Published in Towards Data Science

Prudhvi Vajja Follow

Nov 11, 2020 · 6 min read · Listen

@Decorators in Python (Advanced)

Level up your Python programming (**Bonus :** `*args` `**kwargs`)..... 🔥

Photo by [illegible] on Unsplash

Brace yourselves It does contain lot of code.✌

I don't need to explain why these are difficult to understand, It is the reason you are here **(Don't expect theory only code).** But do you really need to write decorators though?

> *The joy of coding Python should be in seeing short, concise, readable classes that express a lot of action in a small amount of clear code — not in reams of trivial code that bores the reader to death.* ***— Gudio van Rossum. (creator of python)***

Table of contents: (clickable)

Bonus: Learn how to use `*args` & `**kwargs`

> ***Decorators are nothing but functions that add new functionalities to the existing methods without changing them.***

To understand Decorators, you need to understand functions in Python, functions are first-class objects i.e we can assign multiple variables to the same function or we can even send them as arguments to other functions.

Let's see some cool things we can do with python functions.

1. Assign a function to multiple variables.

```
>>> def func(x):
...     return x.upper()
>>> func("roar")
'ROAR'

>>> new_func = func  # Assign func to a variable.

>>> new_func("meow")
'MEOW'

>>> del func
# We can call the new_func even after deleting the func
>>> new_func("meow "*2)
'MEOW MEOW '
```

2. Create functions inside a function. (New for `C/C++` programmers.)

```
def factorial(n):
    """
    Calculates the factorial of n,
    n => integer and n >= 0.
    """
    if type(n) == int and n >= 0:
        if n == 0:
            return 1
        else:
            return n * factorial(n-1) # Recursive Call
    else:
        raise TypeError("n should be an integer and n >= 0")
```

what is the defect in the above code?

`factorial(10)` checks type of 10,9,8,7,...recursively which is not necessary. we can solve this elegantly by using inner functions.

```
def factorial(n):
    """
    Calculates the factorial of n,
    n => integer and n >= 0.
    """
    def inner_factorial(n):
        if n == 0:
            return 1
        else:
            return n * inner_factorial(n-1)
    if type(n) == int and n >=0:
        return inner_factorial(n)
    else:
        raise TypeError("n should be an integer and n >= 0")
```

Hey, you can use a decorator here. right let's wait though.

3. Pass a function as a parameter to other functions.

```
import math
def sin_cos(func, var):
    print("Call this" + func.__name__ +"function")
    print(func(var))

sin_cos(math.sin, 60) # -0.3048106211022167
sin_cos(math.cos, 45) # 0.5253219888177297
```

4. Functions can return other functions.

```
def sound(range):
    """
    Args: range (Type of sound). (<class 'str'>)
    Return: function object of the sound (<class 'function'>)
    """
    def loud(x):
        print(x.upper() + '🐯')
    def low(x):
```

```
        print(x.lower() + '🐱')
    if range == 'loud':
        return loud
    else:
        return low

tiger = sound("loud") # you can use this as a functions.
tiger("roar..") # ROAR..🐯

cat = sound("low")
cat("MEOW..") # meow..🐱
```

I was a bit confused too when I saw this for the first time. For each range, `sound` return its respective function. A more useful example is to create an n-degree polynomial.

Ref: https://www.python-course.eu/python3_decorators.php#Functions-returning-Functions

```
def polynomial_creator(a, b, c):
    """
    Creates 2nd degree polynomial functions
    """
    def polynomial(x):
        return a * x**2 + b * x + c
    return polynomial

p1 = polynomial_creator(2, 3, -1)
p2 = polynomial_creator(-1, 2, 1)

x = -2
print(x, p1(x), p2(x)) # -2 1 -7
```

Short Exercise: Try to write a function that takes degree and parameter as arguments and returns a n-degree polynomial.

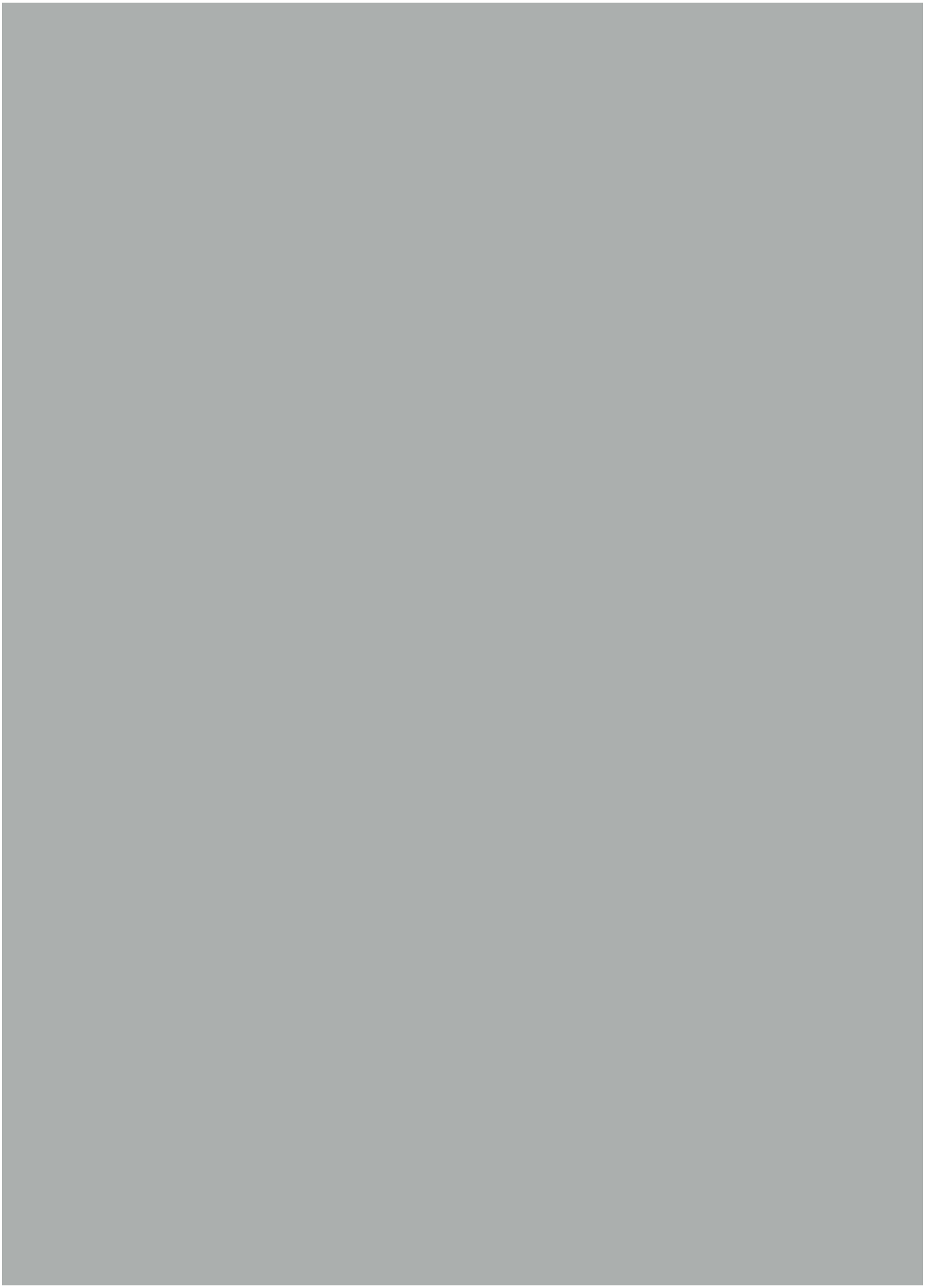

Photo by Charles Deluvio on Unsplash

Done with the Basics, It's gonna be tricky from here. (👀)

Let's start with a simple decorator that print's the name of the function before it runs.

🦋 Try it out with your own examples 🦋

🔥 **Pro Tip:** By using `*args` you can send varying lengths of variables to a function. `*args` accepts it as a tuple. know more about `*args` and `*kwargs`

The syntax of decorators is different from what we have expressed in the above example. It is usually denoted with `@` (*To Beautify your code*).

🦋 Try it out with your own examples 🦋

🔥 **ProTip:** By using Code-Medium you can ***Create*** & ***Edit*** GitHub Gists directly from your medium article. (Like the one above).

> Note: When you wrap a function using decorator the attributes of original function such as __doc__ (docstring), __name__(name of the function), __module__(module in which the function is defined) will be lost.

Although you can overwrite them in the decorator function python has a built-in decorator @wraps to do it.

If you look closely I'm printing the outputs of each function, If you want to return an output. you need to add an extra return inside the wrapper function.

Usage of return while using decorators.

Photo by Dan on Unsplash

To Understand better let's look at some Real-world examples of Decorators.

1. **Timer:** *Track time taken by a function to run.*

🤣 To Understand *args and *kwargs copy the code and run it by passing different variables 🤣

🔥 **Pro Tip:** You can access the original function by unwrapped it using `original_looper = looper.__wrapped__` (*Assuming you have used* `@wraps`)

2. **Deco:** *A library that automatically parallelizes python programs* ***Link*****.**

Alex Sherman created this simple library called deco which allows python functions to run in parallel just by adding a decorator *pretty cool Huh..!.*

The function that you want run in parallel is decorated with @concurrent
The function that calls the parallel function is decorated with @synchronized

```
@concurrent # We add this for the concurrent function
def process_lat_lon(lat, lon, data):
  #Does some work which takes a while
  return result

# And we add this for the function which calls the concurrent function
@synchronized
def process_data_set(data):
  results = defaultdict(dict)
  for lat in range(...):
    for lon in range(...):
      results[lat][lon] = process_lat_lon(lat, lon, data)
  return results
```

Pro Tip 🔥: A similar but advanced Scientific Computing library to run python code faster is **numba.**

3. **Showme:** *ShowMe is a simple set of extremely useful function decorators for Python. It allows you to view trace information, execution time, cputime, and function documentation.* ***LINK*****.**

Print passed-in arguments and function calls.

```
ref: https://github.com/navdeep-G/showme

@showme.trace
def complex_function(a, b, c, **kwargs):
```

```
    ...


>>> complex_function('alpha', 'beta', False, debug=True)
calling haystack.submodule.complex_function with
   args: ({'a': 'alpha', 'b': 'beta', 'c': False},)
   kwargs: {'debug': True}
```

Print function execution time.

```
@showme.time
def some_function(a):
    ...

>>> some_function()
Execution speed of __main__.some_function:
0.000688076019287 seconds
```

Print function cpu-execution time.

```
@showme.cputime
 def complex_function(a, b, c):
     ...

 >>> complex_function()
 CPU time for __main__.complex_function:
      3 function calls in 0.013 CPU seconds

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1    0.013    0.013    0.013    0.013 test_time.py:6(test)
     1    0.000    0.000    0.000    0.000 {method 'disable' of
'_lsprof.Profiler' objects}
     1    0.000    0.000    0.000    0.000 {range}
```

Links to more examples are in the references section.

By now you got an idea of how decorators work and create your own decorator. let's see some built-in Python decorators.

Photo by Leen on Unsplash

Popular Built-In Python Decorators

`@classmethod:` *Creates an instance of the class. (Pass* `cls` *not* `self` *)*

Example: Suppose you are getting data from different data types instead of creating multiple classes we can handle it using @classmethod.

Usage of @classmethod in Python.

`@staticmethod:` *You can use the methods with this decorator without creating a class instance. (No need to pass* `self` *)*

Example: Suppose you want to create some helper functions for a class that contains some logic but it is a not property of the class.

`@Property` : A way to use `get` & `set` commands in OOPs.

Example: Suppose there are some constraints to assign a variable in a class we can use @property to deal with this kind of situation instead of creating a chain rule of every change we make.

Every time you call `set` or `get` to the `maxlimit` value of the box it goes through the `@property` and `@maxlimit.setter`.

References:

- *Advanced Python by Brend Klein.*

- *If you like python you will fall in love with Real Python.*
- *How does @property work in detail.*
- *Awesome Python Decorators.*

> I know this a long one, But if you reached here 👏 to you. Keep learning Keep Growing.

connect with me on LinkedIn or Github.

Python Object Oriented Decorators Advanced Programming