Python Decorators Introduction

Learn **Python Decorators** in this tutorial.

Add functionality to an existing function with decorators. This is called metaprogramming.

A function can take a function as argument (*the function to be decorated*) and return the same function with or without extension.

Extending functionality is very useful at times, we'll show real world examples later in this article.

Related course: Complete Python Programming Course & Exercises

Functions are objects

In Python everything is an object, including functions. This means functions can be passed around and returned. When you see it, it may look odd at first:

```
def hello():
    print("Hello")

# even functions are objects
message = hello

# call new function
message()
```

Call the methods either message() or hello() and they have the same output. That's because they refer to the same object.

Now let's go on with decorators.

Decorators

Example

A decorator takes a function, extends it and returns. Yes, **a function can return a function**.

```
def hello(func):
    def inner():
        print("Hello ")
        func()
    return inner

def name():
    print("Alice")


obj = hello(name)
obj()
```

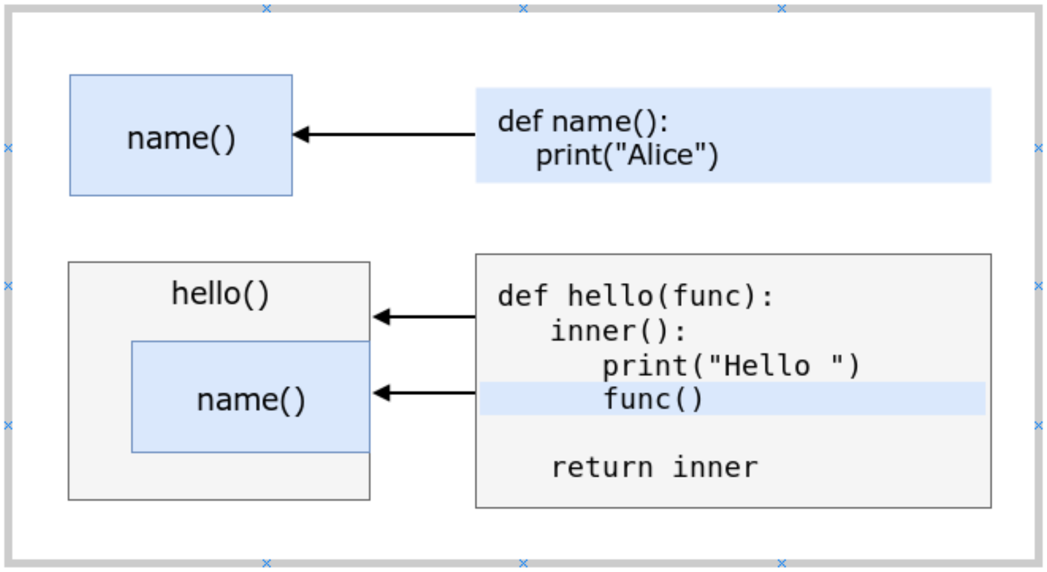
In the above example, hello() is a decorator.

In the statement

```
obj = hello(name)
```

the function name() is decorated by the function hello().

It wraps the function in the other function.



Example 2

Functions can be extended by wrapping them.

```
def who():
    print("Alice")

def display(func):
    def inner():
        print("The current user is : ", end="")
        func()
    return inner

if __name__ == "__main__":
    myobj = display(who)
    myobj()
```

The function who() gets decorated by display().

Syntactic sugar

Decorators are common and can be simplified. While it does exactly the same, its just cleaner code.

Python can simplify the use of decorators with the **@ symbol**.

```
@hello
def name():
    print("Alice")

if __name__ == "__main__":
    name()
```

This will output exactly the same, but is a cleaner way to write the code.

Stay with me. The call

```
@hello
def name():
```

is just a simpler way of writing:

```
obj = hello(name)
```

In both cases we apply the decorator to a function.

Arguments

Parameters can be used with decorators. If you have a funtion that prints the sum a + b, like this

```
def sumab(a,b):
    summed = a + b
    print(summed)
```

You can wrap it in a decorator function.
The example below shows how to do that:

```
def pretty_sumab(func):
    def inner(a,b):
        print(str(a) + " + " + str(b) + " is ", end="")
        return func(a,b)

    return inner

@pretty_sumab
def sumab(a,b):
    summed = a + b
    print(summed)

if __name__ == "__main__":
    sumab(5,3)
```

The function sumab is wrapped by the function pretty_sumab. This is indicated with the @ symbol above it.

Call the function sumab, and see that both the logic of the functions sumab and pretty_sumab are run, with parameters.

Real world examples

Use Case: Time measurement

Execution time

Decorators can be used to measure function run time.

A decorator can be used to measure how long a function takes to execute.

If you define a simple function that sleeps,

```
def myFunction(n):
  time.sleep(n)
```

You can then measure how long it takes simply by adding the line @measure_time

An example below:

```
import time

def measure_time(func):

  def wrapper(*arg):
      t = time.time()
      res = func(*arg)
      print("Function took " + str(time.time()-t) + " seconds to run")
      return res

  return wrapper

@measure_time
def myFunction(n):
  time.sleep(n)

if __name__ == "__main__":
    myFunction(2)
```

This will output the time it took to execute the function myFunction(). The cool thing is by adding one line of code *@measure_time* we can now measure program execution time.

Web apps

Flask @app.route("/") is a decorator

Use Case: Web app
Lets take the use case of web apps. When you build a web app in Flask, you always write url routes.

Every route is a certain page in the web app.
Opening the page /about may call the about_page() method.

```
@app.route("/about")
def about_page():
  return "Website about nachos"
```

In this case it uses the @ symbol for decoration.

Download examples and exercises

Back

Reading JSON from a file

Next

Create a Python Web Server