Curso Básico de Python

6. Módulos

Contenido

Los programas escritos en Python se guardan en archivos denominados scripts y módulos. A su vez, estos archivos pueden integrarse en paquetes, permitiendo así la construcción ordenada de bibliotecas o aplicaciones complejas. En este capitulo repasamos la terminología y analizamos en detalle la estructura modular de los programas escritos en Python.

Lecturas recomendadas:

1. Módulos en el Tutorial de Python.

1 Scripts, módulos y paquetes: terminología

Un *programa* es un conjunto de sentencias que indican al computador como realizar una tarea. Una *sentencia* es una instrucción escrita en un lenguaje de programación para realizar una acción. Las sentencias se suelen guardar en un archivo de texto, para poder ser ejecutadas en forma reiterada sin volver a escribirlas.

Cuando las sentencias se escriben para ser ejecutadas directamente desde un menú o desde una terminal con línea de comandos, el archivo donde se guardan estas instrucciones se denomina un *script*, y debe tener como extensión `.py` para poder ser reconocido como ejecutable en Python. La ejecución de un script responde a un comando tal como

```
$ python3 nom_script.py
```

El comando anterior ejecuta en forma sucesiva las instrucciones (sentencias) incluidas en un archivo llamado `nom_script.py`.

Cuando una tarea se repite muchas veces, se la escribe en forma de *función*, un nombre asociado a un conjunto de sentencias. Al escribir el nombre de la función (invocar o ejecutar la función), se ejecuta ese conjunto de sentencias. Es posible guardar varias funciones en un archivo, para tenerlas disponibles cuando se las necesite, ya sea en un programa o en el intérprete de comandos. Un *módulo* es un archivo donde se guardan definiciones (e.g. variables) y funciones; se implementa como un archivo con extensión `.py`. Para usar las funciones o variables defindas en un módulo, se lo debe importar, ya sea hacia otro programa o hacia el intérprete de comandos:

```
>>> import nom_mod         # importa el módulo nom_mod.py
>>> nom_mod.fn_1()         # ejecuta la función fn_1 definida en nom_mod
>>> print(nom_mod.var_1)   # muestra la variable var_1 de nom_mod
```

Un módulo con definiciones de variables y funciones no es, en principio, ejecutable desde un menú o desde el intérprete de comandos, i.e. no es un script. No obstante, un módulo puede ejecutar sus funciones si se lo indica agregando al final sentencias tales como:

```
if __name__ == "__main__":   # para ejecutar este módulo como script
    fn_1()          # ejecuta la función fn_1 definida en este módulo
    print(var_1)    # muestra la variable var_1 definida en este módulo
```

Las definiciones de script y módulo se solapan; un módulo puede actuar como un script.

Un programa complejo puede tener muchos módulos. Un conjunto de módulos puede agruparse en un *paquete*, identificado con un nombre. Un paquete se implementa como un directorio donde se colcoan los módulos más un archivo de nombre `__init__.py`. Este archivo puede estar vacío, aunque suele contener docstrings descriptivos del módulo. La presnecia de ese archivo permite a Python reconocer ese directorio como un paquete. Ejemplo de paquete:

```
/tmp$ ls -R nom_paq/     # muestra los archivos en el directorio
nom_paq/:
__init__.py  mod_2.py
```

La presencia del archivo `__init__.py` permite a Python reconocer este directorio como el paquete de nombre `nom_paq` donde está contenido el módulo `mod_2`. Para importar el módulo, la notación de `.` permite identificar el módulo en una jerarquía de directorios:

```
>>> import nom_paq.mod_2        # importa módulo mod_2 del paquete nom_paq
>>> nom_paq.mod_2.fn_2()        # invoca función fn_2 de mod_2
>>> print(nom_paq.mod_2.var_2)  # muestra la variable var_2 de mod_2
```

2 Módulos

Para usar las funciones de un módulo, es preciso indicar en el programa o en el intérprete de comandos que se usará el contenido de ese módulo, con la declaración `import`. Hay varias formas de hacerlo:

```
>>> import nom_mod             # importa el módulo
>>> nom_mod.fn_1("a", True)    # invoca la función fn_1 del módulo nom_mod
>>> from nom_mod import fn_2   # importa directamente la función fn_2
>>> fn_2("b", 3)               # no requiere indicar el módulo
>>> from nom_mod import fn_3 as f3   # asigna un alias
>>> f3(nombre="Anónimo")             # invoca la función con el alias
```

El comando `import` permite acceder a las funciones y variables guardadas un módulo desde el intértprete de comandos. Observación: en la sentencia `import` no figura la extensión del módulo, se da por conocida.

Sintaxis, de `help("import")`, en formato simplificado:

```
import módulo [as identificador] (, módulo as identificador)*
from módulo import identificador [as identificador] (, identificador as
identificador)*
from modulo import *
módulo ::= (identificador.)*identificador
```

Ejemplos:

```
import mod_1                            # importa un módulo
import mod_1, mod_2                     # importa varios módulos
import mod_1 as m1, mod_2 as m2         # módulos con alias
from mod_1 import fn_1                  # importa una función específica
from mod_1 import fn_1, fn_2            # importa dos funciones
```

```
from mod_1 import fn_1 as f1, fn_2 as f2   # dos funciones con alias
from mod_1 import *  # importa todos los elemetos del módulo; DESACONSEJADO
```

Los módulos se importan solo una vez. Si luego de importar un módulo, ese módulo resulta modificado, debe recargarse, lo cual puede hacerse así:

```
>>> import importlib
>>> importlib.reload(mod_prueba)
```

3 Paquetes

En el siguiente ejemplo, el paquete `paqs` contiene dos sub paquetes, `paq_a` y `paq_b`. Cada uno de estos tres paquetes contiene uno o varios módulos.

```
paqs:              # paquete de orden superior, contiene a todos los demás
  __init__.py        # define directorio como paquete Python
  mod_igual.py       # módulo de nombre repetido en varios paquetes

  paq_a              # paquete "paq_a" dentro del paquete "paqs"
    __init__.py        # define directorio "paq_a" como paquete Python
    mod_a1.py          # módulo dentro de "paq_a"
    mod_a2.py          # módulo dentro de "paq_a"
    mod_igual.py       # módulo dentro de "paq_a" con nombre repetido

  paq_b              # paquete "paq_b" dentro del paquete "paqs"
    __init__.py        # define directorio "paq_b" como paquete Python
    mod_b1.py          # módulo dentro de "paq_b"
    mod_igual.py       # módulo dentro de "paq_b" con nombre repetido
```

En la estructura anterior, hay tres módulos distintos llamados `mod_igual`, uno en cada paquete, pero esto no origina confusión de nombres si se indica el paquete al cual pertenece cada uno. Los siguientes ejemplos muestran formas de importar módulos e invocar sus funciones:

```
>>> from paqs import mod_igual        # importa un módulo
>>> from paqs.paq_a import mod_a1     # importa otro módulo
>>> dir(mod_igual)    # muestra variables y funciones del módulo
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__',
'__name__', '__package__', '__spec__', 'fn_igual']
>>> dir(mod_a1)       # muestra variables y funciones del módulo
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__',
'__name__', '__package__', '__spec__', 'fn_a11', 'fn_a12', 'fn_igual']
>>> mod_a1.fn_a11()   # invoca una función indicando el módulo
En la función paqs.paq_a.mod_a1.fn_a11().
>>> from paqs.paq_b.mod_b1 import fn_b11    # importa una función específica
>>> fn_b11()
En la función paqs.paq_b.mod_b1.fn_b11().
>>> from paqs.paq_a import mod_igual as a_mod_igual  # define un alias
>>> a_mod_igual.fn_igual()    # indica el módulo donde está la función...
En la función paqs.paq_a.mod_igual.fn_igual()
>>> mod_igual.fn_igual()      # ...y evita confundir por el nombre repetido
En la función paqs.mod_igual.fn_igual().
```

4 Espacio de nombres

Tanto los módulos como los paquetes definen un *espacio de nombres* propio, solo se reconocen los nombres definidos dentro de ese paquete o módulo. Así, en diferentes paquetes puede puede haber módulos con el mismo nombre, y en diferentes módulos puede haber funciones con el mismo nombre, así como también variables. El nombre de una variable de módulo o una función debe ser

única dentro de ese módulo, pero puede aparecer el mismo nombre en otros módulos. En la importación de los módulos, deberá tenerse en cuenta esta potencial repetición de nombres.

El siguiente código muestra la sustitución de funciones `fn_igual` de los módulos `mod_igual` al ser importadas. Aunque los nombres de los módulos y de las funciones se repiten, como están en distintos paquetes no hay confusión; eso sí, la nueva importación sustituye a la anterior:

```
>>> from paqs import mod_igual
>>> mod_igual.fn_igual()
En la función paqs.mod_igual.fn_igual().
>>> from paqs.paq_a import mod_igual
>>> mod_igual.fn_igual()
En la función paqs.paq_a.mod_igual.fn_igual().
>>> from paqs.paq_b import mod_igual
>>> mod_igual.fn_igual()
En la función paqs.paq_b.mod_igual.fn_igual().
```

Para invocar cada una de estas funciones con nombre repetido dentro de módulos también con nombre repetido, es necesario mantener la identificación completa:

```
>>> import paqs.mod_igual, paqs.paq_a.mod_igual, paqs.paq_b.mod_igual
>>> paqs.mod_igual.fn_igual()
En la función paqs.mod_igual.fn_igual().
>>> paqs.paq_a.mod_igual.fn_igual()
En la función paqs.paq_a.mod_igual.fn_igual().
>>> paqs.paq_b.mod_igual.fn_igual()
En la función paqs.paq_b.mod_igual.fn_igual().
```

La importación con aliases puede resultar más clara:

```
>>> from paqs import mod_igual as igual
>>> from paqs.paq_a import mod_igual as igual_a
>>> from paqs.paq_b import mod_igual as igual_b
>>> dir(igual_b)
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__',
'__name__', '__package__', '__spec__', 'fn_igual']
>>> igual_b.fn_igual()
En la función paqs.paq_b.mod_igual.fn_igual().
>>> igual.fn_igual()
En la función paqs.mod_igual.fn_igual().
```

Si bien esta situación de repetición de nombres no es aconsejable, una función con el mismo propósito en distintos módulos puede razonablemente tener el mismo nombre, con la precaución de invocarla dentro del módulo y paquete correspondiente en cada caso.

5 Rutas de búsqueda

Al intentar importar un módulo, el intérprete lo busca en estos lugares sucesivos:

1. como módulo propio de Python, en sus bibliotecas;
2. en el directorio desde donde fue invocado el intérprete;
3. en la lista de directorios contenida en las variables de entorno `PATH` y `PYTHONPATH`.

La variable de entorno `PATH` puede modificarse desde el intérprete de Python, de esta forma:

```
>>> import sys        # importa biblioteca sys de Python
>>> sys.path = sys.path + ["/tmp"]   # agrega /tmp a las rutas de búsqueda
>>> print(sys.path)   # muestra todas las rutas de búsqueda.
```

Para hacer este cambio permanente, en Linux, agregar al archivo `.bashrc` en el directorio propio del usuario una línea similar a la siguiente, indicando el directorio a agregar como ruta de búsqueda (en este ejemplo `/tmp`):

```
export PYTHONPATH=$PYTHONPATH:/tmp
```

Para releer y acutalizar las variables de entorno, es preciso dar el comando

```
$ source ~/.bashrc
```

En lo sucesivo, el nuevo directorio estará en la lista de rutas de búsqueda para módulos de Python.

6 Ejercicios

Sugerencia: para realizar las siguientes tareas, es posible que deba ajustarse la variable `sys.path`.

1. Escribir un módulo `porx` con las funciones `por_dos(n)` y `por_tres(n)` que reciben un número n y devuelven el valor multiplicado por 2 y por 3.
   Importar el módulo; verificar que están disponibles las funciones indicando el módulo (e.g. `porx.por_dos`).
   Importar ahora las funciones directamente, para ejecutar sin aludir al módulo.

2. ¿Por qué no se considera "buena práctica" importar todos los nombres de un módulo?

3. Escribir un script `callporx.py` que importe el módulo `porx` y pruebe las funciones `por_dos()` y `por_tres()`. Debe ejecutar con
   ```
   $ python callporx.py
   ```

4. Agregar al módulo `porx` las variables
   ```
   var1="primera"; var2 = 2; _var3="no importa", __var4 = "tampoco importa"
   ```
   Importar el módulo al intérprete Python y ver si están disponibles. Modificar alguna de las variables, recargar el módulo y ver si la variable tomó el nuevo valor (i.e. si el módulo efectivamente se recargó). Nota: las variables que empiezan con _ se consideran "ocultas", que no deben ser modificadas por el usuario del programa.

5. Paquetes. En los ejemplos, analizar la estructura del paquete `paqs`. Desde el directorio que contiene `paqs`, pobar ejecutar los módulos y las funciones contenidas, tal como se mostró en la sección sobre Paquetes de este capitulo:
   a) desde el intérprete Python,
   b) invocando IDLE desde ese mismo directorio.
   El script `test_paqs.py` prueba la estructura de paquetes, mostrando las variables y las funciones en todos los módulos.

6. En los ejemplos, en el directorio `paqs_html` se encuentra la documentación del paquete `paqs`, generada automáticamente con el utilitario `pydoctor`, a partir de los docstrings contenidos en el código. Analizar la documentación de paquetes, módulos y funciones. Esta documentación fue generada desde el directorio donde está contenido `paqs`, con el comando
   ```
   $ pydoctor --make-html --html-output=paqs_html –add-package=paqs
   --project-name="Ejemplo de paquetes"
   ```
   Documentación de `pydoctor`: https://pydoctor.readthedocs.io/.

7. Anticipar, probar y explicar las salidas de estos comandos:
   ```
   import paqs; dir(paqs); help(paqs)
   import paqs.mod_igual; help(paqs.mod_igual); paqs.mod_igual.fn_igual()
   import paqs.paq_b.mod_igual; help(paqs.paq_b.mod_igual);
   paqs.paq_b.mod_igual.fn_igual()
   ```

8. Rutas de búsqueda. En un directorio del usuario, o en `/tmp`, escribir un módulo `mod_prueba` con una variable `var_mod` con valor "var_mod en mod_prueba" y una función `fn` que imprima "función fn en mod_prueba".

Verificar que es posible importar el módulo, visualizar la variable y ejecutar la función:
a) desde ese mismo directorio, en una terminal de comandos, invocada con `python3`;
b) desde IDLE, pero solo si se arranca IDLE desde ese mismo directorio;
c) iniciar IDLE desde el menú, agregar el directorio a `sys.path` y verificar la importación.
d) desde otro directorio, verificar que no se puede importar `mod_prueba` con `python3`.
e) agregar la ruta al directorio donde se encuentra `mod_prueba` a la variable de entorno `PYTHONPATH`; verificar que ahora sí es posible importar `mod_prueba` desde otro directorio.

9. Espacio de nombres. Anticipar, probar y explicar las salidas de estos comandos:
```
from paqs.paq_b.mod_igual import fn_igual; fn_igual()
from paqs.mod_igual import fn_igual; fn_igual()
```
Explicar el comportamiento en términos de espacios de nombres.
