

ITERATORS

Looping

Important functions to be implemented

`__iter__()`

`__next__()`

`__iter__()` : takes iterable object like list, tuples

`__next__()`: is used to return the next value in iteration

Use of iterators

```
1
2 lst = [1,"Sourabh",5,3.0]
3 itr = iter(lst)
4 # iterate through it using next()
5 print(next(itr))
6 print(next(itr))
7 print(itr.__next__())
8 print(itr.__next__())
```

```
1
Sourabh
5
3.0
```

Iterators with class

```
1 class OddNumber:
2     """Class to implement an iterator
3     of odd numbers upto certain number"""
4
5     def __init__(self, max=0):
6         self.max = max
7
8     def __iter__(self):
9         self.num = 1
10        return self
11
12    def __next__(self):
13        if self.num <= self.max:
14            result = self.num
15            self.num += 2
16            return result
17        else:
18            raise StopIteration
19
20 on = OddNumber(10)
21 i = iter(on)
22 print(next(i))
23 print(next(i))
24 print(next(i))
25 print(next(i))
26 print(next(i))
```

```
1
3
5
7
9
```

GENERATORS

1. Generator generate one element at a time from a sequence.

2. Yield is used to get the value

3 It saves the state not like function where once function is called state will be returned to new call

USE

```
1 #generators
2 # A simple generator function
3 def get_me_no():
4     n = 1
5     yield n
6
7     n += 1
8     yield n
9
10    n += 1
11    yield n
12 My_numbers=get_me_no()
13
14 print(next(My_numbers))
15 print(next(My_numbers))
16 print(next(My_numbers))
```

```
1
2
3
```

DECORATORS

1. A decorator is a special function which adds some extra functionality to an existing function

2. A decorator is a function that accepts a function as a parameter and returns a function.

3. Decorators are useful to perform some additional processing required by a function.

Want to add addition functionality to function

```
1 def decor(func):
2     def inner_function(m,f,y):
3         if "Tesla" in m:
4             print("WOW!! Its Tesla Electric Vehicle")
5             return func(m,f,y)
6         return inner_function
7
8 def showvehicle(model,fueltype,year):
9     print("Broom...its ",model,fueltype,"manufactured in",year)
10
11 showvehicle = decor(showvehicle)
12 print(showvehicle("BMW 2 Series ","Petrol","2020"))
13 print(showvehicle("Tesla Model 3","EV","2022"))
```

```
Broom...its  BMW 2 Series  Petrol manufactured in 2020
None
WOW!! Its Tesla Electric Vehicle
Broom...its  Tesla Model 3 EV manufactured in 2022
..
```

List Comprehension vs Generation

Comprehension: all in one go

Generation : one by one ...fast

EXAMPLE

```
2 lst= [1, 4, 6, 8]
3 # square each term using list comprehension
4 square_list = [x**2 for x in lst]
5 # same thing can be done using a generator expression
6 # generator expressions are surrounded by parenthesis ()
7 generator = (x**2 for x in lst)
8 print(square_list)
9 print(generator)
10
```

```
[1, 16, 36, 64]
<generator object <genexpr> at 0x7f66c39c39d0>
```

```
1 print(next(generator))
```

```
1
```

```
1 print(next(generator))
```

```
16
```

@decor

showvehicle = decor(showvehicle) instead of this line, you can use @decor

@decor implementation

```
1 def decor(func):
2     def inner_function(m,f,y):
3         if "Tesla" in m:
4             print("WOW!! Its Tesla Electric Vehicle")
5             return func(m,f,y)
6         return inner_function
7     @decor
8     def showvehicle(model,fueltype,year):
9         print("Broom...its ",model,fueltype,"manufactured in",year)
10
11 print(showvehicle("BMW 2 Series ","Petrol","2020"))
12 print(showvehicle("Tesla Model 3","EV","2022"))
```

```
Broom...its  BMW 2 Series  Petrol manufactured in 2020
None
WOW!! Its Tesla Electric Vehicle
Broom...its  Tesla Model 3 EV manufactured in 2022
```



By sahusourabh

Published 22nd July, 2022.

Last updated 22nd July, 2022.

Page 1 of 1.

Sponsored by **Readable.com**

Measure your website readability!

<https://readable.com>