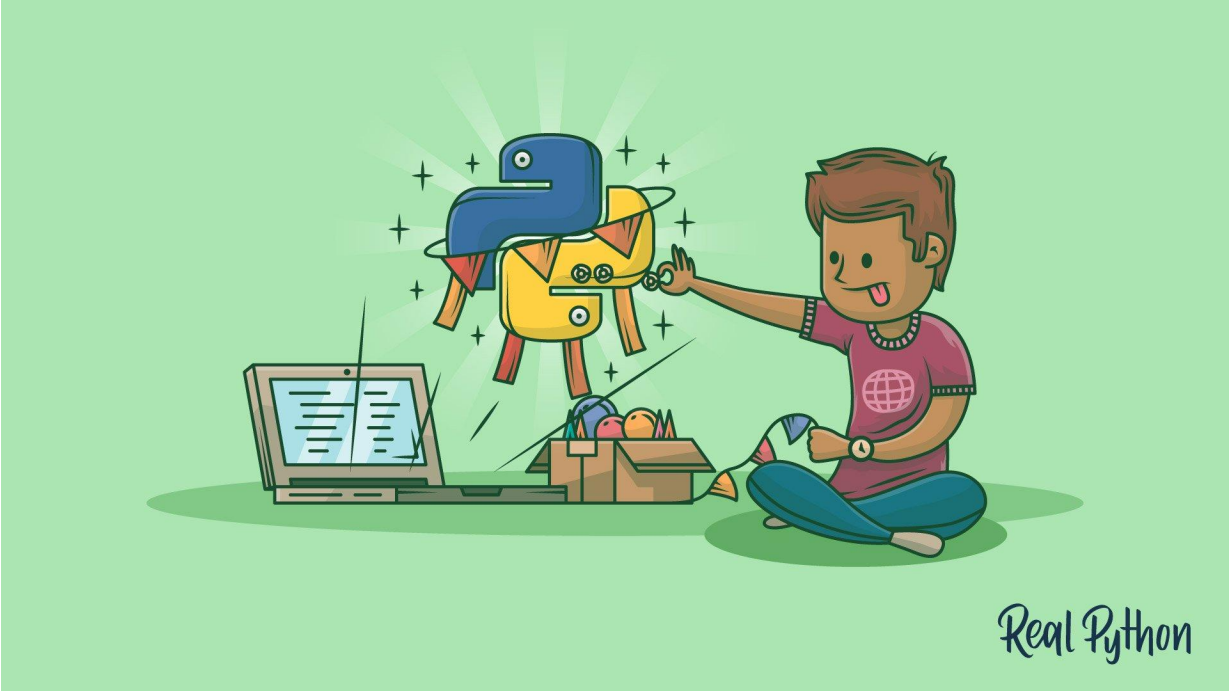
Python Decorators 101



Decorators

What is a decorator?

- A function that takes another function
- Extends the behavior of that function
- Without explicitly modifying the function

TABLE OF CONTENTS

TABLE OF CONTENTS

FUNCTIONS

- Header
- Statement
- Calling the Function

```
def my_function():
    """docstring"""
    statement


my_function()
```

FUNCTIONS

A function returns a value based on the given arguments

```
def my_function(argument):
    """docstring"""
    my_var = argument * 5
    return my_var
```

TABLE OF CONTENTS

FUNCTIONS

Functions are first-class objects

- Functions can be passed around and used as arguments

```
def say_hello(name):
    return f"Hello {name}"

def be_awesome(name):
    return f"Yo {name}, together we are the awesomest!"

def greet_bob(greeter_func):
    return greeter_func("Bob")
```

```
>>> greet_bob(say_hello)
'Hello Bob'

>>> greet_bob(be_awesome)
'Yo Bob, together we are the awesomest!'
```

TABLE OF CONTENTS

FUNCTIONS

Inner functions

- Functions can be defined inside other functions - called **inner** functions

```
def parent():
    print("Printing from the parent() function")

    def first_child():
        print("Printing from the first_child() function")

    def second_child():
        print("Printing from the second_child() function")

    second_child()
    first_child()
```

TABLE OF CONTENTS

FUNCTIONS

Returning functions from functions

- Python also allows you to use functions as return values

```
def parent(num):
    def first_child():
        return "Hi, I am Emma"

    def second_child():
        return "Call me Liam"

    if num == 1:
        return first_child
    else:
        return second_child
```

TABLE OF CONTENTS

TABLE OF CONTENTS

DECORATORS

Simple decorators

- Example decorator

```
def my_decorator(func):
    def wrapper():
        print("Something is happening before the function is called.")
        func()
        print("Something is happening after the function is called.")
    return wrapper

def say_whee():
    print("Whee!")

say_whee = my_decorator(say_whee)
```

DECORATORS

Simple decorators

- What happens when you call `say_whee()`?

```
>>> say_whee()
Something is happening before the function is called.
Whee!
Something is happening after the function is called.
```

- The so called decoration happens at the following line

```
say_whee = my_decorator(say_whee)
```

TABLE OF CONTENTS

DECORATORS

Syntactic Sugar!

- Python allows you to use decorators in a simpler way with the `@` symbol

```
def my_decorator(func):
    def wrapper():
        print("Something is happening before the function is called.")
        func()
        print("Something is happening after the function is called.")
    return wrapper

@my_decorator
def say_whee():
    print("Whee!")
```

TABLE OF CONTENTS

DECORATORS

Reusing Decorators

- Create a file called `decorators.py` with the following

```
def do_twice(func):
    def wrapper_do_twice():
        func()
        func()
    return wrapper_do_twice
```

- You can now use this decorator in other files by doing a regular import

```
from decorators import do_twice

@do_twice
def say_whee():
    print("Whee!")
```

TABLE OF CONTENTS

DECORATORS

Decorating Functions with Arguments

- Can you decorate a function that accepts some arguments?

```
from decorators import do_twice

@do_twice
def greet(name):
    print(f"Hello {name}")
```

- Running this code raises an error

```
>>> greet("World")
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: wrapper_do_twice() takes 0 positional arguments but 1 was given
```

DECORATORS

Decorating Functions with Arguments

- The solution is to use *args and **kwargs in the inner wrapper function

```
def do_twice(func):
    def wrapper_do_twice(*args, **kwargs):
        func(*args, **kwargs)
        func(*args, **kwargs)
    return wrapper_do_twice
```

```
>>> say_whee()
Whee!
Whee!

>>> greet("World")
Hello World
Hello World
```

TABLE OF CONTENTS

DECORATORS

Returning Values From Decorated Functions

- What happens to the return value of decorated functions?

```
from decorators import do_twice

@do_twice
def return_greeting(name):
    print("Creating greeting")
    return f"Hi {name}"
```

```
>>> hi_adam = return_greeting("Adam")
Creating greeting
Creating greeting
>>> print(hi_adam)
None
```

DECORATORS

Returning Values From Decorated Functions

- Change the wrapper function so it returns the value of the decorated function

```
def do_twice(func):
    def wrapper_do_twice(*args, **kwargs):
        func(*args, **kwargs)
        return func(*args, **kwargs)
    return wrapper_do_twice
```

```
>>> hi_adam = return_greeting("Adam")
Creating greeting
Creating greeting
'Hi Adam'
```

TABLE OF CONTENTS

DECORATORS

Who Are You, Really?

- Introspection is the ability of an object to know about its own attributes at runtime

```
import functools

def do_twice(func):
    @functools.wraps(func)
    def wrapper_do_twice(*args, **kwargs):
        func(*args, **kwargs)
        return func(*args, **kwargs)
    return wrapper_do_twice
```

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

CONGRATULATIONS!!!