

# Modules and Packages



Dra. M<sup>a</sup> Dolores Rodríguez Moreno

# Objectives

## Specific Objectives

- Understanding and creating modules in Python
- Understanding and creating packages in Python

## Source

- <https://docs.python.org/3/reference/>
- <https://ellibrodepython.com/>
- Python Tutorial - Tapa blanda. Guido Van Rossum (2012)

# Outline

- **Introduction**
- Modules
- Packages

# Introduction

- All programming languages come pre-packaged with a standard library of functions that are designed to make our job easier
- Some of these functions are: print, input, range, etc
- Other more specialized functions are stored in a series of files called “modules” that Python can access upon request by using the “import” statement (e.g. *import time*)

# Outline

- Introduction
- **Modules**
- Packages

# Modules (I)

- A file containing Python definitions, functions, classes, statements
- Help organize code and make it reusable
- Benefits are code reusability, namespace management, ease of maintenance, etc
- Creating a **Module**
  - Create a file with .py extension
  - Example: mymodule.py
- If a module is too big create a **Package**

## Modules (II)

- The import statement tells Python to load the functions that exist within a specific module into memory and make them available in your code
- Because you don't see the inner workings of a function inside a module we sometimes call them “black boxes”
- A “black box” describes a mechanism that accepts input, performs an operation that can't be seen using that input, and produces some kind of output

# Creating Modules

- You can easily create your own modules that you can populate with your own functions
  - Create a new python script (i.e. “myfunctions.py”)
  - Place your function definitions in this script
  - Create a second python script (i.e “myprogram.py”)
  - Import your function module using the import statement:  
`import myfunctions`
- Call your functions using dot notation  
`myfunctions.function1(...)`  
`myfunctions.dosomethingelse(...)`



# Importing modules

mimodulo.py

```
def suma(a, b):  
    return a + b  
  
def resta(a, b):  
    return a - b
```

otromodulo.py:

```
import mimodulo  
  
print(mimodulo.suma(4, 3)) # 7  
  
print(mimodulo.rest(10, 9)) # 1
```

# Importing Specific components

mimodulo.py

```
def suma(a, b):  
    return a + b  
  
def resta(a, b):  
    return a - b
```

otromodulo.py:

```
from mimodulo import suma, resta  
  
print(suma(4, 3)) # 7  
  
print(resta(10, 9)) # 1
```

# Importing All Components

- Use `*` to import everything:

```
otromodulo.py:  
from mimodulo import *  
  
print(suma(4, 3)) # 7  
  
print(resta(10, 9)) # 1
```

# Modules paths

- Modules are usually in the same folder
- We can access modules in different folders
- Let's imagine:

----- example.py  
----- mifolder  
          ----- hello.py

```
hello.py  
def HelloWorld():  
  
    print("Hello")
```

- To import it from another place:

```
example.py:  
from mifolder.hello import *  
  
HelloWorld()
```

# Search paths

- Python searches for modules in the directories listed in *sys.path*
- When you import a module, the interpreter looks for it in the directories specified in *sys.path*
- This list of directories is initialized when you start Python and can be modified at runtime

```
sys.path
import sys

print(sys.path)

#['/content', '/env/python', '/usr/lib/python3.10',...'/root/.ipython']
```

## sys.argv

- Useful for creating scripts that accept arguments, such as command-line tools

```
import sys

# Mostrar la ruta donde se ejecuta el script
print(f"Ruta del script: {sys.argv[0]}")

# Mostrar argumentos pasados al script
if len(sys.argv) > 1:
    print("Argumentos proporcionados:", sys.argv[1:])
else:
    print("No se proporcionaron argumentos.")
```

# Modifying the path

- If we want to import a module that is in another place?
- You need to add to `sys.path` the path where we want Python to look for it

```
sys.path
```

```
import sys
```

```
sys.path.append(r'/path/to/your/module')
```

# Renaming Modules

- We can change the name of the module using “as”

```
modulowithlongname.py
```

```
def fun():  
  
    print("Bye")
```

```
micall.py
```

```
import modulowithlongname  
  
Modulowithlongname.fun()
```

```
micall.py
```

```
import modulowithlongname as m  
  
m.fun()
```



# Handling Import Errors

- Importing a module can throw an exception, if it has not been found
- Type of Exception: `ModuleNotFoundError`

```
otherimport.py
import non_existent_module # Raises ModuleNotFoundError

try:

    import non_existent_module

except ModuleNotFoundError as e:

    print("Error occurred:", e)
```

# What if I import this?

- We obtain non desired effects

otherimport.py

```
def suma(a, b):  
    return a + b  
  
c = suma(1, 2)  
  
print("The sum is:", c)
```

call.py

```
import otherimport as om  
  
om.suma(3, 4)  
  
#3  
  
#7
```

# What if I import this?

- Solution

otherimport.py

```
def suma(a, b):  
  
    return a + b  
  
if __name__ == '__main__':  
    c = suma(1, 2)  
  
    print("The sum is:", c)
```

call.py

```
import otherimport as om  
  
om.suma(3, 4)  
  
#7
```

# Reloading Modules

- Modules are imported only once even if you write it several times

mimodulo.py

```
print("Importing mimodulo")

def suma(a, b):

    return a + b

def resta(a, b):

    return a - b
```

otherimport.py

```
import mimodulo

import mimodulo

import mimodulo

# Ouput: "Importing mimodulo"
```

# Reloading Modules

- If we want to reload it, we need to be explicit using reload

```
otherimport.py
import mimodulo

import importlib

importlib.reload(mimodulo)

importlib.reload(mimodulo)
```

# Outline

- Introduction
- Modules
- **Packages**

# Packages

- There are the same as a directory containing other sub-packages and modules in a structured way
- It makes the sub-packages and modules easy to access
- This is an analogy to a folder, as folders allow us to store files
- Packages help us keep other sub-packages and modules to be used by the user when necessary

# Packages

- Must contain a special file called `__init__.py` (it can be empty)
- A package can be imported the same way as a module is imported
- Example structure:

```
mypackage/  
    __init__.py  
    module1.py  
    module2.py  
    subpackage/  
        __init__.py  
        module3.py
```



# Example

```
package
# Import a module from a package

from mypackage import module1

# Import a specific item from a module in a package

from mypackage.module2 import some_function

# Import from a subpackage

from mypackage.subpackage import module3
```

# Packages: Syntax

- You can use different Syntax to import Packages in Python

```
import packageName.moduleName
```

```
import packageName.subPackageName.moduleName
```

```
from packageName import moduleName
```

```
from packageName.subPackageName import moduleName
```

```
from packageName.moduleName import func_name
```

```
from packageName import *
```

## Exercise

- Create a module called `calculator.py` with basic arithmetic operations, then use it in another script
- At least you should be able to:
  - `add(5, 3)` # Output: 8
  - `subtract(10, 4)` # Output: 6
  - `multiply(2, 6)` # Output: 12
  - `divide(15, 3)` # Output: 5.0
  - `divide(10, 0)` # Output: Error: Division by zero