Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

domingo, 2 de febrero de 2014

Operaciones con archivos

Un archivo es información identificada con un nombre que puede ser almacenada de manera permanente en el directorio de un dispositivo.

Abrir archivo

Antes de poder realizar cualquier operación de lectura/escritura hay que abrir el archivo con **open**() indicando su ubicación y nombre seguido, opcionalmente, por el modo o tipo de operación a realizar y la codificación que tendrá el archivo. Si no se indica el tipo de operación el archivo se abrirá en modo de lectura y si se omite la codificación se utilizará la codificación actual del sistema. Si no existe la ruta del archivo o se intenta abrir para lectura un archivo inexistente se producirá una excepción del tipo **IOerror**.

```
ObjArchivo = open('/home/archivo.txt')
ObjArchivo = open('/home/archivo.txt', 'r')
ObjArchivo = open('/home/archivo.txt',
                  mode='r', encoding='utf-8')
```

¿Y qué codificación utiliza nuestro sistema? Podemos averiguarlo ejecutando las siguientes líneas de código:

```
import locale
print(locale.getpreferredencoding())
```

Las operaciones que pueden realizarse sobre un archivo:

r	**Lectura**
r+	**Lectura/Escritura**
w	**Sobreescritura. Si no existe archivo se creará**
a	**Añadir. Escribe al final del archivo**
b	**Binario**
+	**Permite lectura/escritura simultánea**
U	**Salto de línea universal: win cr+lf, linux lf y mac cr**
rb	**Lectura binaria**
wb	**Sobreescritura binaria**
r+b	**Lectura/Escritura binaria**

Cerrar archivo

Después de terminar de trabajar con un archivo lo cerraremos con el método **close**().

```
ObjArchivo.close()
```

Leer archivo: read, readline, readlines, with-as

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

Con el método **read**() es posible leer un número de bytes determinados. Si no se indica número se leerá todo lo que reste o si se alcanzó el final de fichero devolverá una cadena vacía.

```
# Abre archivo en modo lectura
archivo = open('archivo.txt','r')

# Lee los 9 primeros bytes
cadena1 = archivo.read(9)

# Lee la información restaste
cadena2 = archivo.read()

# Muestra la primera lectura
print(cadena1)

# Muestra la segunda lectura
print(cadena2)

# Cierra el archivo
archivo.close()
```

El método **readline**() lee de un archivo una línea completa

```
# Abre archivo en modo lectura
archivo = open('archivo.txt','r')

# inicia bucle infinito para leer línea a línea
while True:
    linea = archivo.readline()  # lee línea
    if not linea:
        break  # Si no hay más se rompe bucle
    print(linea)  # Muestra la línea leída
archivo.close()  # Cierra archivo
```

El método **readlines**() lee todas las líneas de un archivo como una lista. Si se indica el parámetro de tamaño leerá esa cantidad de bytes del archivo y lo necesario hasta completar la última linea.

```
# Abre archivo en modo lectura
archivo = open('archivo.txt','r')

# Lee todas la líneas y asigna a lista
lista = archivo.readlines()

# Inicializa un contador
numlin = 0

# Recorre todas los elementos de la lista
for linea in lista:
    # incrementa en 1 el contador
    numlin += 1
    # muestra contador y elemento (línea)
    print(numlin, linea)

archivo.close()  # cierra archivo
```

with-as permite usar los archivos de forma óptima cerrándolos y liberando la memoria al concluir el proceso de lectura.

```
# abre archivo (y cierra cuando termine lectura)
with open("indice.txt") as fichero:
    # recorre línea a línea el archivo
    for linea in fichero:
        # muestra línea última leída
        print(linea)
```

Escribir en archivo: write, writelines

El método **write**() escribe una cadena y el método **writelines**() escribe una lista a un archivo. Si en el momento de escribir el archivo no existe se creará uno nuevo.

```
cadena1 = 'Datos'  # declara cadena1
cadena2 = 'Secretos'  # declara cadena2
```

que permiten obtener de distintos modos números a...

```
# Abre archivo para escribir
archivo = open('datos1.txt','w')

# Escribe cadena1 añadiendo salto de línea
archivo.write(cadena1 + '\n')

# Escribe cadena2 en archivo
archivo.write(cadena2)

# cierra archivo
archivo.close()


# Declara lista
lista = ['lunes', 'martes', 'miercoles', 'jueves', 'viernes']

# Abre archivo en modo escritura
archivo = open('datos2.txt','w')

# Escribe toda la lista en el archivo
archivo.writelines(lista)

# Cierra archivo
archivo.close()
```

Mover el puntero: seek(), tell()

El método **seek**() desplaza el puntero a una posición del archivo y el método **tell**() devuelve la posición del puntero en un momento dado (en bytes).

```
# Abre archivo en modo lectura
archivo = open('datos2.txt','r')

# Mueve puntero al quinto byte
archivo.seek(5)

# lee los siguientes 5 bytes
cadena1 = archivo.read(5)

# Muestra cadena
print(cadena1)

# Muestra posición del puntero
print(archivo.tell())

# Cierra archivo
archivo.close()
```

Leer y escribir cualquier objeto a un archivo: pickle

Para leer y escribir cualquier tipo de objeto Python podemos importar el modulo **pickle** y usar sus métodos **dump**() y **load**() para leer y escribir los datos.

```
# Importa módulo pickle
import pickle

# Declara lista
lista = ['Perl', 'Python', 'Ruby']

# Abre archivo binario para escribir
archivo = open('lenguajes.dat', 'wb')

# Escribe lista en archivo
pickle.dump(lista, archivo)

# Cierra archivo
archivo.close()

# Borra de memoria la lista
del lista

# Abre archivo binario para leer
archivo = open('lenguajes.dat', 'rb')

# carga lista desde archivo
lista = pickle.load(archivo)
```

```
# Muestra lista
print(lista)

# Cierra archivo
archivo.close()
```

Relacionado: Tempfile: archivos y directorios temporales

Ir al índice del tutorial de Python

Publicado por Pherkad en 7:07

Etiquetas: Python3

Entrada más reciente Inicio Entrada antigua