What we will be learning today 💻

1. PIP, PyPI
2. Modules
3. Creating packages
4. `__name__`, `__main__`
5. Function decorators
6. Challenge: hospital interface

1. PyPI, pip

PyPI:

Library of Python packages

`pip`:

Package manager (just like `npm`)

```
# Syntax: pip install <package_name>
pip install colorama
```

TODO:

Install `colorama` using `pip`

2. Modules vs packages??? 📦

2.1. Why do we need modules and packages?

1. As our project gets larger, it gets more and more difficult to organise all our files (or *modules*) in the same folder.
2. We can put our modules in sub-folders (or *packages*).

2.2. Modules

1. Any Python file is a module.
2. Its name would be the filename without the `.py` extension.

2.3. Packages

1. A collection of modules.
2. Its name is the name of the folder
3. `__init__.py` is added to the folder, sometimes it's empty, sometimes it's not 1. For now, it can be empty

2.4. `__init__.py`?

> Directories containing `__init__.py` file will be treated as modules.
> Furthermore, this is the first file to be loaded in a module, so you can use it to execute code that you want to run each time a module is loaded, or specify the submodules to be exported.

3. `__name__`, `"__main__"`

Why you cause me pain?

3.1. When do we usually see `__name__` and `"__main__"`?

```
if __name__ == "__main__":
    # execute only when you explicitly run your file
    main()
```

3.2. What is `__name__`?

1. Every module in Python has a special attribute called `__name__` .
2. It is a built-in variable that returns the name of the module.
    1. Recall what is a module in Python.
    2. What do you think `__name__` would be for a file called `index.py`?

TODO:

1. Print out `__name__` inside any Python file
2. Tell your shoulder partner what you see

3.3. How does `if __name__ == "__main__":` work?

1. You use to specify what functions to run when you explicitly run your file (e.g., when you run `python boggle.py`).
2. When you run a file, `__name__` will be set to `"__main__"`.
3. When you import it, `__name__` will simply be the name of the file.
4. Let's see this in action.

3.4. When to use it?

> You may want to run scripts individually or collectively for testing purposes.
> `if __name__ == "__main__":` allows you to control that.

> When you're writing a module, you only want the functions to run when they are imported and called.
> But you would still need to test and run the script individually. So this conditional pattern becomes handy!

4. Function decorators 🎀

Huh? How to decorate functions?

4.1. What is a function decorator?

1. Decorators modify the behavior of function or class without permanently modifying it.
2. **Very common** in Flask!
3. We do not need to write any ourselves, but it's good to understand.
4. EXCELLENT article

4.2. What does a decorator do?

1. Receives another function as an argument, modifies it, and then returns the changed function

4.3. Why do we use a decorator?

1. Sometimes we want to do something else (action 1) before we perform an action (action 2)
   e.g., We want to check if a user is logged in before displaying a protected webpage
2. Decorators allow us to do in a really nice and concise way

4.4. Passing functions like how we would variables

What can we expect to see?

```
def print_hello(name):
    print(f'Hello, {name}!')

def greet_nic(func):
    func('Nic')

greet_nic(print_hello)
```

4.5. These two are the same: 😮

> The @decorator syntax is what we call **syntactic sugar**

```
def some_decorator(func):
    # The inner function could be named anything
    # but by convention it's usually named wrapper
    def wrapper():
        print('This extends the functionality of func')
        func()

    return wrapper
```

```
@some_decorator
def func():
    print('This function is for demo purposes only')
```

TODO:

1. Run the code from the previous slide
2. Explain to your shoulder partner what you think is happening

5. Challenge: hospital interface 🏥

Demo time!