Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

martes, 4 de febrero de 2014

Módulos

Existen varios métodos para crear módulos pero la forma más sencilla es crear un archivo con extensión .**py** que contenga un conjunto de funciones y/o de clases. Un módulo puede ser importado por otro programa para hacer uso de su funcionalidad, de igual forma que se hace con aquellos módulos que pertenecen a la biblioteca estándar de Python.

La carga de un módulo compilado es más rápida. Los archivos compilados tienen la extensión .**pyc** y tienen la ventaja de que son independientes de la plataforma.

Ejemplo de uso del modulo **sys** que ofrece funciones específicas para interactuar con el sistema:

```
import sys

print('Las rutas de PYTHONPATH:', sys.path, '\n')
```

Namespaces y Alias

Cuando se hace referencia a algún elemento que pertenece a un módulo importado se indica previamente su espacio de nombres (**namespaces**) seguido de un punto "." como en el ejemplo anterior: **sys.path**.

En el momento de importar un módulo puede asignarse también un alias con **as**:

```
# Carga módulo megamodulo.py y asigna alias:
import megamodulo as mmod

# Se hace referencia a una función del módulo:
mmod.mfuncion
```

De un módulo es posible importar sólo aquella función que vamos a utilizar y así cada vez que la usemos evitamos hacer referencia al nombre del módulo. No se suele recomendar este uso porque puede crear conflictos con variables o con funciones que tengan nombres iguales en otros módulos importados en una misma aplicación.

```
# Se carla la función pi del módulo math
from math import pi

# Este uso no se recomienda:
print('El valor de pi', pi)  # es mejor: math.pi
```

Mediante el atributo **__name__** tenemos acceso al nombre de un módulo. Si accedemos a este atributo cuando se ejecuta un programa podemos conocer si el módulo es ejecutado de forma independiente (en ese caso __name__ = '__main__') o es importado:

```
if __name__ == '__main__':
    print("Este programa es independiente")
else:
    print("El modulo ha sido importado")
```

En Geany cuando se comienza a escribir un programa a partir de una plantilla Python (Menú Archivo, Nuevo desde plantilla), en el código inicial se incluye la función **main()** para situar dentro de ella nuestro código.

Creación y uso de un módulo

En el siguiente ejemplo se crea el módulo **moduloseries.py** y un programa **programa.py** lo importa y hace uso de la función **sumadesde1aN** (que para un número n, suma todo los

Buscar

Buscar

Python para impacientes

Python
IPython
EasyGUI
Tkinter
JupyterLab
Numpy

Anexos

Guía urgente de MySQL
Guía rápida de SQLite3

Entradas + populares

Dar color a las salidas en la consola
En Python para dar color a las salidas en la consola (o en la terminal de texto) existen varias posibilidades. Hay un método basado ...

Instalación de Python, paso a paso
Instalación de Python 3.6 A finales de 2016 se produjo el lanzamiento de Python 3.6 . El propósito de esta entrada es mostrar, pas...

Añadir, consultar, modificar y suprimir elementos en Numpy
Acceder a los elementos de un array. [], [,], ... Acceder a un elemento de un array. Para acceder a un elemento se utiliz...

Variables de control en Tkinter
Variables de control Las variables de control son objetos especiales que se asocian a los widgets para almacenar sus valore...

Cálculo con arrays Numpy
Numpy ofrece todo lo necesario para obtener un buen rendimiento cuando se trata de hacer cálculos con arrays. Por como está concebid...

Tkinter: interfaces gráficas en Python
Introducción Con Python hay muchas posibilidades para programar una interfaz gráfica de usuario (GUI) pero Tkinter es fácil d...

Operaciones con fechas y horas. Calendarios
Los módulos datetime y calendar amplían las posibilidades del módulo time que provee funciones para manipular expresiones de ti...

Convertir, copiar, ordenar, unir y dividir arrays Numpy
Esta entrada trata sobre algunos métodos que se utilizan en Numpy para convertir listas en arrays y viceversa; para copiar arrays d...

Tkinter: Tipos de ventanas
Ventanas de aplicación y de diálogos En la entrada anterior tratamos los distintos gestores de geometría que se utilizan para di...

El módulo random
El módulo random de la librería estándar de Python incluye un conjunto de funciones

números que van desde 1 a n).

moduloseries.py:

```
def sumadesde1aN(pnumero):
    ''' Suma todos los números consecutivos desde 1 hasta
    el número expresado. Si el valor es menor que 1 devuelve 0 '''
    ntotal= 0
    if pnumero >0:
        for nnum in range(1,pnumero+1):
            ntotal = ntotal + nnum

    return ntotal

__version__ = '1.0'
```

programa.py:

```
import moduloseries

valor = 10
print('Suma desde 1 a '+str(valor)+':',
      moduloseries.sumadesde1aN(valor))
print('Versión:', moduloseries.__version__)
print('Nombre:', moduloseries.__name__)
print('Doc:', moduloseries.sumadesde1aN.__doc__)
```

También sería posible importar la función y la versión del módulo, así:

```
from moduloseries import sumadesde1aN, __version__

# Importa las funciones pero no la __version__
from moduloseries import *
```

La función dir()

La función **dir**() lista los identificadores que definen un objeto. Para un módulo, los identificadores incluyen las funciones, las clases y variables definidas en ese módulo.

```
import sys
dir(sys)
```

Resultado:

['__displayhook__', '__doc__', '__excepthook__', '__name__', '__package__', '__stderr__', '__stdin__', '__stdout__', '_clear_type_cache', '_current_frames', '_getframe', '_mercurial', '_xoptions', 'abiflags', 'api_version', 'argv', 'builtin_module_names', 'byteorder', 'call_tracing', 'callstats', 'copyright', 'displayhook', 'dont_write_bytecode', 'exc_info', 'excepthook', 'exec_prefix', 'executable', 'exit', 'flags', 'float_info', 'float_repr_style', 'getcheckinterval', 'getdefaultencoding', 'getdlopenflags', 'getfilesystemencoding', 'getprofile', 'getrecursionlimit', 'getrefcount', 'getsizeof', 'getswitchinterval', 'gettrace', 'hash_info', 'hexversion', 'int_info', 'intern', 'last_traceback', 'last_type', 'last_value', 'maxsize', 'maxunicode', 'meta_path', 'modules', 'path', 'path_hooks', 'path_importer_cache', 'platform', 'prefix', 'ps1', 'ps2', 'setcheckinterval', 'setdlopenflags', 'setprofile', 'setrecursionlimit', 'setswitchinterval', 'settrace', 'stderr', 'stdin', 'stdout', 'subversion', 'version', 'version_info', 'warnoptions']

Paquetes (Packages)

Las variables suelen ir dentro de las funciones, las llamadas variables globales dentro de los módulos. Los módulos suelen organizarse en un tipo de carpetas especiales que se llaman **paquetes**. Dentro de estas carpetas deben existir necesariamente un archivo llamado **__init__.py**, aunque esté vacío. No es obligatorio que todos los módulos pertenezcan a un paquete.

La estructura básica de las carpetas de un paquete se parece a:

- <carpeta definida en sys.path>/

- mismodulos/
  - **__init__.py**
  - modulo1/
    - __init__.py
  - modulo2/
    - __init__.py

que permiten obtener de distintos modos números a...

Archivo

febrero 2014 (17)

python.org

pypi.org

Sitios

- ActivePython
- Anaconda
- Bpython
- Django
- Flask
- Ipython
- IronPython
- Matplotlib
- MicroPython
- Numpy
- Pandas
- Pillow
- PortablePython
- PyBrain
- PyCharm
- PyDev
- PyGame
- Pypi
- PyPy
- Pyramid
- Python.org
- PyTorch
- SciPy.org
- Spyder
- Tensorflow
- TurboGears

```
 - codpyton21.py
- modulo3/
 - __init__.py
 - modulo31/
  - __init__.py
  - codpython311.py
```

Un **paquete** puede contener a otros **subpaquetes** y éstos, también, **módulos**. Cuando se importa un módulo es posible indicar su espacio de nombres:

```
# Importa módulo no empaquetado:
import modulo

# Importa módulo del paquete indicado:
import paquete.modulo

# importa módulo del subpaquete/paquete:
import paquete.subpaquete.modulo
```

Ir al índice del tutorial de Python

Publicado por Pherkad en 10:17

Etiquetas: Python3

Entrada más reciente Inicio Entrada antigua