Ficheros

Hasta ahora hemos visto como interactuar con un programa a través del teclado (entrada de datos) y la terminal (salida), pero en la mayor parte de las aplicaciones reales tendremos que leer y escribir datos en ficheros.

Al utilizar ficheros para guardar los datos estos perdurarán tras la ejecución del programa, pudiendo ser consultados o utilizados más tarde.

Las operaciones más habituales con ficheros son:

- Crear un fichero.
- Escribir datos en un fichero.
- Leer datos de un fichero.
- Borrar un fichero.

Creación y escritura de ficheros

Para crear un fichero nuevo se utiliza la siguiente función:

- `open(ruta, 'w')` : Crea el fichero con la ruta `ruta`, lo abre en modo escritura (el argumento 'w' significa *write*) y devuelve un objeto que lo referencia.

Si el fichero indicado por la ruta ya existe en el sistema, se reemplazará por el nuevo.

Una vez creado el fichero, para escribir datos en él se utiliza el siguiente método:

- `f.write(c)` : Escribe la cadena `c` en el fichero referenciado por `f` y devuelve el número de caracteres escritos.

```
>>> f = open('saludo.txt', 'w')
>>> f.write('¡Bienvenido a Python!')
21
```

Añadir datos a un fichero

Si en lugar de crear un fichero nuevo queremos añadir datos a un fichero existente se debe utilizar la siguiente función:

- `open(ruta, 'a')` : Abre el fichero con la ruta `ruta` en modo añadir (el argumento 'a' significa *append*) y devuelve un objeto que lo referencia.

Una vez abierto el fichero, se utiliza el método de escritura anterior y los datos se añaden al final del fichero.

```
>>> f = open('saludo.txt', 'a')
>>> f.write('\n¡Hasta pronto!')
15
```

Leer datos de un fichero

Para abrir un fichero en modo lectura se utiliza la siguiente función:

- `open(ruta, 'r')` : Abre el fichero con la ruta `ruta` en modo lectura (el argumento 'r' significa *read*) y devuelve un objeto que lo referencia.

Una vez abierto el fichero, se puede leer todo el contenido del fichero o se puede leer línea a línea. Para ello se utilizan las siguientes funciones:

- `f.read()` : Devuelve todos los datos contenidos en el fichero referenciado por `f` como una cadena de caracteres.
- `f.readlines()` : Devuelve una lista de cadenas de caracteres donde cada cadena es una linea del fichero referenciado por `f`.

```
>>> f = open('saludo.txt', 'r')
>>> print(f.read())
¡Bienvenido a Python!
¡Hasta pronto!

>>> f = open('saludo.txt', 'r')
>>> lineas = f.readlines()
>>> print(lineas)
['Bienvenido a Python!\n', '¡Hasta pronto!']
```

Cerrar un fichero

Para cerrar un fichero se utiliza el siguiente método:

`f.close()` : Cierra el fichero referenciado por el objeto `f`.

Cuando se termina de trabajar con un fichero conviene cerrarlo, sobre todo si se abre en modo escritura, ya que mientras está abierto en este modo no se puede abrir por otra aplicación. Si no se cierra explícitamente un fichero, Python intentará cerrarlo cuando estime que ya no se va a usar más.

```
>>> f = open('saludo.txt'):
>>> print(f.read())
¡Bienvenido a Python!
¡Hasta pronto!
>>> f.close()  # Cierre del fichero
>>> print(f.read())  # Produce un error
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: I/O operation on closed file.
```

La estructura `with open(...) as`

Para despreocuparnos del cierre de un fichero cuando ya no es necesario y no tener que cerrarlo explícitamente, se utiliza la siguiente estructura:

```
with open(ruta, modo) as f:
    bloque código
```

Esta estructura abre el fichero con la ruta `ruta` en el modo `modo` (`'w'` para escribir, `'a'` para añadir y `'r'` para leer) y devuelve una referencia al mismo en la variable `f`. El fichero permanece abierto mientras se ejecuta el bloque de código asociado y se cierra automáticamente cuando termina la ejecución del bloque.

```
>>> with open('saludo.txt', 'w') as f:
...     f.write("Hola de nuevo")
...
13
>>> with open('saludo.txt', 'r') as f:
...     print(f.read())
...
Hola de nuevo
>>> print(f.read())  # Produce un error al estar el fichero cerrado
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: I/O operation on closed file.
```

Renombrado y borrado de un fichero

Para renombra o borrar un fichero se utilizan funciones del módulo `os`.

`os.rename(ruta1, ruta2)` : Renombra y mueve el fichero de la ruta `ruta1` a la ruta `ruta2`.

`os.remove(ruta)` : Borra el fichero de la ruta `ruta`.

Antes de borrar o renombra un directorio conviene comprobar que existe para que no se produzca un error. Para ello se utiliza la función

`os.path.isfile(ruta)` : Devuelve `True` si existe un fichero en la ruta `ruta` y `False` en caso contrario.

Renombrado y borrado de un fichero o directorio

```
>>> import os
>>> f = 'saludo.txt'
>>> if os.path.isfile(f):
...     os.rename(f, 'bienvenida.txt') # renombrado
... else:
...     print('¡El fichero', f, 'no existe!')
...
>>> f = 'bienvenida.txt'
>>> if os.path.isfile(f):
...     os.remove(f) # borrado
... else:
...     print('¡El fichero', f, 'no existe!')
...
```

Creación, cambio y eliminación de directorios

Para trabajar con directorios también se utilizan funciones del módulo `os`.

`os.listdir(ruta)` : Devuelve una lista con los ficheros y directiorios contenidos en la ruta `ruta`.

`os.mkdir(ruta)` : Crea un nuevo directorio en la ruta `ruta`.

`os.chdir(ruta)` : Cambia el directorio actual al indicado por la ruta `ruta`.

`os.getcwd()` : Devuelve una cadena con la ruta del directorio actual.

`os.rmdir(ruta)` : Borra el directorio de la ruta `ruta`, siempre y cuando esté vacío.

Leer un fichero de internet

Para leer un fichero de internet hay que utilizar la función `urlopen` del módulo `urllib.request`.

`urlopen(url)` : Abre el fichero con la `url` especificada y devuelve un objeto del tipo fichero al que se puede acceder con los métodos de lectura de ficheros anteriores.

```
>>> from urllib import request
>>> f = request.urlopen('http://sisek.com.ar/Python-ej/archivo00.txt')
>>> datos = f.read()
>>> print(datos.decode('utf-8'))

===========================
Curso: programacion para NO programadores

Este es un archivo para el curso en el CSI

Pergamino

===========================
(fin del archivo)
```

Control de errores mediante excepciones

Python utiliza un objeto especial llamado **excepción** para controlar cualquier error que pueda ocurrir durante la ejecución de un programa.

Cuando ocurre un error durante la ejecución de un programa, Python crea una excepción. Si no se controla esta excepción la ejecución del programa se detiene y se muestra el error (*traceback*).

```
>>> print(1 / 0)  # Error al intentar dividir por 0.
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

Tipos de excepciones

Los principales excepciones definidas en Python son:

- `TypeError` : Ocurre cuando se aplica una operación o función a un dato del tipo inapropiado.
- `ZeroDivisionError` : Ocurre cuando se itenta dividir por cero.
- `OverflowError` : Ocurre cuando un cálculo excede el límite para un tipo de dato numérico.
- `IndexError` : Ocurre cuando se intenta acceder a una secuencia con un índice que no existe.
- `KeyError` : Ocurre cuando se intenta acceder a un diccionario con una clave que no existe.
- `FileNotFoundError` : Ocurre cuando se intenta acceder a un fichero que no existe en la ruta indicada.
- `ImportError` : Ocurre cuando falla la importación de un módulo.

Consultar la documentaciónde Python para ver la lista de exepciones predefinidas.

Control de excepciones

`try - except - else`

Para evitar la interrución de la ejecución del programa cuando se produce un error, es posible controlar la exepción que se genera con la siguiente instrucción:

```
try:
    bloque código 1
except excepción:
    bloque código 2
else:
    bloque código 3
```

Esta instrucción ejecuta el primer bloque de código y si se produce un error que genera una excepción del tipo *excepción* entonces ejecuta el segundo bloque de código, mientras que si no se produce ningún error, se ejecuta el tercer bloque de código.

Control de excepciones

```
>>> def division(a, b):
...     try:
...         result = a / b
...     except ZeroDivisionError:
...         print('¡No se puede dividir por cero!')
...     else:
...         print(result)
...
>>> division(1, 0)
¡No se puede dividir por cero!
>>> division(1, 2)
```

```
0.5

>>> try:
...     f = open('fichero.txt')  # El fichero no existe
... except FileNotFoundError:
...     print('¡El fichero no existe!')
... else:
...     print(f.read())
¡El fichero no existe!
```