In [2]:
```
# Iterators

def add(a,b):
    z = a + b
    return z

add(10,20)
```

Out[2]: 30

In [15]:
```
my_list = [10,20,30]

my_iter = iter(my_list) # converting
```

In [24]:
```
iter(my_list)
```

Out[24]: <list_iterator at 0x2cda6d458a0>

In [4]:
```
for i in iter(my_list):
    print(i)
```

```
10
20
30
```

In [18]:
```
print(next(my_iter))
```

```
30
```

In [13]:
```
print(next(my_iter))
```

```
20
```

In [14]:
```
print(next(my_iter))
```

```
30
```

In [22]:
```
lst = [23,234,11]

lst.append(4)
lst.append(5)
print(lst)
```

```
[23, 234, 11, 4, 5]
```

In [23]:
```
# generator

# generators are a simpler way to create iterator using the functions with "Yiel

def counter(n):
    i = 0
    while i <=n:
        yield i
        i+=1
counter(10)
```

Out[23]: <generator object counter at 0x000002CDA6A5A380>

In [25]:
```
my_generator = counter(10)
```

In [31]:
```
next(my_generator)
```

Out[31]: 5

In [32]:
```
for num in my_generator:
    print(num)
```

```
6
7
8
9
10
```

In [ ]:
```
# Decorator

definition1 - a function call function is called the decorator

Or

a function call inside the function is called the decorator

Or

Decorator are functions that modify or extend the behaviour of other functions w
```

In [ ]:
```
# Baisc
```

In [33]:
```
def hello_world():

    def hello():
        print("Hi I am Hello function")

    return hello
```

In [35]:
```
h = hello_world()
```

In [36]:
```
h()
```

```
Hi I am Hello function
```

In [37]:
```
# 2

def outer(x):

    def inner(y):
        return x + y
    return inner

add = outer(10)
add(20)
```

Out[37]: 30

In [38]:
```
# @

def make(func):
```

```
    def inner():
        print("I am the decorator function")
        func()
    return inner
```

In [47]:
```
@make
def outer():
    print("I am the outer function")
```

In [48]:
```
outer()
```

```
I am the decorator function
I am the outer function
```

In [56]:
```
@make
def add():
    print(eval("10+20"))
```

In [57]:
```
add()
```

```
I am the decorator function
30
```

In [58]:
```
#
def s_divide(func):

    def inner(a,b):
        if b == 0:
            print("You can not divide anything by zero")
            return
        return func(a,b)
    return inner
```

In [60]:
```
@s_divide
def divide(a,b):
    print(a/b)

divide(2,5)
divide(2,0)
```

```
0.4
You can not divide anything by zero
```

In [ ]: