Python Decorator with Arguments

If this Python Tutorial saves you hours of work, please whitelist it in your ad blocker 😭 and

Donate Now

(https://www.pythontutorial.net/donation/)

to help us ❤️ pay for the web hosting fee and CDN to keep the website running.

Summary: in this tutorial, you'll learn how to define Python decorators with arguments using a decorator factory.

Introduction to Python decorator with arguments

Suppose that you have a function (https://www.pythontutorial.net/python-basics/python-functions/) called `say` that prints out a message:

```
def say(message):
    ''' print the message
    Arguments
        message: the message to show
    '''
    print(message)
```

and you want to execute the `say()` function 5 times repeatedly each time you call it. For example:

```
say('Hi')
```

It should show the following the `Hi` message five times as follows:

```
Hi
Hi
Hi
Hi
Hi
```

To do that, you can use a regular decorator (https://www.pythontutorial.net/advanced-python/python-decorators/) :

```
@repeat
def say(message):
    ''' print the message
    Arguments
        message: the message to show
    '''
    print(message)
```

And you can define the `repeat` decorator as follows:

```
def repeat(fn):
    @wraps(fn)
    def wrapper(*args, **kwargs):
        for _ in range(5):
            result = fn(*args, **kwargs)
        return result

    return wrapper
```

The following shows the complete code:

```
from functools import wraps
```

```
def repeat(fn):
    @wraps(fn)
    def wrapper(*args, **kwargs):
        for _ in range(5):
            result = fn(*args, **kwargs)
        return result

    return wrapper


@repeat
def say(message):
    ''' print the message
    Arguments
        message: the message to show
    '''
    print(message)


say('Hello')
```

What if you want to execute the `say()` function repeatedly ten times. In this case, you need to change the hard-coded value 5 in the `repeat` decorator.

However, this solution isn't flexible. For example, you want to use the `repeat` decorator to execute a function 5 times and another 10 times. The `repeat` decorator would not meet the requirement.

To fix this, you need to change the `repeat` decorator so that it accepts an argument that specifies the number of times a function should execute like this:

```
@repeat(5)
def say(message):
    ...
```

To define the `repeat` decorator, the `repeat(5)` should return the original decorator.

```
def repeat(times):
    # return the original "repeat" decorator
```

The new `repeat` function returns a decorator. And it's often referred to as a decorator factory.

The following `repeat` function returns a decorator:

```
def repeat(times):
    ''' call a function a number of times '''
    def decorate(fn):
        @wraps(fn)
        def wrapper(*args, **kwargs):
            for _ in range(times):
                result = fn(*args, **kwargs)
            return result
        return wrapper
    return decorate
```

In this code, the `decorate` function is a decorator. It's equivalent to the original `repeat` decorator.

Note that the new repeat function isn't a decorator. It's a decorator factory that returns a decorator.

Put it all together.

```
from functools import wraps


def repeat(times):
    ''' call a function a number of times '''
    def decorate(fn):
        @wraps(fn)
        def wrapper(*args, **kwargs):
```

```
            for _ in range(times):
                result = fn(*args, **kwargs)
            return result
        return wrapper
    return decorate


@repeat(10)
def say(message):
    ''' print the message
    Arguments
        message: the message to show
    '''
    print(message)


say('Hello')
```

Summary

- Use a factory decorator to return a decorator that accepts arguments.