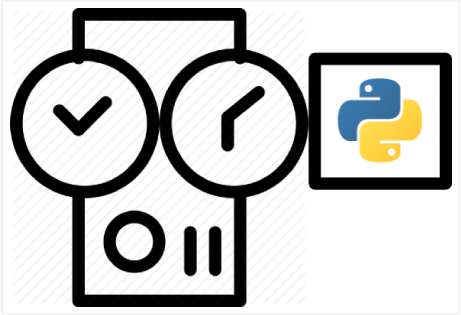
Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

martes, 6 de enero de 2015

Archivos de configuración



El módulo ConfigParser

Es frecuente que un programa tenga que leer su configuración inicial de un archivo para establecer ciertos aspectos de su funcionamiento. Para implementarlo en un programa Python podemos usar el módulo **ConfigParser** que está incluido en la librería estándar. Este módulo se utiliza para manejar los archivos de configuración con una estructura muy parecida a los archivos .INI; permitiendo a los usuarios finales personalizar los programas con facilidad.

La forma de tratar la información de estos archivos tiene mucha similitud con el manejo de los **diccionarios**. En un archivo de configuración de este tipo la información estará organizada en secciones y, dentro de ellas, podrán existir una o más opciones con sus correspondientes valores (o sin ellos). Un ejemplo válido de archivo de configuración podría ser el siguiente:

Nombre del archivo: *juego.cfg*

Contenido del archivo:

[General]
difucultad = baja
jugadores = 1

[Vehiculo]
fabricante = Ferrari
modelo = Testarossa

[Circuito]
nombre = Jerez
clima = lluvioso
obstaculos = True
longitud = 4423

Crear secciones y opciones de configuración

En el siguiente ejemplo se muestran distintas formas de crear las secciones con sus opciones y valores.

```
import configparser
configuracion = configparser.ConfigParser()

# Modo 1: Se añade la sección 'General', asignando directamente
# un diccionario con las opciones y los valores (en una línea).

configuracion['General'] = {'dificultad' : 'baja' , 'jugadores' : '1'}

# Modo 2: Se añade la sección 'Vehiculo' asignando un diccionario
# vacío y agregando con posterioridad las opciones por separado.
```

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...

Threading: programación con hilos (I)
En programación, la técnica que permite que una aplicación ejecute

```
configuracion['Vehiculo'] = {}
configuracion['Vehiculo']['fabricante'] = 'Ferrari'
configuracion['Vehiculo']['modelo'] = 'Testarossa'

# Modo 3: Se instancia un nuevo objeto sección 'Circuito' asignando un
# diccionario vació y después se agregan al objeto opciones y valores.

configuracion['Circuito'] = {}
circuito = configuracion['Circuito']
circuito['nombre'] = 'Jerez'
circuito['clima'] = 'lluvioso'
circuito['obstaculos'] = 'True'
circuito['longitud'] = '4423'
```

Guardar en un archivo la configuración

```
with open('juego.cfg', 'w') as archivoconfig:
    configuracion.write(archivoconfig)
```

Leer datos del archivo de configuración

Las opciones y valores de un archivo de configuración son siempre de tipo cadena (str). Si necesitamos trabajar con valores numéricos utilizaremos los métodos **getint()** y **getfloat()** para evaluar el tipo de número de la opción. Con **int()** y **float()** podremos convertir las cadenas a números enteros o de coma flotante, respectivamente. Por otro lado para evaluar valores lógicos utilizaremos el método **getboolean()**.

A continuación, se lee el archivo de configuración utilizado en los ejemplos anteriores y se muestra cómo acceder a la información.

```
import configparser
configuracion = configparser.ConfigParser()

# Leer el archivo de configuración:

configuracion.read('juego.cfg')

# Mostrar lista con las secciones leidas en el archivo:

configuracion.sections()
# ['General', 'Vehiculo', 'Circuito']

# Otra forma de mostrar las secciones:

for seccion in configuracion.sections():
    print(seccion)
# General Vehiculo Circuito

# Comprobar si una sección está en la configuración:

if 'General' in configuracion:
    print("La sección existe")

# Mostrar el valor de una opción de una determinada sección:

print(configuracion['General']['dificultad'])  # 'baja'

# Mostrar de otro modo el valor de una opción:

general = configuracion['General']
print(general['dificultad'])  # 'baja'

# Mostrar con get() el valor de una opción:

coche = configuracion['Vehiculo']
coche.get('modelo')  # 'Testarossa'

# Leer el valor de una opción y convertir a entero:

juegan = int(configuracion['General']['jugadores']  # 1

# Comprobar si el dato de una opción es de tipo booleano:

if configuracion.getboolean('Circuito', 'obstaculos'):
    print("Aparecerán obstáculos sorpresa en carrera")

# Listar opciones de una sección determinada:
```

simultáneamente varias operaciones en el mismo espacio de proceso se...

Archivo

enero 2015 (2)

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

```
for opcion in configuracion['Vehiculo']:
    print(opcion)    # fabricante, modelo

# Listar opciones y valores de una sección:

for opcion, valor in configuracion['Circuito'].items():
    print (opcion, ":", valor)
    # nombre : Jerez, clima : lluvioso,
    # obstaculos : True, longitud : 4423
```

La sección DEFAULT

La sección **DEFAULT** es una sección especial que mantiene opciones que pueden existir o no en otras secciones. Si una opción de DEFAULT no existe en otras secciones se podrá referenciar como si perteneciera a cualquiera de esas secciones. Es más, si mostramos la lista de opciones de esas secciones comprobaremos que las opciones de DEFAULT también aparecen. En cambio, si una opción de DEFAULT existe en otra sección, cuando se referencie la de esa sección prevalecerá su valor.

En el ejemplo siguiente se lee el archivo de configuración utilizado con anterioridad. Después, se añade la sección **DEFAULT**, se exploran los valores y para finalizar se muestran algunos métodos que pueden facilitarlos la tarea con los archivos de configuración: **defaults()**, **has_section()**, **add_section()**, **options()**, **has_option()**, **set()**, **remove_option()** y **remove_section()**.

```
# Leer el archivo de configuración

import configparser
configuracion = configparser.ConfigParser()
configuracion.read('juego.cfg')

# Añadir sección DEFAULT

configuracion['DEFAULT']['dificultad'] = 'alta'
configuracion['DEFAULT']['jugadores'] = '2'
configuracion['DEFAULT']['sonido'] = 'True'

# Mostrar opciones y sus valores de DEFAULT

for opcion, valor in configuracion['DEFAULT'].items():
    print (opcion, ":", valor)
    # dificultad : alta, jugadores : 2, sonido : True

# Mostrar opciones y sus valores de General
# Como las opciones 'dificultad' y 'jugadores' existen
# en la sección General prevalecen sus valores

for opcion, valor in configuracion['General'].items():
    print (opcion, ":", valor)
    # dificultad : baja, jugadores : 1, sonido : True

# Mostrar el valor de una opción que no existe en la
# sección que se referencia pero sí existe en DEFAULT

print(configuracion['General']['sonido'])
# Toma el valor de DEFAULT: 'True'

# Añadir opcion nueva a 'General' con un valor distinto
# al que tiene en DEFAULT

configuracion['General']['sonido'] = "False"
# Se añade clave a sección General

# Mostrar el valor de una opción que existe en la
# sección que se referencia y también en DEFAULT

print(configuracion['General']['sonido'])
# Toma el valor de General: 'False'

# Obtener el mismo valor con get():

generales = configuracion['General']
generales.get("sonido")    # 'False'

# Mostrar diccionario con opciones y valores por defecto:

configuracion.defaults()
```

```
# OrderedDict([('dificultad', 'alta'), ('jugadores', '2'),
#              ('sonido', 'True')])

# Comprobar si una sección existe con has_section():

if configuracion.has_section('Piloto'):
    print("La sección ya existe")

# Añadir sección con el método add_section() y
# añadir nueva opción:

configuracion.add_section('Piloto')
configuracion.sections()
# ['General', 'Vehiculo', 'Circuito', 'Piloto']

configuracion['Piloto']['nombre'] = 'Alonso'
for clave, valor in configuracion['Piloto'].items():
    print (clave, ":", valor)
    # nombre : Alonso, dificultad : alta,
    # jugadores : 2, sonido : True

# Otro modo de mostrar opciones de una sección, options():

configuracion.options("Piloto")
# ['nombre', 'dificultad', 'jugadores', 'sonido']

# Mostrar si existe una opción en una sección
# con has_option().

if configuracion.has_option('Piloto','nombre'):
    print('La opción en la sección indicada existe')

# Asignar el valor a una opción con set().
# Si la sección dada existe establece la opción con el
# valor especificado; de lo contrario se producirá una
# excepción (NoSectionError). Tanto la opción como su valor
# deben ser cadenas o se producirá un error de tipo (TypeError).

configuracion.set("Piloto", "nombre", "Massa")

# Borrar una opción de una sección con remove_option()
# Si la clave no existe devuelve False y si la sección
# no existe se producirá un error (NoSectionError).

configuracion.remove_option("Motor")

# Borrar una sección con remove_section()

configuracion.remove_section("Piloto")
```

Si utiliza el módulo EasyGUI le podría interesar: Egstore: archivos de configuración con EasyGUI

Ir al Índice del tutorial de Python

Publicado por Pherkad en 5:10

Etiquetas: Python3

Entrada más reciente Inicio Entrada antigua