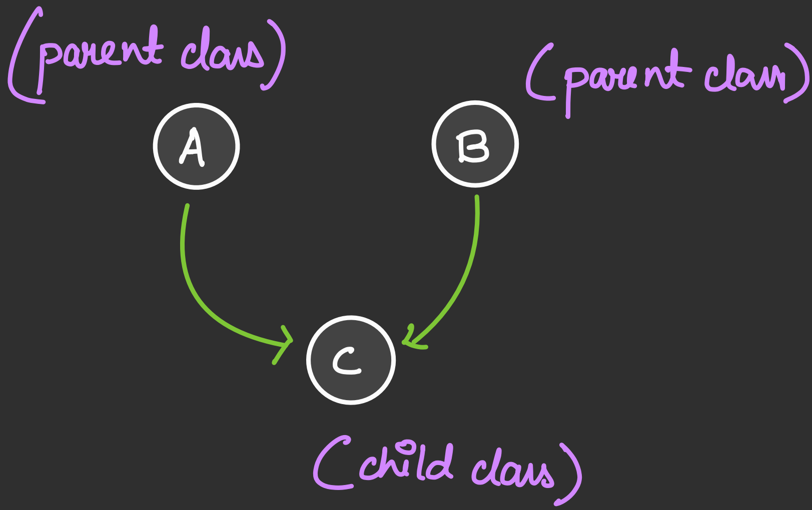
Lecture 3: Functional Programming

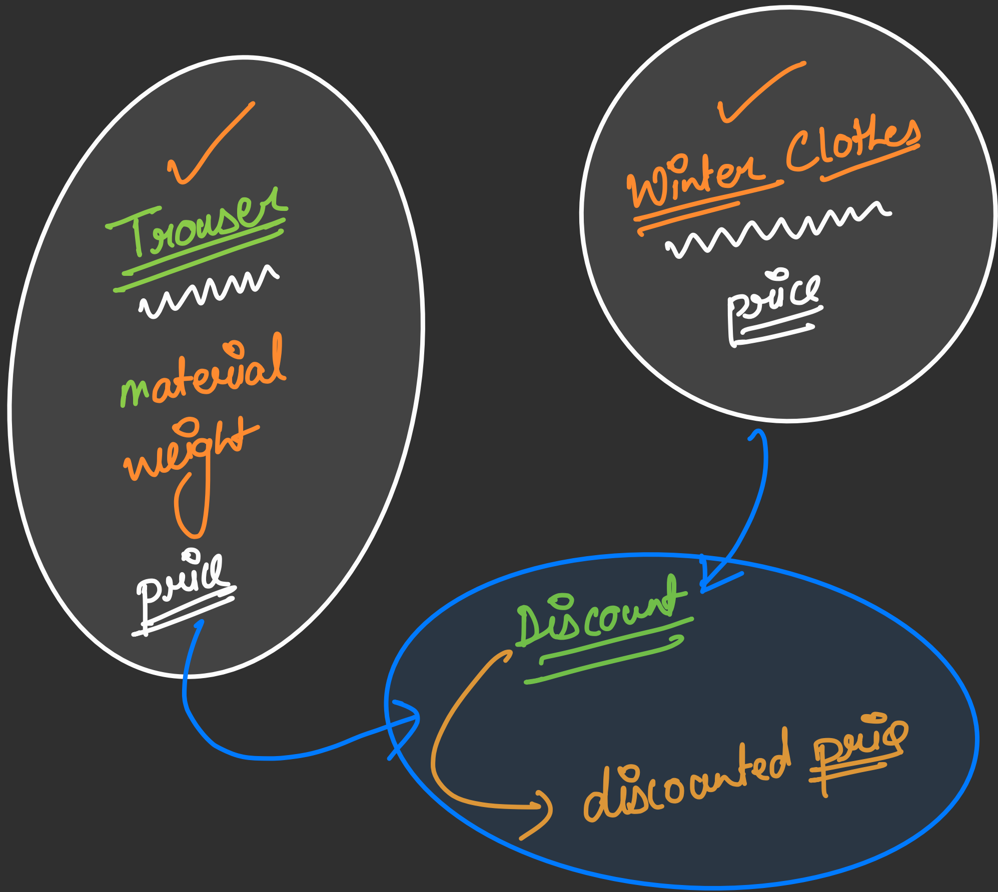
Agenda

1. Multiple Inheritance & MRO
2. Functional programming
3. Lambda Functions
4. Higher Order Functions
5. Decorators [Next lecture]

Multiple Inheritance

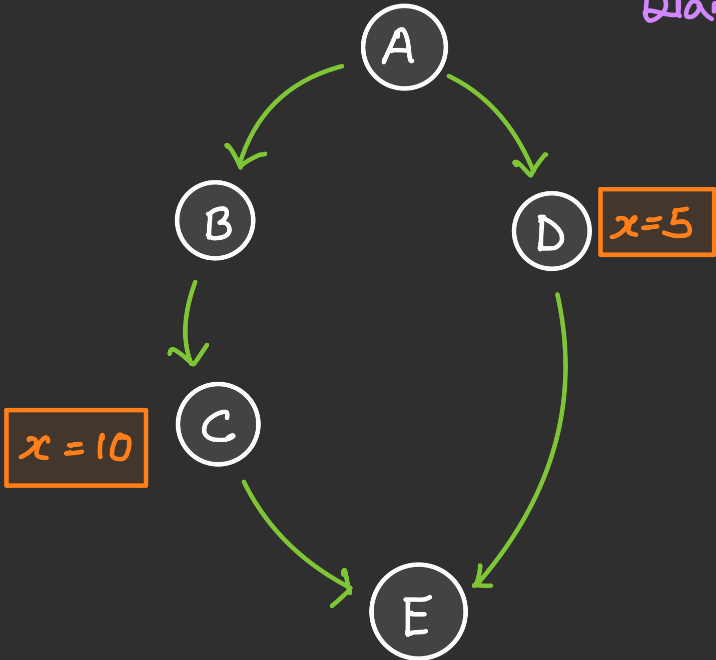


Retail Industry

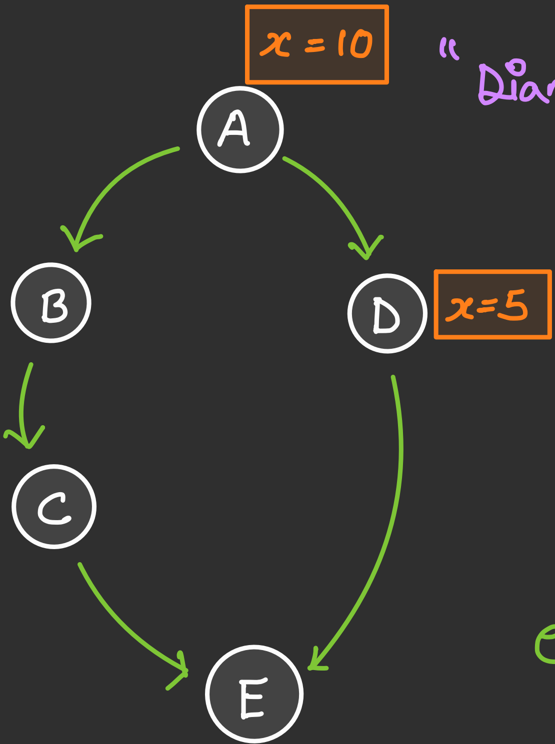


Method Resoultion Order (MRO)

"Diamond shape Inheritance"



Method Resoultion Order (MRO)

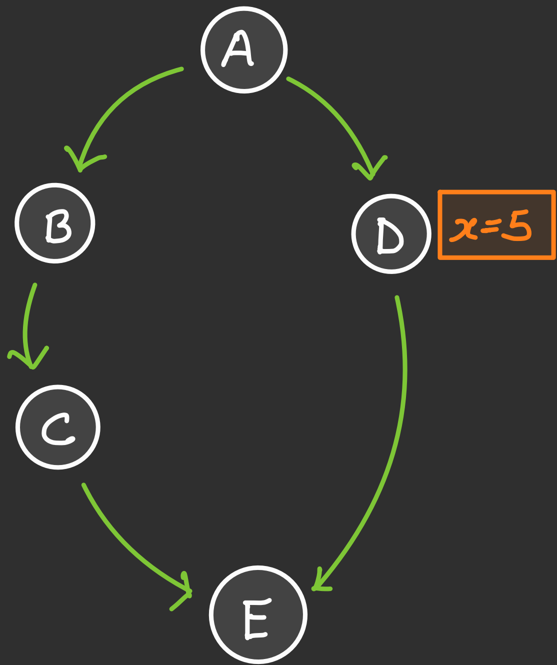


"Diamond shape Inheritance"

1. Left to Right
2. We go to parent when all child are covered

e = E() ⟶ $x=10$

Taking closure E



E → C → B → D → A

Functional Programming

Programming Paradigm → Set of rules

OOPs

1. Abstraction
2. Encapsulation
3. iheritance
4. polymorphism

Functional programming

Function programming

It is a paradigm where computation is viewed as the evaluation of mathematical function

a = [1, 2, 3, 4] ⇒

```
squared_num = []
for num in a:
    squared_num.append(num * num)
```

~~loop~~ & ~~a*a~~

Two major Reasons

1. Immutable Data : It doesnot affect the original data.

2. Declarative Style : In FP, you focus on "what you want to acheive rather than "how" to acheive it"

a

def anything ( )

return

Imperative Approach

a = [1, 2, 3, 4] ⇒ squared_num = []

for num in a:
    squared_num.append(num * num)

↳ loop & a*a

Functional Approach

↳ map, reduce, filter

a = [1, 2, 3, 4]

userdefined func

map(func, a) → list

(1, 4, 9, 16)

def gen-exp(n):

def exp(x):

return x^n

return exp

5

2

$2^5 = 32$

x

exp5(2)

exp5 = gen_exp(5)

funⁿ