A decorator in Python is any callable Python object that is used to modify a function or a class. A reference to a function "func" or a class "C" is passed to a decorator and the decorator returns a modified function or class. In Other terms, a decorator is something which takes a function/class as input, return function/class as output after modifications.

In [16]:
```
def normal_function():
    print("This is a normal function which prints something !!")
normal_function()
```

```
This is a normal function which prints something !!
```

Here i am passing a normal function to a decorator, decorator is adding a Before execution and After execution steps to that normal function. i.e If some steps that are mandatory to be executed before execution and after execution of the actual function, we can put those steps in that decorator and then we can use it whenever we want.

In [8]:
```
def myself_decorator(func):
    def inner1():
        print("Hello, this is before function execution")
        func()
        print("This is after function execution")
    return inner1


def normal_function():
    print("This is a normal function which prints something !!")
normal_function = myself_decorator(normal_function)

normal_function()
```

```
Hello, this is before function execution
This is a normal function which prints something !!
This is after function execution
```

Common Syntax of using decorators, using a @decorator_name, at the time of function defination.

In [9]:
```
@myself_decorator
def normal_function():
    print("This is a normal function which prints something !!")
normal_function()
```

```
Hello, this is before function execution
This is a normal function which prints something !!
This is after function execution
```

Another example of a decorator, which calculates the total run time of a given function.

In [10]:
```
import time
import math

def calculate_time(func):
    def inner1(*args, **kwargs):
        begin = time.time()
        func(*args, **kwargs)
        end = time.time()
        print("Total time taken in : ", func.__name__, end - begin)
    return inner1
```

Output of "factorial" function when decorator is not used.

In [12]:
```
def factorial(num):
    time.sleep(2)
    print(math.factorial(num))
factorial(10)
```

```
3628800
```

Output of "factorial" function when is used.

In [15]:
```
@calculate_time
def factorial(num):
    time.sleep(2)
    print(math.factorial(num))
factorial(10)
```

```
3628800
Total time taken in :  factorial 2.0022976398468018
```