Módulos

Curso de Python 3 Iniciación

César Husillos Rodríguez

IAA-CSIC

1-5 de Abril de 2019

Índice

Índice

Módulo

¿Qué es?

- Un módulo es un fichero que contiene código Python.
- Su extensión es `.py`.
- Almacena declaración de variables e implementación de funciones, además de la posibilidad de hacer referencia a otros módulos.

Módulo

¿Para qué sirve?

- Desde un punto de vista de **Organización**
  - Agrupar código común.
  - Estructurar el código en ficheros y directorios.
- Con respecto a la **Reutilización**
  - Aprovechar el trabajo anterior.
  - Código ya probado.
- Si nos fijamos en el **Espacio de Nombres**
  - Usamos variables cuyo nombre ya existe en el código escrito sin que una interfiera en la visibilidad de la otra. (Algo parecido se ha visto en el tema de funciones.)

Módulo

Organización

- Todo empieza por programas simples.
- A medida que se aprende sobre `Python`, la complejidad de los programas aumenta.
- El número de líneas pasa de decenas a cientos, y el programa se vuelve difícil de revisar.

La tendencia normal es la de dividir el código complejo en partes (funciones o módulos).

Éstas piezas o fragmentos de código se gestionan de forma independiente (programación, pruebas, mejoras...) y luego se juntan en programas más complejos.

Módulo

Reutilización

- En nuestro trabajo diario, nuestras tareas se centran en temas concretos.
- Desarrollamos código para resolver un problema. Se plantea otro diferente y nos damos cuenta de que nuestro programa se parece a parte de otro que ya hemos hecho antes.

¿Trabajamos dos veces?

- Si organizamos nuestro trabajo en módulos podremos
  1. Aprovechar lo que ya hemos hecho (o han hecho otros).
  2. Tener la certeza de que lo que usamos funciona (porque ya se probó antes).

Módulo

Espacio de nombres

Pese a que el número de nombres que se pueden usar para funciones y variables es infinito, un programador usa unas pocas docenas. Los motivos principales:

- No se desean nombres largos.
- Se pretende que el nombre se relacione semánticamente con el resultado de la variable o la tarea asignada a la función: `resultado`, `suma`, `apellidos`...)

Módulo

Espacio de nombres

Es muy probable que, al crear un programa nuevo e importar un módulo, haya nombres comunes.

- ¿Cómo evitar que una variable de módulo sobreescriba una de programa? (o viceversa).
- ¿Cómo diferenciar los nombres de módulo de los de programa?
- ¿Cómo saber qué variable o función estoy usando?

Supongamos que el módulo mod1 contiene la variable edad y que vale 30.

```
>>> import mod1
>>> edad = 5
>>> edad
5
>>> mod1.edad
30
```

Se llama **espacio de nombres** porque se crea una zona reservada para los nombres del módulo.

Módulo

Implementación

El proceso de creación de un módulo Python pasa por tres fases:

1. Se genera un fichero con extensión `.py`
2. Se escriben todas las definiciones de funciones, clases, ... y el código ejecutable. Este código ejecutable puede servir:
   - Para inicializar las variables del módulo.
   - Para ejecutar rutinas de testeo del módulo.
3. Se guarda el fichero en una ruta predeterminada.

Ejercicios propuestos

Módulo

1. Cree un módulo llamado `operaciones_basicas` con las siguientes funciones: `sumar`, `restar`, `multiplicar`, `dividir`.
   - Cada función recibirá dos parámetros (los operandos).
   - Cada función devolverá el resultado de la operación.
   - Controle la excepción de la división por cero en la función dividir.

Ejercicios propuestos

Módulo

2. Complete el ejercicio anterior. Documente el módulo y las funciones con estos textos
   - Modulo – Módulo operaciones_basicas.
   - sumar – Función que suma los dos parametros y devuelve el resultado.
   - restar – Función que resta los dos parametros y devuelve el resultado.
   - multiplicar – Función que multiplica los parámetros y devuelve el resultado.
   - dividir – Función que divide los parámetros y devuelve el resultado. Lanza una excepción en caso de división por cero. El código de esa excepción mostrará el mensaje `ERROR: No se puede dividir por cero` y lanzará una excepción del tipo `ZeroDivisionError`.

Módulo

Utilización

Para usar un módulo desde un script se debe

1. Crear un fichero con código Python (`.py`).
2. Importar el módulo.
   Se usará la instrucción `import` de Python seguida del nombre del fichero que contiene el módulo **sin la extensión** `.py`.
3. Usar las variables y funciones del módulo en el código del nuevo programa Python.

Módulo

Error de importación

Si al importar el módulo obtengo un error

```
>>> import mimodulo
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'mimodulo'
```

puede deberse a dos causas:

a. Ese módulo no existe o no he escrito bien su nombre.

b. El módulo no está en la ruta de búsqueda de módulos de Python.

Módulo

Ruta de búsqueda

Si estamos seguros que de el módulo existe pero da error al importar, comprobaremos que el fichero del módulo está en la ruta de búsqueda de módulos de Python.

¿Cuál es la ruta de búsqueda de módulos en Python?

1. Por defecto, busca en el directorio actual.
2. Si no lo encuentra, busca en los directorios contenidos en la variable de entorno `PYTHONPATH`.
3. Si aun así no lo encuentra, busca en los directorios de instalación de Python.

Es posible ver las rutas que usa nuestro Python de dos formas:

- Variable de entorno del sistema: `PYTHONPATH`.
- Ojeando la variable `path` del módulo `sys`.

Módulo

Ruta de búsqueda

Si no está el módulo en todas las posibles rutas de búsqueda de Python

¿Cómo soluciono el problema?

Entre las posibles opciones se recomiendan

1. Agregar el directorio donde está nuestro módulo a la lista dada por la variable `path` del módulo `sys`. Este es un **cambio** que se aplica de forma **temporal** a nuestra sesión o script.
2. Modificar o crear la variable de entorno `PYTHONPATH` en nuestro sistema operativo. Este **cambio** es **permanente**.
3. Copiando (o mejor, moviendo) nuestro módulo a un directorio registrado por Python en su ruta de búsqueda. Es **cambio** es **permanente**.

Módulo

PYTHONPATH

Para modificar la variable PYTHONPATH en nuestro sistema operativo:

- Windows
  1. Pulsar sobre el botón **Inicio**, después con el botón derecho del ratón sobre **Equipo** y seleccionar **Propiedades**. En la pestaña **Opciones avanzadas** pulsar sobre el botón **Variables de entorno**.
  2. Buscar PYTHONPATH y, si no está, pulsar sobre el botón **Nueva...**.
  3. Establecer **Nombre de la variable** a PYTHONPATH.
  4. Establecer el **Valor de la variable** a

     %USERPROFILE%\dir_modulos[1]

[1] donde

- %USERPROFILE% es nuestro directorio de usuario en Windows.
- dir_modulos es el directorio donde está nuestro módulo.

Módulo

PYTHONPATH

Para modificar la variable PYTHONPATH en nuestro sistema operativo:

- Linux
  1. Editar nuestro fichero de configuración de usuario en nuestro directorio $HOME.
  2. Agregar la siguiente línea al final del fichero
     - `export PYTHONPATH=$HOME/dir_modulos` (intérprete BASH).
     - `setenv PYTHONPATH $HOME/dir_modulos` (intérprete CSH).
  3. Guardar el fichero.
  4. Abrir otra consola/terminal o recargar el fichero de configuración mediante el comando
     `source $HOME/.bashrc` (para BASH)
     `source $HOME/.cshrc` (para CSH)

Módulo

sys.path

- La variable `sys.path` es una lista que contiene las rutas donde PYTHON busca módulos.
- Como tal, puede ser modificada dentro del código de un fichero PYTHON (script) o durante una sesión de consola.

```
>>> import mimodulo
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'mimodulo'
>>> import sys
>>> sys.path
['', '/usr/lib/python36.zip', '/usr/lib/python3.6',
'/usr/lib/python3.6/lib-dynload',
'/usr/local/lib/python3.6/dist-packages',
'/usr/lib/python3/dist-packages']
>>> sys.path.append("/home/cesar")
>>> import mimodulo
>>>
```

Módulo

sys.path

- Ahora que el módulo está cargado, puedo obtener la información acerca de su contenido (función `help()`).

```
>>> help(mimodulo)
Help on module mimodulo:

NAME
    mimodulo

FUNCTIONS
    saluda(nombre='desconocido')
        Funcion que muestra un mensaje por pantalla al
        'nombre' que se pasa como parametro.

DATA
    dato = 3

FILE
    /home/cesar/mimodulo.py
```

Módulo

Copia de módulo en `sys.path`

La tercera posibilidad de hacer nuestro módulo accesible a Python es copiarlo en la ruta de búsqueda predefinida. Para ello,

1. Listamos los directorios de la *ruta de búsqueda* dada por la variable `sys.path`.
2. Copiamos nuestro módulo en cualquier directorio dado por `sys.path`.

Debemos tener en cuenta que las modificaciones a nuestro módulo debemos hacerlas sobre el fichero que está en la *ruta de búsqueda*.

```
>>> import sys
>>> sys.path
['', '/usr/lib/python36.zip', '/usr/lib/python3.6',
'/usr/lib/python3.6/lib-dynload',
'/usr/local/lib/python3.6/dist-packages',
'/usr/lib/python3/dist-packages']
```

Módulo

Uso

Posibilidades:

1. Abro el módulo, copio todo su contenido al portapapeles y lo pego en mi nuevo programa.
2. Cargo en mi fichero todo el contenido del módulo mediante el uso de la instrucción
   `import nombreModulo`
3. Cargo parcialmente los contenidos de aquellos módulos que me interesan para mi programa. Hago uso de la instrucción
   `from nombreModulo import variable/función [as alias]`

Módulo

Copia de módulo en script

La copia de un módulo en otro script es un procedimiento sencillo, pero adolece de una serie de problemas que lo desaconsejan:

- Puedo equivocarme al copiar por fallar al seleccionar lo que quiero, o por no incluirlo todo.
- Mis programas siguen siendo enormes (muchas líneas).
- Puedo sobrescribir variables que uso en el nuevo código.
- No puedo hacer un seguimiento de las modificaciones útiles que realizo (¿En qué fichero hice aquellos cambios...?)

Módulo

Importación completa

Haciendo uso de la instrucción

```
import nombreModulo
```

cargo todo el contenido del módulo en mi nuevo script o en la consola.
Como inconvenientes:

- Puede que el módulo ocupe mucho (en KB). Efecto creciente a medida que importe más y más módulos.
- Puede que los nombres de variables dentro de los módulos sean muy largos

  `modulo.clase.nombrevariable`

Como ventajas:

- Sintaxis sencilla.
- No tengo que conocer con relativa profundidad el módulo.

Módulo

Importación completa

Otra posibilidad se da mediante el uso de la instrucción

```
from nombreModulo import *
```

Carga todo el módulo en mi nuevo script o en consola.
Como inconvenientes (además de los anteriores):

- Aumentamos las posibilidades de colisión de nombres.

Como ventajas (además de las anteriores):

- Simplificación del espacio de nombres.
  Ya no es necesaria la sintaxis `nombreModulo.variable` bastaría con `variable`.

```
>>> import mimodulo
>>> mimodulo.dato
3
>>> from mimodulo import *
>>> dato
3
```

Módulo

Importación parcial

Haciendo uso de la instrucción

```
from nombreModulo import variable
```

cargo sólo lo que voy a usar del módulo en mi nuevo script o en la consola.
Como inconvenientes:

- Necesito conocer el contenido del módulo.
- Varias sentencias de importación.
- Posibilidad de *colisión de nombres*.

Como ventajas:

- Programas óptimos a nivel de tamaño y simplicidad.
- Simplificación de los nombres de variables y métodos. Se accede a ellos mediante la llamada a variable o función.

Módulo

Ejemplos de importación

```
>>> # Importacion total del modulo
>>> import mimodulo
>>> mimodulo.saluda("Ana")
Hola Ana
>>> dato
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'dato' is not defined
>>> mimodulo.dato
3
>>> dato = 5 # defino variable
>>> # importacion parcial de una variable
>>> from mimodulo import dato
>>> dato # colision de nombres
3
>>> # importacion parcial usando "alias"
>>> from mimodulo import saluda as s
>>> s("Ana")
Hola Ana
```

Módulo

Recarga

Cuando desarrollamos módulos, es habitual probarlos desde consola. Cuando corregimos o mejoramos el código del módulo, debemos actualizar la importación para que esos cambios sean accesibles. Para eso, tenemos que:

1. Importar un módulo `importlib`.
2. Ejecutar la función `reload` que recibe como parámetro el nombre de módulo a recargar.

```
>>> import importlib
>>> importlib.reload(mimodulo)
<module 'mimodulo' from '/home/cesar/mimodulo.py'>
```

Módulo

Obtener información: Posibilidades

1. Abrir y leer el código fuente.
   - Poco práctico.
   - Tarea tan complicada como lo sea el módulo a utilizar.
   - Requiere de cierto dominio de programación en Python, si hemos de interpretar un código.
   - Problemas si la documentación del módulo no es buena.
2. Usar funciones Python
   - `help(nombreModulo)`
     Toda la documentación disponible sobre el módulo.
   - `help(nombreModulo.variable/función)`
     Documentación disponible sobre esa variable o función del módulo.
   - `dir(nombreModulo)`
     Lista de variables y funciones del módulo.
   - `nombreModulo.__dict__`

     Diccionario que tiene por claves, las dadas por la función `dir` y por valores, los asignados a esas claves.

Módulo

Ejemplo de información de módulo

Para poder usar las funciones de documentación, es imprescindible haber cargado el módulo.

```
>>> import mimodulo
>>> help(mimodulo)
>>> help(mimodulo.saluda)
>>> dir(mimodulo)
>>> mimodulo.__dict__
```

La carga parcial del módulo no impide consultar la documentación de la parte cargada.

```
>>> from mimodulo import saluda
>>> help(saluda)
>>> from mimodulo import saluda as s
>>> help(s)
```

Ejercicios propuestos

Módulos

3. Importe el módulo del `Ejercicio1` en el siguiente código, de forma que las instrucciones siguientes se ejecuten sin producir errores:

```
a, b = 13, 3
print 'Operandos␣=', a, b
print 'sumar␣=␣', operaciones_basicas.sumar(a, b)
print 'restar␣=␣', operaciones_basicas.restar(a, b)
```

¿Cómo deberiá importar el módulo para que no hubiera errores al ejecutar?

```
a, b = 13, 3
print 'Operandos␣=', a, b
print 'multiplicar␣=␣', ob.multiplicar(a, b)
print 'dividir␣=␣', ob.dividir(a, b)
```

¿Y cómo importariá para que funcionara el siguiente código?

```
a, b = 13, 0
print 'Operandos␣=', a, b
print 'sumar␣=␣', sumar(a, b)
print 'dividir␣=␣', dividir(a, b)
```

Ejercicios propuestos

Módulos

4. Muestre la documentación completa del módulo operaciones básicas desde el intérprete de python.
5. Cree un nuevo directorio en su directorio de usuario llamado `modulos`. Copie el modulo `operaciones_basicas.py` en dicho directorio. Haga lo necesario, en su código o en el sistema, para que se ejecute el apartado (a) del `Ejercicio 3` desde cualquier otro directorio que no sea modulos.
6. Realice las operaciones necesarias para que el directorio modulos del `Ejercicio 5` figure de forma permanente en la ruta de búsqueda de módulos de Python.

Modulo

Código ejecutable

- Cuando se carga un módulo, **todo su código es ejecutado**.
- Las definiciones de variables producen su inicialización.
- Las definiciones de variables hacen que ese código se ejecute cuando se llamen.
- Lo normal es importar un módulo para acceder a las funciones y variables que contiene. No suele ser habitual la ejecución de instrucciones que contenga.

Modulo

Código ejecutable

- Si queremos separar declaración de variables y funciones del código ejecutable, usamos la variable interna `__name__`.
- La parte ejecutable debería escribirse tras la instrucción

```
if __name__ == "__main__":
        # codigo ejecutable
```

- El código que está dentro del `__main__` es lo que normalmente se ejecuta cuando lanzamos un script de python desde consola, en la forma `python nombreScript.py`

Modulo

Paso de parámetros

Del mismo modo que se pasan parámetros a una función, se pueden pasar a un script.

- El código del módulo necesita importar el módulo `sys`.
- Todos los parámetros pasados se almacenan en la variable `argv` del módulo `sys`. Es una **lista**.
- El **primer elemento** de `sys.argv` es el nombre del programa, esto es, el del módulo. Los siguientes son los parámetros pasados en la llamada.
- Todos los elementos de la lista de parámetros son **cadenas de caracteres** (*string*).

Módulo

Ejemplo de paso de parametros

```
import sys

dato = 3

def saluda(nombre='desconocido'):
    """Funcion que muestra un mensaje por
    pantalla al 'nombre' que se pasa como parametro."""
    print("Hola", nombre)

if __name__ == "__main__":
    print("mimodulo.py se ha cargado correctamente.")
    print("Variable sys.argv = ",sys.argv)
    cont = 0
    for v in sys.argv:
        print("sys.argv[ %(count)d] = %(value)s -->" %
        {"count": cont, "value": v}, type(v))
        cont += 1;
    # Llamada a la funcion con paso de parametro del modulo
    saluda(sys.argv[1])
```

Módulo

Ejemplos de paso de parametros

Ejemplo 1

```
cesar@totoro:~$ python3 mimodulo.py Ana 1 "[2, 4, 8]"
mimodulo.py se ha cargado correctamente.
Variable sys.argv =  ['mimodulo.py', 'Ana', '1', '[2, 4, 8]']
sys.argv[0] = mimodulo.py --> <class 'str'>
sys.argv[1] = Ana --> <class 'str'>
sys.argv[2] = 1 --> <class 'str'>
sys.argv[3] = [2, 4, 8] --> <class 'str'>
Hola Ana
```

Ejemplo 2

```
cesar@totoro:~$ python3 mimodulo.py
mimodulo.py se ha cargado correctamente.
Variable sys.argv =  ['mimodulo.py']
sys.argv[0] = mimodulo.py --> <class 'str'>
Traceback (most recent call last):
  File "mimodulo.py", line 17, in <module>
    saluda(sys.argv[1])
IndexError: list index out of range
```

Ejercicios propuestos

Módulos

7. Modifique el modulo `operaciones_basicas.py`. Cuando ejecute la instrucción en la consola

   ```
   python operaciones_basicas.py numero1 numero2
   ```

   donde `numero1` y `numero2` son dos números cualesquiera (enteros o decimales), deberá mostrar por pantalla el resultado de la suma, resta, multiplicación y división (si esta última es posible) de `numero1` y `número2`.

8. Modifique el ejercicio anterior de la siguiente forma:
   - Si el número de parametros es 1, muestre el siguiente mensaje por pantalla `ERROR: Necesito más de un número`.
   - Si el número de parámetros es igual a dos, realice todas las operaciones con ambos operandos. Muestre el resultado en pantalla.
   - Si se pasan 3 o más parámetros, realice todas las operaciones de forma iterativa[2] y muestre el resultado por pantalla.

[2] `operando1 * operando2 * ... * operandoN` (como ejemplo para el producto)