Modules

Julia Clemente

Máster en Tecnologías de la Información Geográfica
Universidad de Alcalá

/gso>

Objectives

1. Understand the relevance to use modules and packages.
2. Be able to install some widely used Python packages about geospatial software.
3. Be able to apply some modules and packages of both Python Standard Library and others Python Geospatial Libraries.

Índice

Introduction (I)

You loose everything when exit the interpreter.

- *Solution*: Write it down in a script.

When a script becomes big, it is difficult to maintain.

- *Solution*: Split your script in several ones.

As you get more scripts, you will need to reuse your functions

- *Solution*: Create a module.
- **Module**: A file that contains definitions, functions and classes.

If a module is too big, it is too difficult to maintain.

- *Solution*: Create a package.
- **Package**: A module of modules.

Introduction (II)

Why modules?

- **Main function**: Organization.
- **Reuse**: To provide software solutions, that have been proven to work, to solve similar problems.

Using modules

Creation and Implementation

A module is just a Python script with .py extension.

fibo.py

```
1  def fib(n):
2      """Print a Fibonacci series up to n """
3      a, b = 0, 1
4      while a < n:
5          print(a, end= ' ')
6          a, b = b, a+b
7      print()
8
9  def fib2(n):
10     """Print a Fibonacci series up to n """
11     result = [] # Declare a new list
12     a, b = 0, 1
13     while a < n:
14         result.append(a) # Add to the list
15         a, b = b, a+b
16     return result
```

Using modules

Where is it stored?

Accessible and reusable module:

- Set path in the file directory where the module is stored.
- Variable `PYTHONPATH`.

Using modules

How do I use them? (I)

```
>>> import fibo
>>> fibo.fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> fibo.__name__
'fibo'
>>> fib = fibo.fib
>>> fib(100)
1 1 2 3 5 13 21 34 55 89
```

Using modules

How do I use them? (II)

A module can import other modules.

- Name conflicts may arise: Each module has a symbol table.
- It means you should invoke it as `modname.itemname`

It is possible to import items directly:

- `from module import name1, name2`
- `from module import *`
- It uses the global symbol table (no need to use the modname)

```
>>> from fibo import fib, fib2
>>> fib(100)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

Using modules

How do I use them? (III)

List zip file contents (file.zip must exist. Open in read mode)

```
1  import zipfile
2
3  file = zipfile.ZipFile("file.zip", "r")
4
5  # list filenames
6  for name in file.namelist():
7      print(name)
8      print
9
10 # list file information
11 for info in file.infolist():
12     print(info.filename, info.date_time, info.file_size)
```

Several examples here:

`https://docs.python.org/3/library/zipfile.html`

`https://www.datacamp.com/community/tutorials/zip-file#ZM`

Using modules

How do I use them? (IV)

Error while importing:

- The module does not exist.
- The module name has not been well written.
- The module is not on the search path of Python modules:
  1. By default, it searches in the current directory.
  2. If it does not find it here, it then searches in the directories of the environment variable `PYTHONPATH`.
     - `echo $PYTHONPATH`
     - ```
       import sys
       print sys.path
       ```
  3. If it still does not find, it then searches in the installation directories of Python.
- How can I troubleshoot it?
       ```

Executing modules

Modules as scripts (I)

When a module is imported, its statements are executed

- It declares functions, classes, variables ...
- ... and also executes code.
- It serves to initialize the module.

Very useful to use modules as programs and libraries

Executing modules

Modules as scripts (II)

fibo2.py

```
1  def fib(n):
2      """Print a Fibonacci series up to n """
3      a, b = 0, 1
4      while a < n:
5          print(a, end= ' ')
6          a, b = b, a+b
7      print()
8
9  def fib2(n):
10     """Print a Fibonacci series up to n """
11     result = [] # Declare a new list
12     a, b = 0, 1
13     while a < n:
14         result.append(a) # Add to the list
15         a, b = b, a+b
16     return result
17
18 if __name__ == "__main__":
19     import sys
20     fib(int(sys.argv[1]))
```

```
$ python3 fibo2.py 50
1 1 2 3 5 8 13 21 34
```

Compiled Python files

- We said Python is an interpreted language but ...
  - ... this is almost a lie.
- Python, as other interpreted languages, has a speed-up trick.
  - It can use *bytecode*, just as Java.
- **Bytecode**: Intermediate code between machine code and source code.
  - Faster than source code, slower than machine code.
  - It is transparent to the programmer.
  - The first time a `.py` module is imported, it is compiled automatically, generating a `.pyc` file.
    - To improve loading speed.

Content of a module

The dir() function

Very usefull to get an insight to a module.

- It return the names defined in a module.
- Without arguments, it returns your names.

```
>>> import fibo, sys
>>> dir(fibo)
['__name__', 'fib', 'fib2']
>>> dir()
['__builtins__', ... , '__spec__']
>>> variable = 'Hello'
>>> dir()
['__builtins__', ... , '__spec__', 'variable']
```

Packages

Package concept (I)

If a module gets too big, arises many problems:

- Name collisions.
- It is good to organize modules in a bigger structure: *Packages*.

Packages can be seen as "dotted module names".

- It is just a module that contains more modules.
- Make life easier in big proyects.
- The name `A.B` designates a submodule `B` in a package named `A`.

Must contain a `__init__.py` file in the root directory.

- Executed when the package is imported for the first time.

Sound module structure

```
sound/                          Top-level package
      __init__.py               Initialize the sound package
      formats/                  Subpackage for format
          conversions
                __init__.py
                wavread.py
                wavwrite.py
                aiffread.py
                aiffwrite.py
                auread.py
                auwrite.py
                ...
      effects/                  Subpackage for sound effects
                __init__.py
                echo.py
                surround.py
                reverse.py
                ...
      filters/                  Subpackage for filters
                __init__.py
                equalizer.py
                vocoder.py
                karaoke.py
                ...
```

Packages

Importing a package (I)

Ways to use a package

- Import an individual module:
  - `import sound.effects.echo`
  - Use function as `sound.effects.echo.echofilter(input, output)`
- Alternative way to import an individual module:
  - `from sound.effects import echo`
  - Use function as `echo.echofilter(input, output)`
- Other alternative way to import an individual module:
  - `from sound.effects.echo import echofilter`
  - Use function as `echofilter(input, output)`

Packages

Importing a package (II)

- Imagine we run `from sound import *`:
  - In theory, it would import the whole package.
  - In practice, it would take too much time.
- There is a convention to avoid waste of resources:
  - There may be a variable `__all__` defined in `__init__`
  - `__all__` contains modules to be imported.
    About __init__.py, Click here!

sounds/effects/__init__.py

```
__all__ = ["echo", "surround", "reverse"]
```

Packages

Installing packages (I)

- Install with Anaconda:
  - Install Anaconda: About it, Click here!
  - Preferably, install package with conda: About it, Click here!
  - Installing packages with *pip*. Potential conflicts with packages already installed in Anaconda: About it, Click here!
- Command-line automatic tool: `pip`[1].
  - Very similar to `apt-get` in Linux.
  - How to install `pip` in Windows, Linux and OS X: About it, Click here!
- Standard python method:

```
$ python3 setup.py install
```

[1] Included by default with Python binary installers (version >= 3.4).

Packages

Installing packages (II)

- **`pip` usages** (from the system console):
  - Using `python -m`:

```
$ python -m pip install SomePackage
```

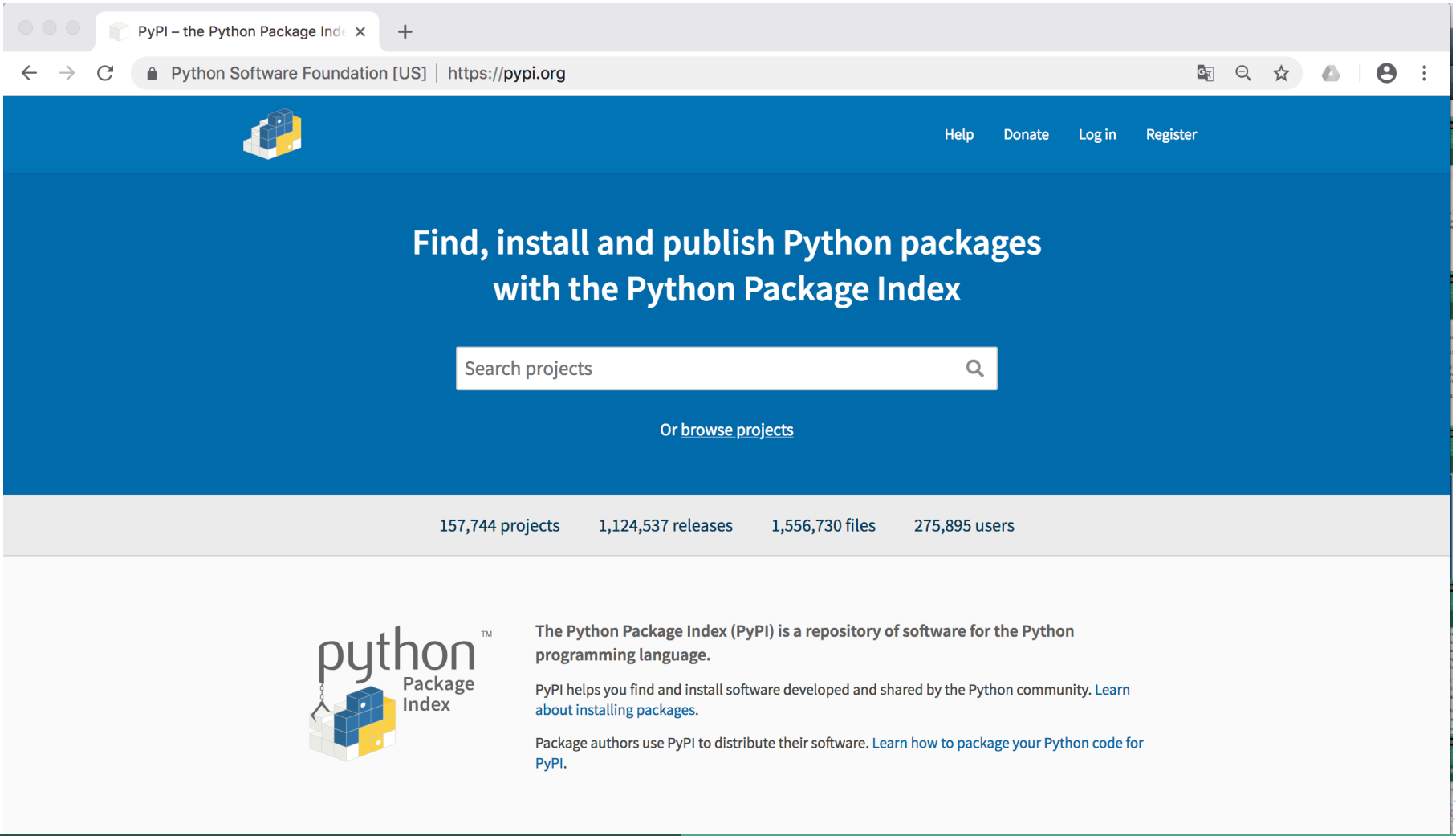
  - Using directly the `pip` version:

```
$ pip install SomePackage
$ pip3 install SomePackage # Python 3
```

- Example of the PIL package installation in Python 3.x:
  `$ pip3 install Pillow`

Packages

What has been developed?



os module

Functions to manipulate files and processes

- **Functions for managing files and directory paths**: `os.path`
- **Create directories**. Example: `os.mkdir('data')`
- **Current working directory**: `os.getcwd()`
- **Moving to a certain directory**. Example: `os.chdir('data')`
- **Value of an environment variable**. Example: `os.chdir(os.environ['HOME'])`
- **Rename a file**. Example: `os.rename('fich1.py', 'palindrome.py')`
- **Deleting a file**. Example: `os.remove('practica1.py')`
- **List the files in the current directory**. Example: `os.listdir(os.curdir)`
- **List the files in a certain directory**. Example: `os.listdir('C:\\data')`
- **Call operating system (execute OS services)**. Example: `os.kill`, `os.execv`, etc.

os.path module: Path handling

Examples

```
>>> import os
>>> os.chdir('/Users/jcp/Documents')
>>> os.getcwd()
'/Users/jcp/Documents'
>>> os.path.split('/Users/jcp/Documents/notas.txt')
('/Users/jcp/Documents', 'notas.txt')
>>> os.path.splitext('/Users/jcp/Documents/notas.txt')
('/Users/jcp/Documents/notas', '.txt')
>>> os.path.abspath('notas.txt')
'/Users/jcp/Documents/notas.txt'
>>> os.path.join('/Users/jcp', 'Documents/notas.txt')
'/Users/jcp/Documents/notas.txt'
>>> os.path.exists('/Users/jcp/Documents/notas.txt')
True
>>> os.path.isfile('/Users/jcp/Documents/notas.txt')
True
>>> os.path.isdir('/Users/jcp/Documents/notas.txt')
False
>>> os.path.getsize('/Users/jcp/Documents/notas.txt')
6106
```

sys module

It provides access to some variables maintained by the interpreter (at execution environment) and the functions that interact with the interpreter:

- **List the arguments passed to *script* on the command line**: `sys.argv`
- **Python output**. `sys.exit()`
- **Files for access to input, output and standard error of the interpreter**: `sys.stdin`, `sys.stdout`, `sys.stderr`, respectively.

sys module

Example

example_sys.py

```
import sys

# datos introducidos por teclado
data = sys.argv[:]

print 'data =  ', data

print "%d arguments were passed to the script %s: " \
          %(len(sys.argv) - 1, sys.argv[0])
for arg in sys.argv[1:]:
    print " %s" %arg
```

time module (I)

- It provides functions related to the measurement of time.
- Python provides the date and time of three ways:
  - Tuple: year-month-day-hour-min-sec-dayweek-dayyear-x (*tup*)
  - String (*str*)
  - Total of seconds since an origin (*sec*)

time module (II)

- **Current time**: `time.time()-> float`
- **Time elapsed since the start of the execution**. `time.clock()`
- **Pause n seconds**. `time.sleep(n)`
- **GMT: tuple expressing UTC**. `time.gmtime()`
- **Local time**. `time.localtime()`
- **Convert the tuple to a character string**. `time.asctime()`
- **Convert the tuple to a string according to a format specification**. `strftime(format[, tuple]) -> string`
- **Convert the tuple to seconds**. `mktime(tuple) -> floating point number`
- **Convert the seconds to a string**. `ctime(seconds) -> string`
- **Convert the string to a tuple**. `time.strptime(string[, format]) -> tuple (struct_time)`
- ...

time module (III)

Example

example_time.py

```
#!/usr/bin/python
import time

localtime = time.localtime(time.time())
print localtime
print type(localtime)
print "Local current time :", time.asctime(localtime)
```

Output:

```
time.struct_time(tm_year=2016, tm_mon=11, tm_mday=8, tm_hour=18,
tm_min=19, tm_sec=29, tm_wday=1, tm_yday=313, tm_isdst=0)
<type 'time.struct_time'>
Tue Nov 8 18:19:29 2016
```

More example here!

Cool code examples

Example 1: Open a web browser

browser.py

```
import webbrowser

url = input('Give me an URL: ')

webbrowser.open(url)
```

Cool code examples

Example 2: Create a thumbnail

thumbnail.py

```
from PIL import Image

size = (128, 128)
saved = "africa.jpeg"

im =  Image.open("africa.png")
im.thumbnail(size)
im.save(saved)
im.show()
```

Source

africa.jpeg

Cool code examples

Example 3: List a directory contents

dir.py

```
import os

os.system("clear")

path = input("Specify a folder >> ")

for root, dirs, files in os.walk(path):
    print(root)
    print("--------------------")
    print(dirs)
    print("--------------------")
    print(files)
    print("--------------------")
```

Source

Cool code examples

Example 4: Send an email with Gmail

gmail.py

```
"""The first step is to create an SMTP object,
each object is used for connection
with one server."""

import smtplib
server = smtplib.SMTP('smtp.gmail.com', 587)

# Next, log in to the server
server.login("youremailusername", "password")

# Send the mail
msg = "\nHello!" # \n separates the message from the headers
server.sendmail("you@gmail.com", "target@example.com", msg)
```

Source

Bibliographic references I

- [van Rosum, 2022] G. van Rossum, Jr. Fred L. Drake.
  Python Tutorial Release 3.10.7, chapter 6.
  Python Software Foundation, 2022.
  https://docs.python.org/3/download.html

- [Westra, 2016] E. Westra.
  Modular programming with Python.
  Packt Publishing, 2016.

- [vanRosum, 2021] G. van Rossum, Python Development Team.
  The Python Library Reference. Release 3.6.4.
  New Publisher, 2021.

Bibliographic references II


[Lutz, 2013] M. Lutz.
Learning Python.
O'Reilly, 5th. edition. 2013.


[Hellmann, 2017] D. Hellmann.
The Python 3 Standard Library by Example (Developer's Library).
Addison Wesley Professional, 1st. edition. 2017.