

# Programming fundamentals with Python

## Modules and packages

Pepe García [jgarciah@faculty.ie.edu](mailto:jgarciah@faculty.ie.edu)

# Plan for today

- Learn what are modules and why they exist

# Plan for today

- Learn what are modules and why they exist
- Learn about standard library modules



# Plan for today

- Learn what are modules and why they exist
- Learn about standard library modules
- Create our own modules



# Plan for today

- Learn what are modules and why they exist
- Learn about standard library modules
- Create our own modules
- Packages



With modules, Python provides functionality to package our functions, variables, and anything else into small packages ready to use anywhere else.

Every time we're creating a file in Python, we're creating a module. In order to import it we'll use its name without the `.py` extension.

To use modules, we need the import statement:

```
import module
```

When we import a module, all variables in the module are evaluated, all functions created, and top level statements executed.



When we import a module, all variables in the module are evaluated, all functions created, and top level statements executed.

## Example

Let's see how importing a module will make Python evaluate everything in it. In the Python file at the root of the repo, import the utilities.py file, let's see what happens.

## Warning

As you've probably noticed, when we import a module, everything inside it gets evaluated.

This can be dangerous if the module has statements at the top level you're not expecting.

But, where does Python search for modules?

But, where does Python search for modules?

- The directory containing the current module

But, where does Python search for modules?

- The directory containing the current module
- `sys.path`



sys contains functionality related with the current Python session

```
import sys
```

```
print(sys.path)
```

```
# ['',
```

```
#  '/Library/Developer/CommandLineTools/Library/Frameworks/Py
```

```
#  '/Library/Developer/CommandLineTools/Library/Frameworks/Py
```

```
#  '/Library/Developer/CommandLineTools/Library/Frameworks/Py
```

```
#  '/Users/pepe/Library/Python/3.7/lib/python/site-packages',
```

```
#  '/Library/Developer/CommandLineTools/Library/Frameworks/Py
```



The stdlib is a set of modules for a lot of different purposes

Python follows a batteries included approach, trying to give us as programmers everything we need



We have already used some modules from the standard library before:



The `time` module from the standard library contains utilities related for dealing with time. For example, we can use it for making our program stop for a certain amount of time.

```
import time

while True:
    time.sleep(1)
    print("hello!")
```

```
import math
```

```
math.ceil(3.4)
```

```
# 4
```

```
math.floor(3.4)
```

```
# 3
```

The `math` module contains useful functions and constants for doing numeric and mathematic operations



# Importing

We have already seen a way of importing modules, using the `import module` syntax, but there are more.

```
from module import variable # Cherry-picking what we want to import  
from module import * # importing everything from a module  
import module as alias # It's possible to give alias to imported module
```



# from ... import name

We can use this technique to import specific names from a module and use them directly.

```
from json import dumps
print(dumps([1,2,3]))
# '[1, 2, 3]'
```



# from ... import \*

```
from json import *  
print(dumps([1,2,3]))  
# '[1, 2, 3]'
```

This technique, also called wildcard import will import all names from a module.



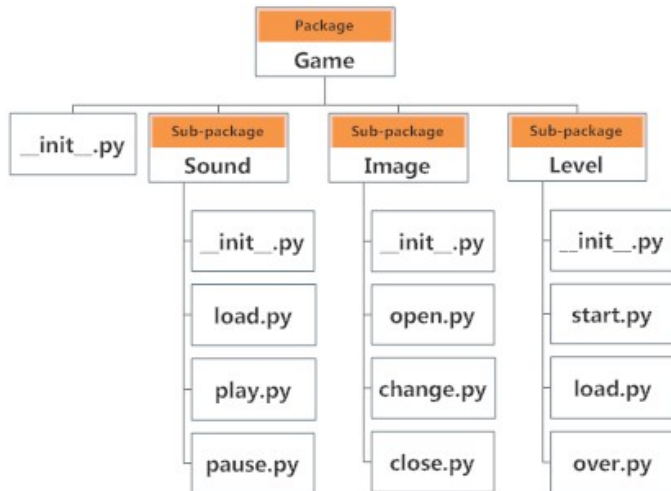
We can also rename modules when importing. This is specially useful whenever we will have a clash of names between imported stuff and our current module.

```
import json as momoa
momoa.dumps({"roles": ["Aquaman", "Khal  
{'roles': ["Aquaman", "Khal Drogo"]}]}
```

When Python projects get bigger it's common to start dividing them into files, and when there are too many into one single directory, into **packages**.

**Packages** are a way of organizing **modules** in Python projects.

# Packages





**Packages** are implemented using simple directories. These directories should contain a `__init__.py` (notice the two leading and trailing underscores) file in order for Python to find them.

In order to import modules from packages we will use a dot (.) as a directory separator.

```
import pandas.io.pickle # we're importing the pickle *module*  
                        # from the pandas.io *package*
```

```
pickle("whatever")
```

## Importing packages

Let's do a quick exercise here in class. In the `python` file in the root of the repository, import the dictionary from `core/utils/data.py` and print the second element.

- Import modules with import

# Recap

- Import modules with import
- Create our own modules as files



- Import modules with `import`
- Create our own modules as files
- `stdlib` contains a lot of useful stuff



- Import modules with `import`
- Create our own modules as files
- `stdlib` contains a lot of useful stuff
- third party libs can help when something is not on `stdlib`

