Modules

Inteligencia Artificial en los Sistemas de Control Autónomo Máster en Ciencia y Tecnología desde el Espacio

Departamento de Automática





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.

Table of Contents

- I. Introduction
- 2. Modules
 - Using modules
 - Executing modules
 - Compiled Python files
 - Content of a module
- 3. Packages
 - Package concept
 - Importing a package
 - Installing packages
 - What has been developed about Python packages?
- 4. The Python Standard Library
 - os module
 - sys module
 - time module
- 5. Other cool code examples
 - Example 1: Open a web browser
 - Example 2: Create a thumbnail
 - Example 3: List a directory contents
 - Example 4: Send an email with Gmail



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

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

Modules

A module is just a Python script with . py extension

```
fibo.py
  def fib (n):
         Print a Fibonacci series up to n
      a, b = 0, I
      while a < n:
          print(a, end= ' ')
          a, b = b, a+b
      print()
  def fib2(n):
         Print a Fibonacci series up to n
      result = [] # Declare a new list
      a, b = 0, I
      while a < n:
          result.append(a) # Add to the list
          a, b = b, a+b
      return result
16
```

Using modules

Where is it stored?

Accessible and reusable module:

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



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
```



How do I use them? (II)

w do i use them: (11)

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)

```
import zipfile

import zipfile . ZipFile("file.zip", "r")

# list filenames
for name in file.namelist():
    print(name)
    print

# list file information
for info in file.infolist():
    print(info.filename, info.date_time, info.file_size)
```

Several examples here: http://pymotw.com/2/PyMOTW-1.132.pdf

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.
 - 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?



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
def fib (n):
      """ Print a Fibonacci series up to n
     a, b = 0, I
     while a < n:
         print(a, end= ' ')
         a, b = b, a+b
     print()
 def fib2(n):
      """ Print a Fibonacci series up to n
     result = [] # Declare a new list
     a, b = 0, I
     while a < n:
          result.append(a) # Add to the list
         a, b = b, a+b
      return result
    __name__ == "__main__":
     import sys
     fib (int (sys.argv[1]))
```

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

Compiled Python files

We said Python is an interpreted language

... 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 file is executed, it is compiled automatically, generating a .pyc file.



Content of a module

The dir() function

Very usefull to get an insight to a module

- It returns 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']
```

Package concept (I)

If a module gets too big, many problems arise

- 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



Packages

Package concept (II)

Sound module structure

```
sound /
                            Top-level package
                            Initialize the sound package
      __init__.py
      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)

Alternative way to import an individual module

- from sound.effects.echo import echofilter
- Use function as echofilter(input, output)



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

```
sounds/effects/__init__.py
```

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



Packages

Installing packages

Command-line automatic tool:pip

• Very similar to apt-get in Linux

pip usage

\$ python -m pip install SomePackage

pip alternative usage

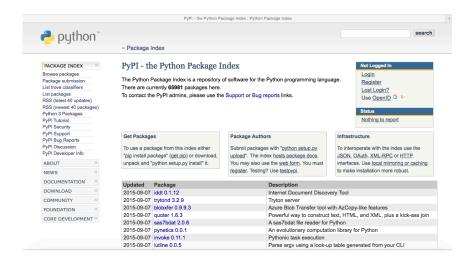
\$ pip3 install SomePackage

Example of the PIL installation:

\$ pip3 install Pillow
(More info)



Packages What has been developed?





Functions to manipulate files and processes

- Functions for managing files and paths: os.path
- Create directories. Example: os.mkdir('data')
- Current working directory: os.getcwd()
- Moving to a certaing 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.



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.



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[o])
for arg in sys.argv[1:]:
    print " % s" % arg
```

- 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-day year-x (tup)
 - String (str)
 - Total of seconds since an origin (sec)



- Current time: time()
- Time elapsed since the start of the execution. clock()
- Pause n seconds. sleep()
- GMT.gmtime()
- Local time. local time()
- Convert the tuple to a character string. asctime()
- Convert the tuple to a string. strftime()
- Convert the tuple to seconds. mktime()
- Convert the seconds to a string. ctime()
- Convert the string to a tuple. strptime()
- ...



Example

```
example_time.py

#!/usr/bin/python
import time;

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

Output:

Local current time: Fri Nov 20 20:45:34 2015



Example 1: Open a web browser

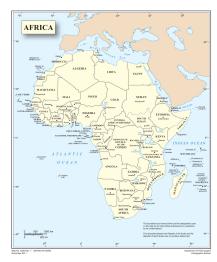
browser.py

```
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.jpg" im = Image.open("africa.tif") im.thumbnail(size)



africa.jpg

im.save(saved)
im.show()

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)



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



[Lutz, 2013] M. Lutz. Learning Python.

O'Reilly, 2013.

[Bahit, 2008] E. Bahit.

Curso: Python para principiantes.

Creative Commons Atribución-NoComercial 3.0, 2012.

Bibliographic references II

[vanRosum, 2012] G. van Rossum, Jr. Fred L. Drake. The Python Library Reference. Release 3.2.3. Python Software Foundation, 2012.

[Hellman, 2011] D. Hellman.
The Python Standard Library by Example (Developer's Library).
Addison Wesley Professional, 2011.