Decorators

https://www.datacamp.com/community/tutorials/decorators-python

What are decorators in Python?

Decorators is used to add functionality to an exicting code.

A decorator is a design pattern in Python that allows a user to add new functionality to an existing object without modifying its structure.

Decorators are usually called before the definition of a function you want to decorate

Functions in Python are first class citizens. This means that they support operations such as being passed as an argument, returned from a function, modified, and assigned to a variable. This is a fundamental concept to understand before we delve into creating Python decorators.

Our decorator function takes a function as an argument, and we shall, therefore, define a function and pass it to our decorator. We learned earlier that we could assign a function to a variable. We'll use that trick to call our decorator function.

Syntax:

decorate = uppercase_decorator(function_name) #It is decorator function and accept the other function as argument

decorate()

However, Python provides a much easier way for us to apply decorators. We simply use the @ symbol before the function we'd like to decorate.

Syntax:

@Decorator function name

def function_name(arguments...) #before it add the decorator

- ✔ **Applying Multiple Decorators to a Single Function**

We can use multiple decorators to a single function. However, the decorators will be applied in the order that we've called them

Syntax:

@Decorator1 function name

@Decorator2 function name

def function_name(arguments...) #before it add the decorator

- ✔ **Accepting Arguments in Decorator Functions**

Sometimes we might need to define a decorator that accepts arguments. We achieve this by passing the arguments to the wrapper function. The arguments will then be passed to the function that is being decorated at call time.

Syntax:

@Decorator1 function name with arguments

def function_name(arguments...) #before it add the decorator

✔ Defining General Purpose Decorator

To define a general purpose decorator that can be applied to any function we use args and **kwargs. args and **kwargs collect all positional and keyword arguments and stores them in the args and kwargs variables. args and kwargs allow us to pass as many arguments as we would like during function calls.

✔ Python Decorators Summary

Decorators dynamically alter the functionality of a function, method, or class without having to directly use subclasses or change the source code of the function being decorated. Using decorators in Python also ensures that your code is DRY(Don't Repeat Yourself). Decorators have several use cases such as:

•Authorization in Python frameworks such as Flask and Django
•Logging
•Measuring execution time
•Synchronization