Twitter Thread by Jan Giacomelli

Jan Giacomelli
@jangiacomelli

■ Python decorators

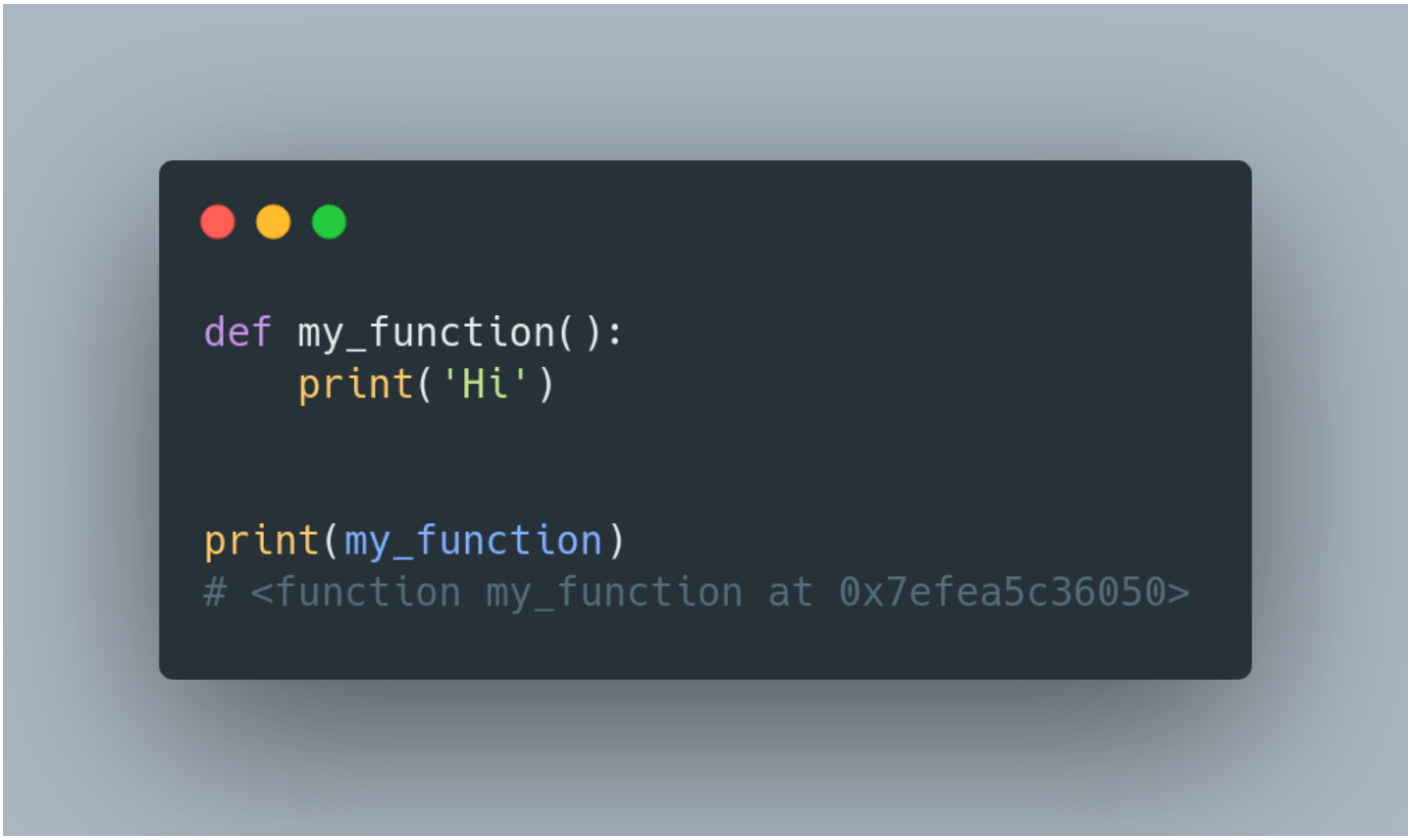
What are they? How do you use them?

■ Let's find out ■

1■■ Decorator is a function or a class that wraps another function or class modifying its behavior.

So how does that work?

The first thing to know is that everything in Python is an object - functions too



2■■ That means they can be passed to another function as an argument or returned from a function

Functions that take other functions as an argument are called higher-order functions



```python
def send_email_ses(email_message):
    print(f'Send email using AWS SES: {email_message}')


def send_email_sendgrid(email_message):
    print(f'Send email using SendGrid: {email_message}')


def send_email(send_email_function):
    email_message = 'Hi'
    send_email_function(email_message)


send_email(send_email_ses)
# Send email using AWS SES: Hi
send_email(send_email_sendgrid)
# Send email using SendGrid: Hi
```

3■■ In Python, you can define a function inside other function - such functions are called inner functions



```python
def get_send_email_function(email_service):
    def send_email_ses(email_message):
        print(f'Send email using AWS SES: {email_message}')

    def send_email_sendgrid(email_message):
        print(f'Send email using SendGrid: {email_message}')

    def send_email_dummy(email_message):
        print(f'I just pretend that I will send email: {email_message}')

    if email_service == 'SES':
        return send_email_ses
    elif email_service == 'SENDGRID':
        return send_email_sendgrid
    else:
        return send_email_dummy


send_email = get_send_email_function('SES')
send_email('Hi')
# Send email using AWS SES: Hi
```

4■■ To create a decorator you just need to apply all of that together

log_enter_leave is a decorator.

my_function is the function.

To alter my_function's behavior we reassign it applying log_enter_leave decorator.



```python
def log_enter_leave(func):
    def wrapper(*args, **kwargs):
        print('Enter function')
        result = func(*args, **kwargs)
        print('Leave function')
        return result

    return wrapper


def my_function():
    print('Hi')


my_function = log_enter_leave(my_function)
my_function()
# Enter function
# Hi
# Leave function
"""
*log_enter_leave* is a decorator - it accepts any function object as an argument.
We want to call decorated function *func* the same way as undecorated *func*.
To do that we define inner function *wrapper*.
It accepts any arguments and keyword arguments.
*wrapper* alters behavior of *func*.
It prints before and after *func* is executed.
*log_enter_leave* just returns *wrapper* function object to the caller.
Because functions are objects we can store them in variables.
So to alter *my_function* we reassign decorated *my_function* to *my_function* - my_function =
log_enter_leave(my_function).
When we do that *wrapper* function is returned which now looks like this:
```
    def wrapper(*args, **kwargs):
        print('Enter function')
        result = my_function(*args, **kwargs)
        print('Leave function')
        return result
...

So now when we call:
```
my_function()
```
We see this printed out:
Enter function
Hi
Leave function
"""
```

5■■ To simplify usage of decorators Python offers us syntactic sugar

A "pie-decorator" syntax using @

@decorator_name

6■■ The only problem here is that my_function now identify as the wrapper function

To solve that we just need to use wraps from functools
```

```python
def log_enter_leave(func):
    def wrapper(*args, **kwargs):
        print('Enter function')
        result = func(*args, **kwargs)
        print('Leave function')
        return result

    return wrapper


@log_enter_leave
def my_function():
    print('Hi')


print(my_function.__name__)
# wrapper

from functools import wraps


def log_enter_leave(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        print('Enter function')
        result = func(*args, **kwargs)
        print('Leave function')
        return result

    return wrapper


@log_enter_leave
def my_function():
    print('Hi')


print(my_function.__name__)
# my_function
```

7■■ You can decorate classes too.

For example, you can dataclass decorator on your class to automatically generate its __init__ and __repr__ methods



8■■ You can also use a class as a decorator

Decorator class needs methods:
- __init__
- __call__ (it makes class callable)



```python
import functools

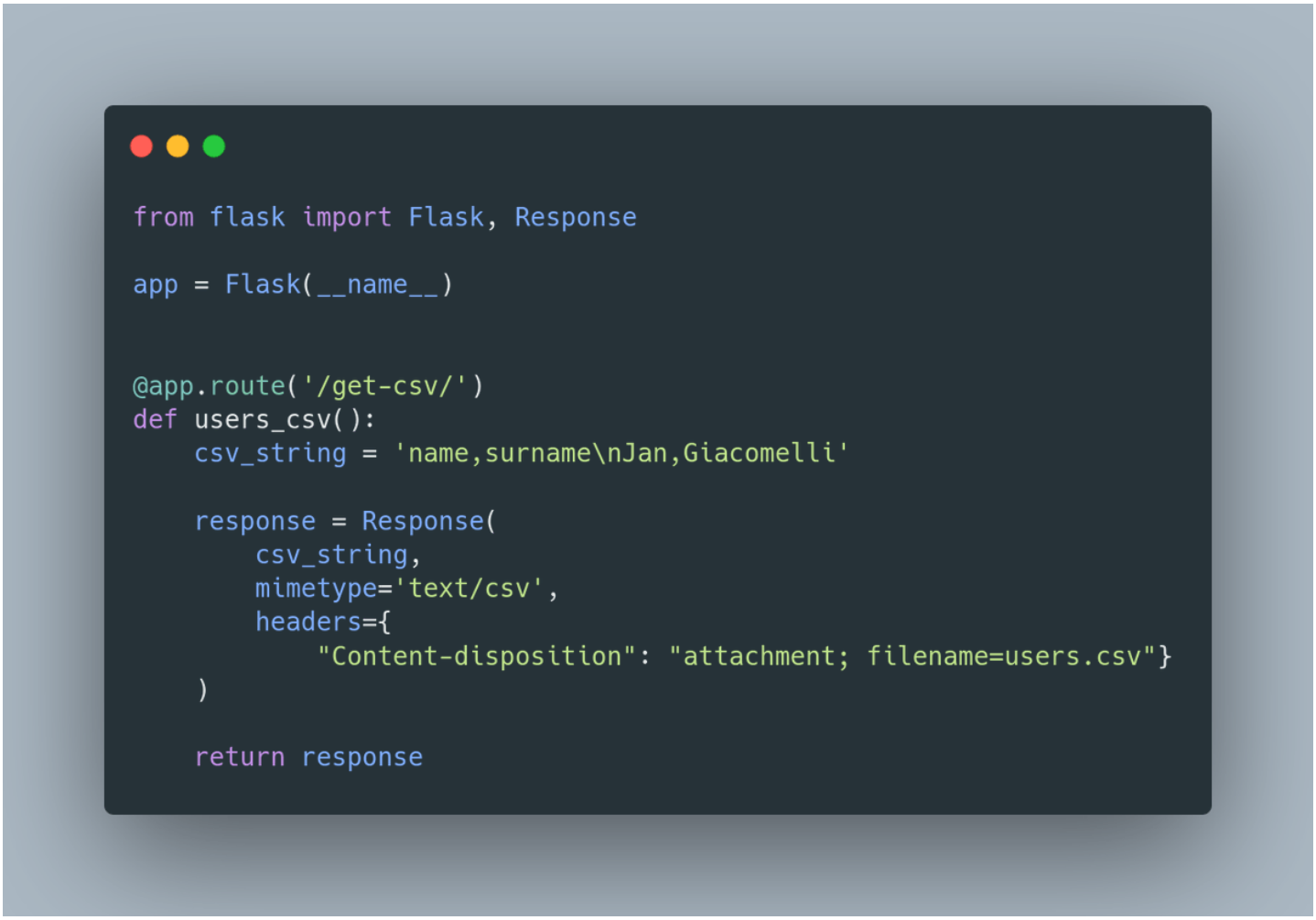

class CallCounter:
    def __init__(self, function):
        functools.update_wrapper(self, function)
        self.function = function
        self.number_calls = 0

    def __call__(self, *args, **kwargs):
        self.number_calls += 1
        print(f"{self.function.__name__} was called {self.number_calls} times.")
        return self.function(*args, **kwargs)


@CallCounter
def my_function():
    print("Hi!")


"""
What's going one here?
@CallCounter actually does this:
```
my_function = CallCounter(my_function)
```
So my function is now instance of CallCounter class.
It must be callable to be able to call it the same way as my_function.
When instance of CallCounter is called - __call__ is executed ->
number_calls is incremented and result from decorated function is returned.
Every time we call my_function the same instance of CallCounter is called.
That's why number_calls is bigger for every consecutive call.
"""
my_function()
# my_function was called 1 times.
# Hi!
my_function()
# my_function was called 2 times.
# Hi!
```

9■■ For example, decorators are used for registering view functions to the Flask application



```python
from flask import Flask, Response

app = Flask(__name__)


@app.route('/get-csv/')
def users_csv():
    csv_string = 'name,surname\nJan,Giacomelli'

    response = Response(
        csv_string,
        mimetype='text/csv',
        headers={
            "Content-disposition": "attachment; filename=users.csv"}
    )

    return response
```

1■■0■■ Read more:

https://t.co/eBMh1Gv0xZ

https://t.co/2YdA2ZIQoV

https://t.co/Wb0jSjmzlc

1■■1■■ Script with all of the examples:

https://t.co/Tw1qrbN0nN