

Módulos & Paquetes

Antonio Espín Herranz

Módulos

- Python permite organizar las aplicaciones mediante módulos y en un nivel superior los paquetes.
- Un módulo puede contener clases, funciones y código.
- Se guarda en un fichero con extensión py.
- Un paquete almacenaría varios módulos.
 - El módulo \leftrightarrow fichero.
 - El paquete $\leftarrow \rightarrow$ directorio.

Módulos

- Ver el contenido de un módulo:
`print (dir(os))` # os es un módulo de python
- Se puede aplicar a objetos:
`print (dir(list))` # Devuelve una lista con los métodos de list
- Obtener ayuda de un módulo u objeto:
Ayuda sobre el módulo **random**:
`print (help(random))`

Ayuda sobre el objeto **list**:
`print (help(list))`

Ayuda sobre un método de un objeto:
`print(help(list.copy))`

Módulos

- Ejemplo: modulo.py

```
def mi_funcion():  
    print ("una funcion")
```

```
class MiClase:  
    def __init__(self):  
        print ("una clase")
```

```
print ("un modulo")
```

Módulos

- Cuando queremos utilizar un módulo hay que importarlo.
- **import mi_modulo.py**
 - Tiene que estar accesible físicamente para que el intérprete lo encuentre.
 - Para utilizar algo del módulo:
 - `modulo.mi_funcion()`
 - **OJO**, el módulo al importarlo también se ejecuta. La sentencia `print` del ejemplo anterior se mostraría simplemente con importarlo.

Módulos

- En el mismo import se pueden indicar varios import:
- Ejemplo:

```
import os, sys, time  
print time.asctime() # Thu May 21 08:53:43 2015
```
- **Otra forma** de importar un módulo:

```
from time import asctime  
print asctime()  
# En este caso NO es necesario poner el nombre del módulo.
```
- No se considera una buena práctica pero se puede hacer:

```
from time import *
```

Módulos

- La variable de entorno **PYTHONPATH** busca el nombre del archivo en los directorios contenidos en dicha variable.
- El valor de la variable PYTHONPATH se puede consultar desde Python mediante **sys.path**

```
>>> import sys
```

```
>>> sys.path
```

Módulos

- En Python **los módulos** también **son objetos**; de **tipo module** en concreto.
- Como cualquier objeto: pueden tener atributos y métodos.
- Uno de sus atributos, **`__name__`**, se utiliza a menudo para incluir código ejecutable en un módulo pero que este sólo se ejecute si se llama al módulo como programa, y no al importarlo.
- Para lograr esto basta saber que cuando se ejecuta el módulo directamente **`__name__`** tiene como valor **`"__main__"`**, mientras que cuando se importa, el valor de **`__name__`** es el nombre del módulo:

Funciones en módulos

- **sys**

- Hace referencia al sistema.
- Se puede importar de una forma similar a math.
- Tenemos las constantes: version y maxint.

```
import sys  
sys.version  
sys.maxint
```

- Funciones matemáticas:

- $\sin(x)$, $\cos(x)$, $\tan(x)$, $\exp(x)$, $\text{ceil}(x)$, $\text{floor}(x)$, $\log(x)$, $\log_{10}(x)$, $\text{sqrt}(x)$.

- Constantes matemáticas:

- Número e y pi.

Paquetes

- Los **módulos** sirven para **organizar** el **código**, los paquetes sirven para organizar los módulos.
- Los **paquetes** son tipos especiales de módulos (ambos son de tipo module) que **permiten agrupar módulos relacionados**.
- A nivel físico:
 - Los **módulos** se corresponden con los **archivos**.
 - Los **paquetes** se representan mediante **directorios**.

Módulos

- Ejemplo

```
print ("Se muestra siempre")
```

```
if __name__ == "__main__":
```

```
    print ("Se muestra si no es importación")
```

- Dependiendo como llamemos al módulo “en ejecución” o en un import, saldrá o no el mensaje.

Documentación en Módulos

- El atributo `__doc__`, que, como en el caso de funciones y clases, sirve a modo de documentación del objeto (docstring o cadena de documentación).
- Su valor es el de la primera línea del cuerpo del módulo, en el caso de que esta sea una cadena de texto; en caso contrario valdrá **None**.

Paquetes

- Para que Python trate a un directorio como un paquete es necesario crear un archivo **`__init__.py`** en dicha carpeta.
- En este archivo se pueden **definir elementos que pertenezcan a dicho paquete**, como una constante DRIVER para el paquete bbdd, **aunque habitualmente se tratará de un archivo vacío**.
- Para hacer que un cierto **módulo** se encuentre **dentro de un paquete**, basta con **copiar el archivo** que define el módulo **al directorio del paquete**.

Paquetes

- Como los módulos, para importar paquetes también se utiliza **import** y **from-import** y el caracter . para separar paquetes, subpaquetes y módulos.
- `import paq.subpaq.modulo`
- `paq.subpaq.modulo.func()`
- Referencia a módulos python en la red:
 - <http://pypi.python.org/>

Ejemplo

- Estructura de un paquete:
 - Ejemplo para importar: **import sound.effects.echo**

sound/	Paquete superior
__init__.py	Inicializa el paquete de sonido
formats/	Subpaquete para conversiones de formato
__init__.py	
wavread.py	
wavwrite.py	
aiffread.py	
aiffwrite.py	
auread.py	
auwrite.py	
...	
effects/	Subpaquete para efectos de sonido
__init__.py	
echo.py	
surround.py	
reverse.py	
...	
filters/	Subpaquete para filtros
__init__.py	
equalizer.py	
vocoder.py	
karaoke.py	
...	

Otra forma de importar

- Cuando queremos acceder a otro módulo que no está en la jerarquía de paquetes.

```
import sys
```

Concatenar la ruta hasta la carpeta que contiene el módulo que queremos importar.

```
sys.path.append(“ruta_relativa o absoluta”)
```

```
from modulo import funcion
```


Importaciones Relativas

- Cuando tenemos una estructura de carpetas similar a esta:

App

```
principal.py
modulos /
    __init__.py
    BBDD/
        __init__.py
        CajeroBBDD.py
    excepciones/
        __init__.py
        CajeroException.py
    objetos/
        __init__.py
        Cajero.py
        Cuentas.py
```

OJO, con el **IDLE** daba problemas la importación con **Geany NO.**

- Desde el principal.py podemos hacer:
from modulos.BBDD.CajeroBBDD import *
- Desde Cajero.py podemos acceder al paquete hermano: excepciones
from ..excepciones.CajeroException import *