Investigación sobre el Manejo de Archivos con Python

Existen dos formas de ejecutar código Python. Podemos escribir líneas de código en el intérprete y obtener una respuesta del intérprete para cada línea (sesión interactiva) o bien podemos escribir el código de un programa en un archivo de texto y ejecutarlo, generando archivos .pyc o .pyo (bytecode optimizado).

Ejecutar este programa es tan sencillo como indicarle el nombre del archivo a ejecutar al intérprete de Python.

Para que el sistema operativo abra el archivo .py con el intérprete adecuado, es necesario añadir una nueva línea al principio del archivo:

```
#!/usr/bin/python
print "Hola Mundo"
raw_input()
```

A esta línea se le conoce en el mundo Unix como shebang, hashbang o sharpbang. El par de caracteres #! indica al sistema operativo que dicho script se debe ejecutar utilizando el intérprete especificado.

Otra opción es utilizar el programa env (de environment, entorno) para preguntar al sistema por la ruta al intérprete de Python, de forma que nuestros usuarios no tengan ningún problema si se diera el caso de que el programa no estuviera instalado en dicha ruta:

```
#!/usr/bin/env python
print "Hola Mundo"
raw_input()
```

También podríamos correr el programa desde la consola como si tratara de un ejecutable cualquiera:

```
./hola.py
```

Módulos:

Para facilitar el mantenimiento y la lectura los programas demasiado largos pueden dividirse en módulos, agrupando elementos relacionados. Los módulos son entidades que permiten una organización y división lógica de nuestro código, cada archivo Python almacenado en disco equivale a un módulo.

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

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

print "un modulo"
```

Si quisiéramos utilizar la funcionalidad definida en este módulo en nuestro programa tendríamos que importarlo. Para importar un módulo se utiliza la palabra clave import seguida del nombre del módulo, que consiste en el nombre del archivo menos la extensión.

```
import modulo

modulo.mi_funcion()
```

Los imports se colocan siempre en la parte superior del archivo, justo después de cualquier comentario o cadena de documentación del módulo, y antes de las variables globales y las constantes del módulo.

Los imports deberían agruparse siguiendo el siguiente orden:

1. imports de la librería estándar
2. imports de proyectos de terceras partes relacionados
3. imports de aplicaciones locales/imports específicos de la librería

Deberías añadir una línea en blanco después de cada grupo de imports.

Si es necesario especificar los nombres públicos definidos por el módulo con __all__ esto debería hacerse después de los imports.

Paquetes:

Si 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. Mientras los módulos se corresponden a nivel físico con los archivos, los paquetes se representan mediante directorios.

Para hacer 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.

Archivos:

Los ficheros en Python son objetos de tipo file creados mediante la función open (abrir). Esta función toma como parámetros una cadena con la ruta al fichero a abrir, que puede ser relativa o absoluta; una cadena opcional indicando el modo de acceso (si no se especifica se accede en modo lectura) y, por último, un entero opcional para especificar un tamaño de buffer distinto del utilizado por defecto.

El modo de acceso puede ser cualquier combinación lógica de los siguientes modos:

• 'r': read, lectura. Abre el archivo en modo lectura. El archivo tiene que existir previamente, en caso contrario se lanzará una excepción de tipo IOError.

• 'w': write, escritura. Abre el archivo en modo escritura. Si el archivo no existe se crea. Si existe, sobreescribe el contenido.

• ‘a’: append, añadir. Abre el archivo en modo escritura. Se diferencia del modo ‘w’ en que en este caso no se sobreescribe el contenido del archivo, sino que se comienza a escribir al final del archivo. • ‘b’: binary, binario.

• ‘+’: permite lectura y escritura simultáneas.

• ‘U’: universal newline, saltos de línea universales. Permite trabajar con archivos que tengan un formato para los saltos de línea que no coincide con el de la plataforma actual (en Windows se utiliza el caracter CR LF, en Unix LF y en Mac OS CR).

```
f = open("archivo.txt", "w")
```

Bibliografía

González Duque, R. *Python para todos.* España.

van Rossum, G., & Warsaw, B. (10 de 08 de 2007). *Guía de estilo del código Python*. Obtenido de http://mundogeek.net/traducciones/guia-estilo-python.htm