27.06.2022 **Yusuf Ünlü**

Decorators in Python

A decorator in Python is a function that takes another function as its argument and returns yet another function. Decorators can be extremely useful as they allow the extension of an existing function, without any modification to the original function source code.

Before explaining decorators in detail, let's see what is the **first class objects** and **nested functions,** because this will help us to understand decorators better.

First Class Objects: These are entities within a programming language like functions and they can do several things;

- ✓ Can appear in an expression
- ✓ Can be assigned to a variable
- ✓ Can be used as an argument
- ✓ Can be returned by a function call

Nested Functions: Nested functions are functions that you define inside other functions. In Python, this kind of function has direct access to variables and names defined in the enclosing function. Inner functions have many uses, most notably as decorator functions and closure factories. **Example:**

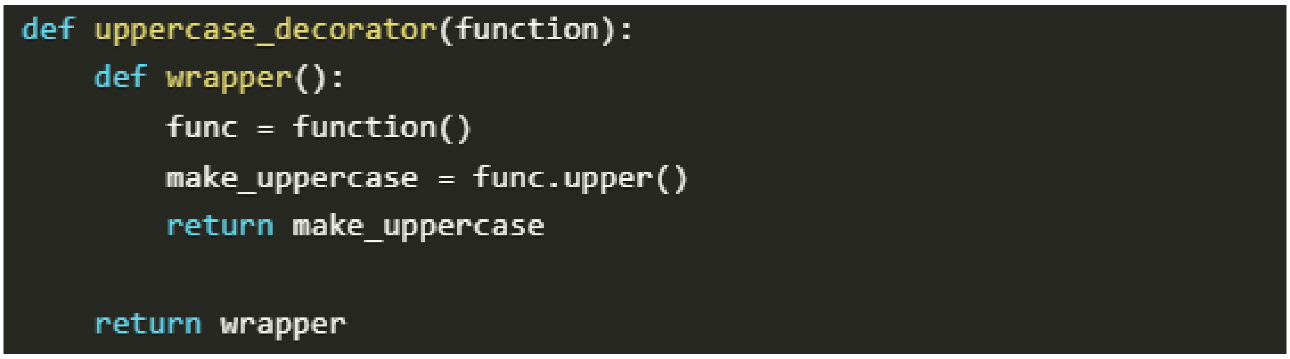
```
>>> def outer_func():
...     def inner_func():
...         print("Hello, World!")
...     inner_func()
>>> outer_func()
Hello, World!
```

In this code, you define inner_func() inside outer_func() to print the Hello, World! message to the screen. To do that, you call inner_func() on the last line of outer_func().

ⓘ Note that inner functions **cannot be accessible** globally. Because they are being created by outer functions and destroyed after that process is completed.

So, now we know what is a first class object and nested functions. Let's continue with how to use a decorator:

To use a decorator function, you need to write "**@**" character and function name of the decorator **above** the function that you want to decorate. Let's see an **example:**



```
def uppercase_decorator(function):
    def wrapper():
        func = function()
        make_uppercase = func.upper()
        return make_uppercase

    return wrapper
```

This is a decorator function that will make every lower case letter to upper case in the function it's used. We write this function before the function/s it will be used.

```python
@uppercase_decorator
def say_hi():
    return 'hello there'

say_hi()
```

As you can see we used the decorator function as **@uppercase_decorator.** In **say_hi** function we returned **‘hello there’** which its all letters are lower case. At first the decorator function will be run and make all letters uppercase. So, our output will be this:

```
'HELLO THERE'
```

Conclusion

If you define a function inside another function, then you’re creating an **inner function**, also known as a nested function. In Python, inner functions have direct access to the variables and names that you define in the enclosing function. This provides a mechanism for you to create helper functions, closures, and decorators.

Source:

https://medium.com/herkes-i%C3%A7in-python/python-decorators-45aac5308fb#

https://towardsdatascience.com/how-to-use-decorators-in-python-by-example-b398328163b

https://www.datacamp.com/tutorial/decorators-python

https://realpython.com/inner-functions-what-are-they-good-for/#:~:text=Inner%20functions%2C%20also%20known%20as,closure%20factories%20and%20decorator%20functions.