11/4

Decorators :-

It will decorate the given function.

Ex :-
```
def make_pretty(func):
    def inner():
        print("I got decorated")
        func()
    return inner

@make_pretty
def Ordinary():
    print("I'm Ordinary")
Ordinary()
```

O/P :- I got decorated
I'm Ordinary.

So decorators are mainly used coheneven we want to change the functionality but without changing the code functionality.

@make_pretty means we are giving the power to ordinary function.

As @make_pretty is defined somewhere else & we are giving the functionality to ordinary function

To get more clarity on above program, let us see below

```
def extraordinary(link):
    def inner():
        print("I got decorated")
        link()
    return inner

@extraordinary #decorator
def ordinary():
    print("Im Ordinary")
ordinary()
```

O/p:- I got decorated
Im Ordinary.

Iterators:- In this one by one Elements we get.

```
for i in [1,2,3.5]:
    print(i)
```

O/p:- 1
2
3.5

```
for i in 1:
    print(i)
```

O/p:- TypeError: 'int' object is not iterable

```
list1 = [1,2,3]
for i in list1:
    print(i)
```

o/p:
1
2
3

Here without iterators it prints all elements.

Using Iterators

```
list1 = [1,2,3]
i = iter(list1)
print(next(i))
```

o/p:- 1

```
list1 = [1,2,3]
i = iter(list1)
print(next(i))
print(next(i))
print(next(i))
```

o/p:
1
2
3

```
print(i)
```

o/p: <list_iterator object at 0x22[illegible]97>

~~for i in i:~~
~~print~~

```
list1 = [1,2,3,4,5,56]
i = iter(list1)
print(next(i))
print(next(i))
print(next(i))
for j in i:
    print(j)
```

o/p's 1
2
3
4
5
6

Note: Here it is not printing everything, it is printing the number whichever not have been printed.

Generators:-

These are useful for iterator.

Let us know about list comprehension.

List Comprehension means it is a special way of writing list

Ex: `list1 = [i for i range(10)]`
`print(list1)`

o/p: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Generators -

Ex: `tuple1 = (i for i in range(10))`
`print(tuple1)`

o/p: <generator object <genexpr> at 0x0000017A2
`tuple1 = (i for i in range(10000))`
o/p: `print(tuple1)`

o/p:- <generator object <genexpr> at 0x00001C7A2L>

print(next(tuple1))

o/p:- 0

for i in tuple1 :-

    print(next(tuple1))

o/p:- try it your self.

Ex:- def my_gen():

    n=1

    print("first")

    yield n

    n+=1

    print("Second")

    yield n

    n+=1

    print("third")

    yield n

a=my_gen()

print(a)

o/p:- <generator object my_gen at 0x00007SDA432>

if inplace of print(a) we write code as follows then o/p will be

next(a)

next(a)

next(a)

o/p:- first

Second

third