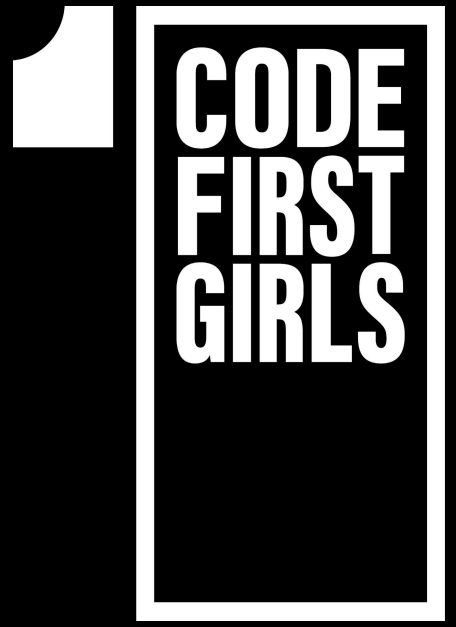
DECORATORS

LESSON 3



NANODEGREE → ENGINEERING MODULE

AGENDA

01 Decorator functions

02 Decorator syntax and implementation

03 Class as decorator

04 Practice

PYTHON DECORATOR

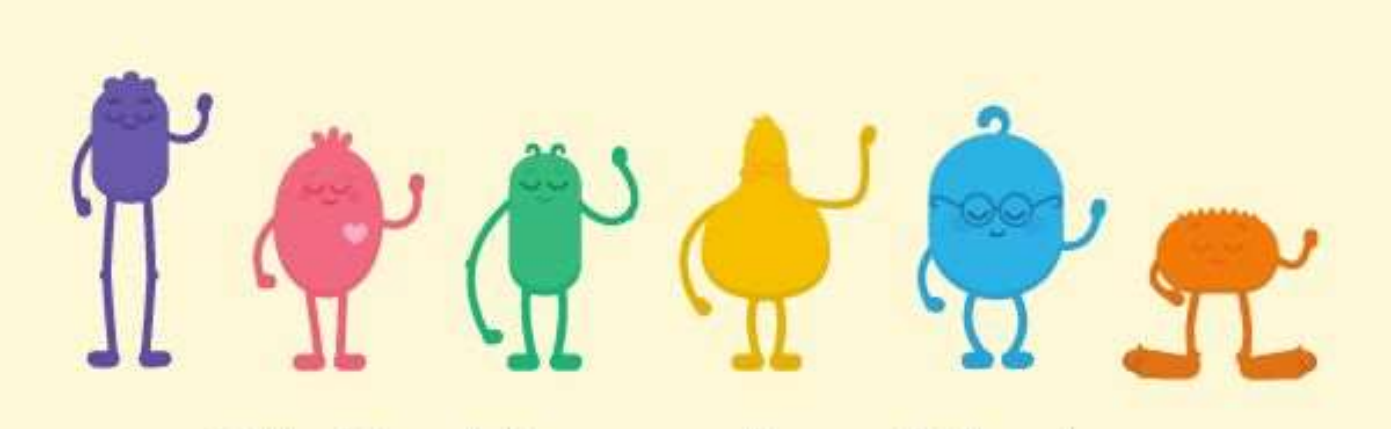
```
@ name_of_decorator

def my_function():
    # some function logic
```

We “decorate” a function with a **decorator** by adding the **@** sign and the name of a decorator on top of the function body.

PYTHON CITIZENS

DEFINITION



Functions ARE FIRST-CLASS citizens in Python

> In programming language design, a first-class citizen (also type, object, entity, or value) in a given programming language is an entity which supports all the operations generally available to other entities.

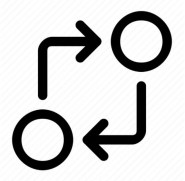
ADVANTAGES

1. we can pass them to other functions **as arguments**
2. we can return them from other functions **as values**
3. we can store them in variables and data structures

PYTHON DECORATOR

DEFINITION

A **decorator is a function** which takes another function as an argument and returns a **modified version of it**, enhancing its functionality in some way.



Sometimes also called:

- Decorators
- Metaprogramming

PYTHON DECORATOR

```
def decorator_function(function):

    def inner_wrapper_function()
            # some logic
            function()
            # some logic

    return inner_wrapper_function
```

PYTHON DECORATOR

KEY POINTS

- Decorators wrap a function, modifying its behavior
- Use decorators in a simpler way with the **@** symbol (also called “pie” syntax or syntactical “sugar”)
- A decorator is just a regular Python function, so it can be reused as many times as you want
- **Good practice:** move decorators to its own module, so that it can be used in many other functions. Use *import* to make it available in other modules and scripts

PYTHON DECORATOR

```
@decorator
def simple_function(arg1, arg2):
    #some logic
```

- How to decorate a function with parameters?
- Do we pass the function arguments to the decorator or its inner wrapper?

PYTHON DECORATOR

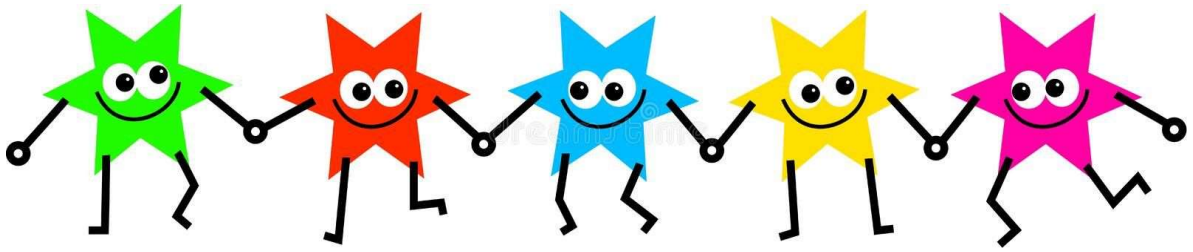
```
@decorator
def simple_function_A(arg1, arg2):
    #some logic
```

```
@decorator
def simple_function_B(arg1, arg2, arg3):
    #some logic
```

- How to create a **UNIVERSAL** decorator that can decorate a function with **ANY** number of arguments?

CHAINED DECORATORS

```
@decorator_one
@decorator_two
@decorator_three
def simple_function():
    #some logic
```



- Multiple decorators can be chained in Python.
- They all can be applied to the same function.

CLASS AS DECORATOR

```
class MyClassDecorator():

    def __init__(self, function)
        self.function = function

    def __call__(self):
        # some logic
        self.function()
        # some logic
```

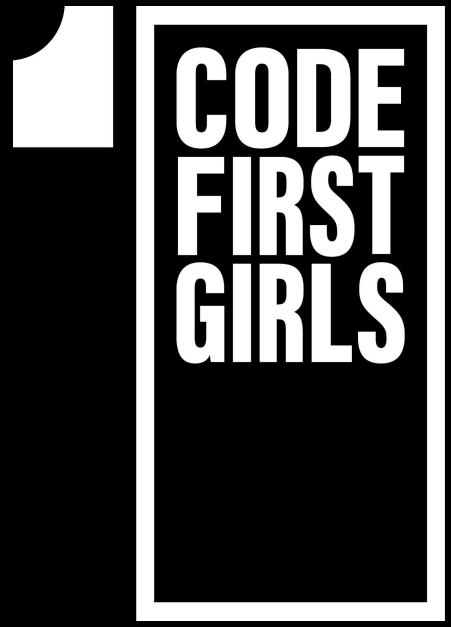
We can define a decorator as a class in order to do that, we have to use a *__call__* method of classes.

```
@MyClassDecorator
def function():
    # some logic
```

DEMO & EXERCISES

DECORATORS EXERCISES

- Timer
- Cacher



THANK YOU!