Functional Programming

→ Zip ← Takes two or more iterable

→ list1 → [1, 2, 3]

list2 → ['a', 'b', 'c']

Returns

→ list of Tuples

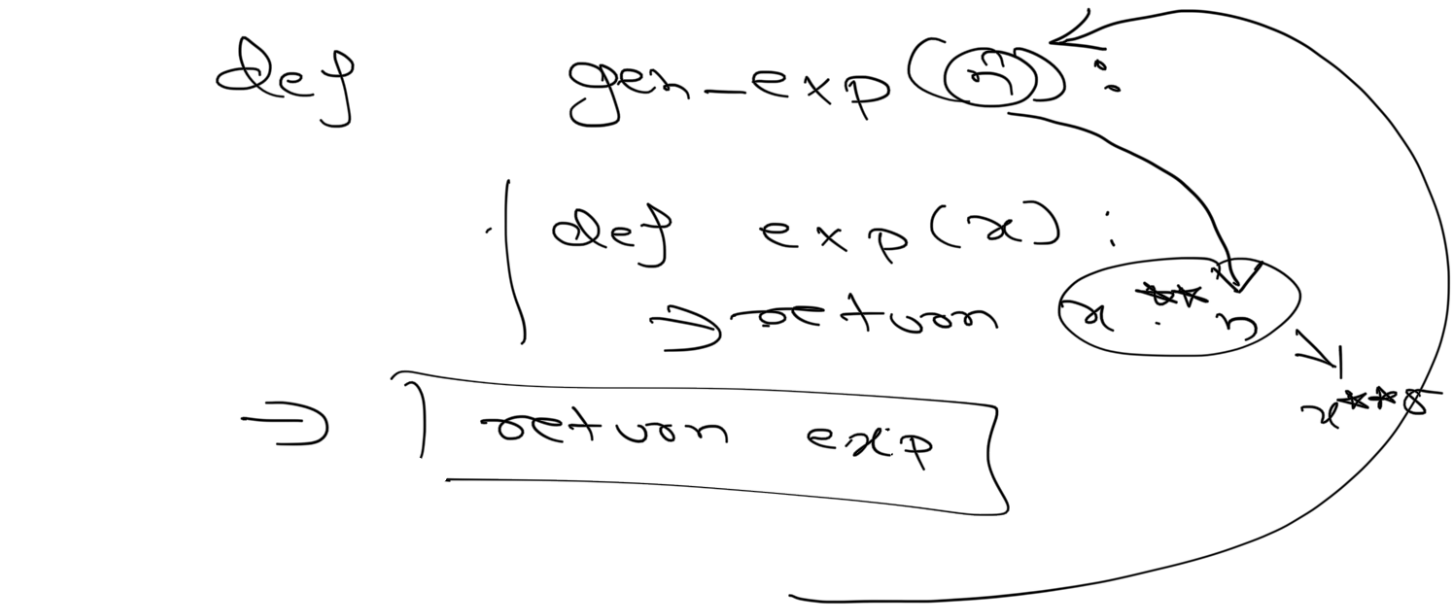
list(zip(list1, list2))

[(1, 'a'), (2, 'b'), (3, 'c')]

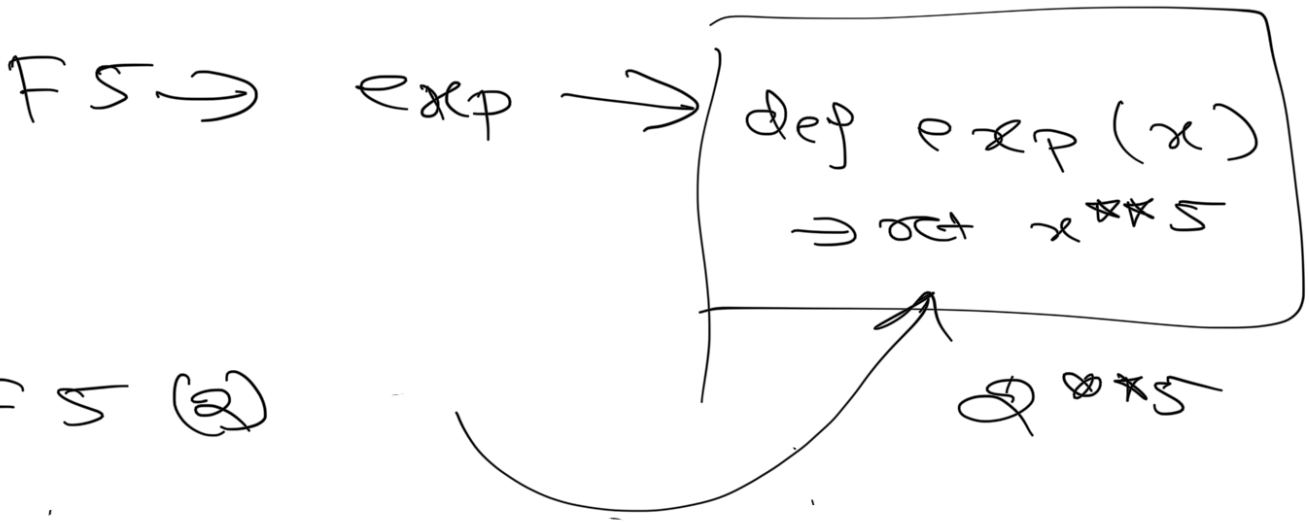
3 Tuples
Stored inside a list

→ Higher Order function (HOF)

→ A function which returns another function

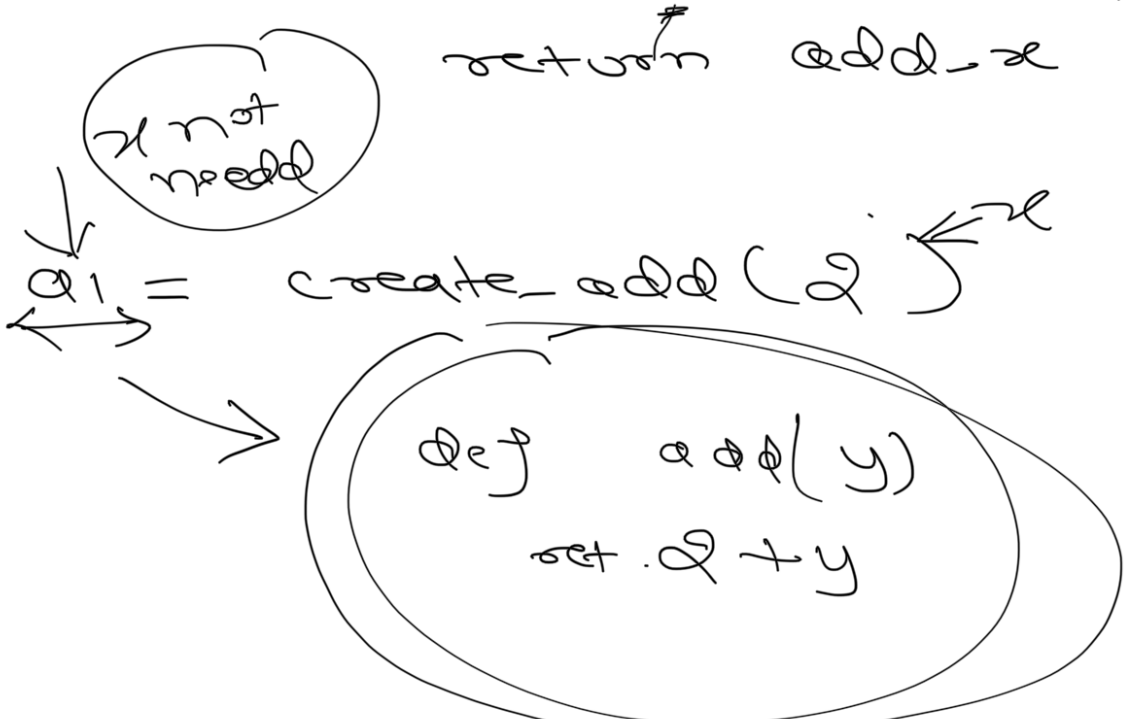


F5 ≡ gen_exp(5)



3) creat HOF to sum two numbers.

```
def create_add(x):
    def add_x(y):
        return x + y
    return add_x

a1 = create_add(2)
```



q1 (4) ⇒ 6

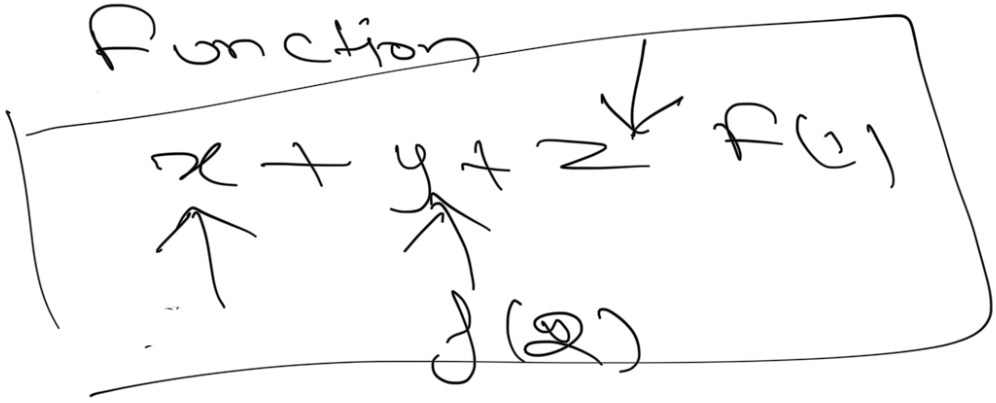
---

q2 ⇒ add_x(3) (O1)

↓

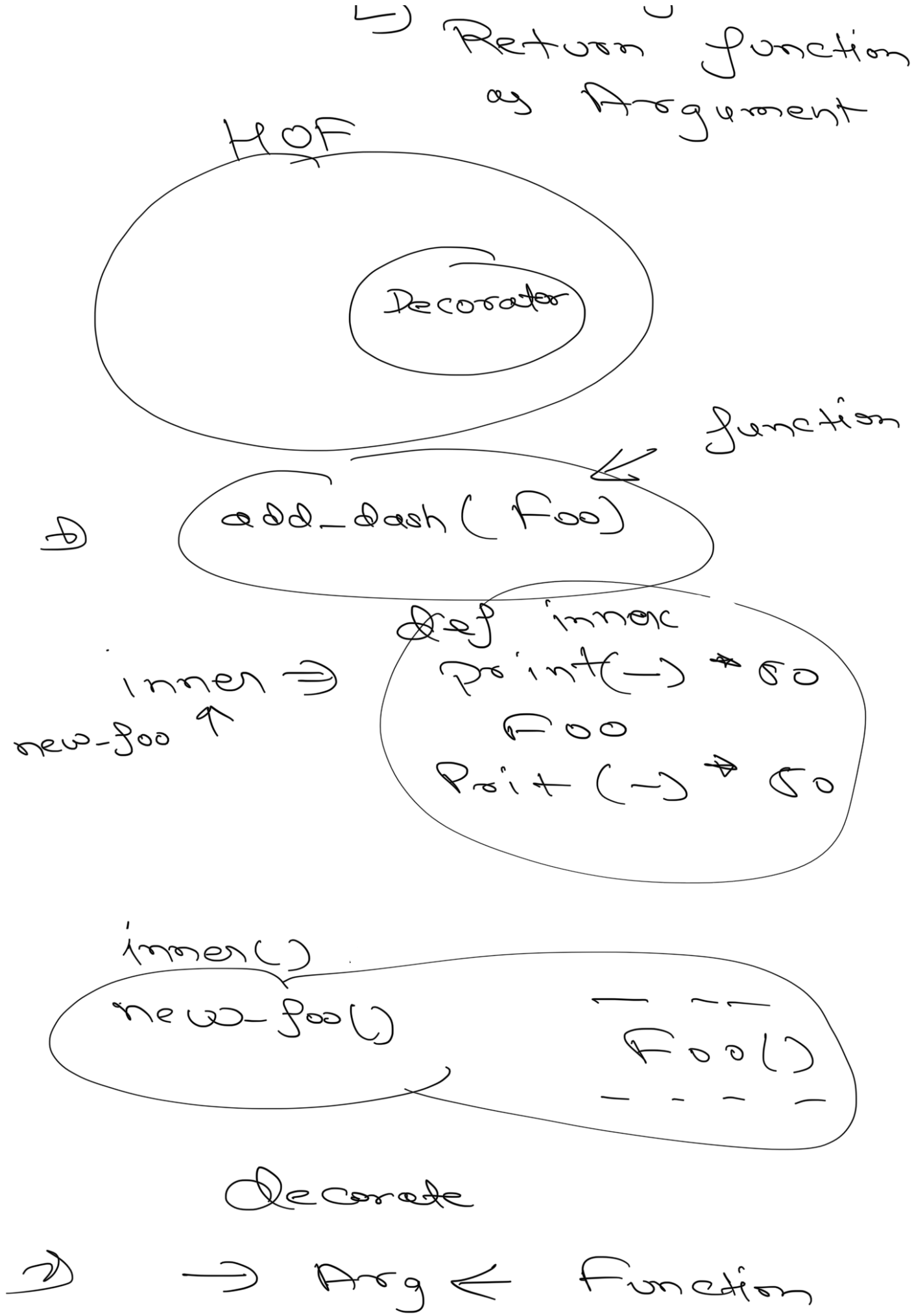
```
def add(y):
    return 3+y
```
} O(1)

H.W return a func
inside a function
which is already
function



→ Decorators

↳ Take function
as Argument


Return function as Argument
HOF
Decorator
function
add_dash(foo)
def inner
Print(—) * 50
foo
Print (—) * 50
inner ⇒
new-foo
inner()
new-foo()
foo()
Decorate
Arg
function

Output ← Function

Application

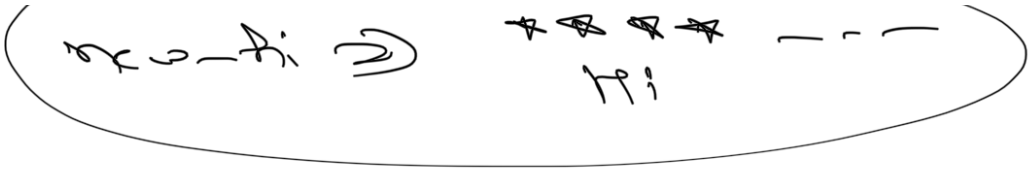
→ adds common functionalities to other funcs

```
def pretty(func):
    def inner():
        print("*"*50)
        func()
    return inner
```

```
def Hi():
    print("Hi")
```

Hi()
Hi

```
newHi = pretty(Hi)
```




```
def printHello()
    print('Hello')
```

```
printHello()
'Hello'
```

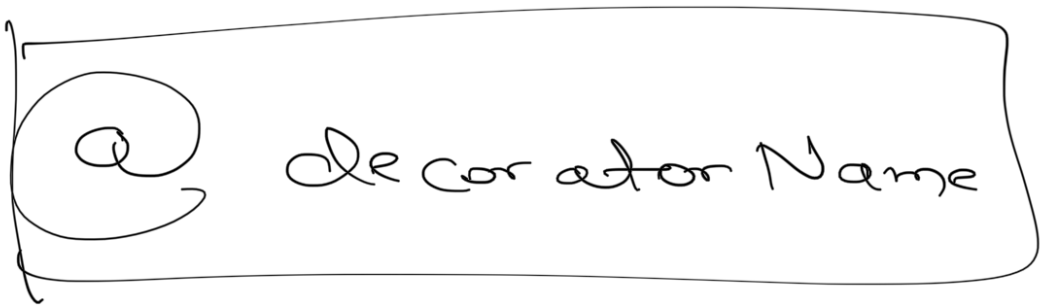
```
new_H = Pretty(printHello)
new_H()
****
Hello
```

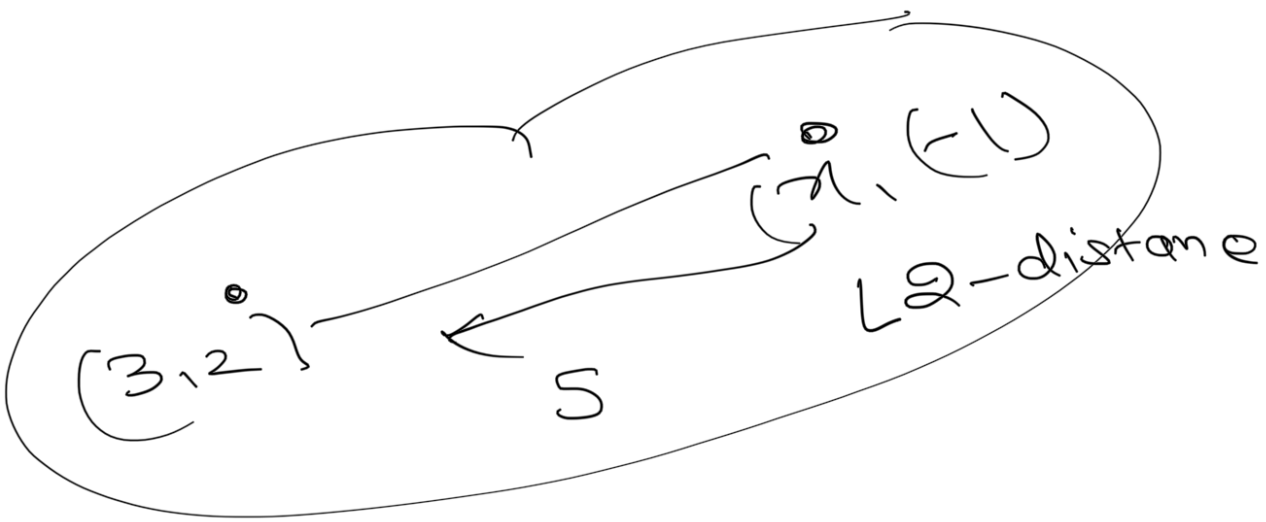
```
@pretty
def PrintHello()
    print('Hello')

PrintHello()
```

*

* * * * * - —

Hello





eq of line ?

$m \leftarrow$

c

$x^2 + c = 0$

⇒ LL Marks

Pair of Friends X

⇒ 3 pair of Students have 2 units

[(A,B) (B,C), (C,D)]

⇒ name of Second lowest

name ⇒ [D] ← list "D"

name ⇒ [A, B]

(B : 2 , C : 1 D : 2 , E : 3)

B and D ⇒ [B, D]