GOBIERNO DE ESPAÑA
MINISTERIO DE HACIENDA Y FUNCIÓN PÚBLICA
INAP
INSTITUTO NACIONAL DE ADMINISTRACIÓN PÚBLICA

Introducción a la programación en Python

Tema 10. Entrada/Salida y Ficheros

Autor: Antonio Villena Martín

Contenido

1. Entrada y salida

1.1. Introducción

Con Python se puede interactuar con el usuario a través de los comandos `print` e `input`. Con `print` se muestran mensajes en pantalla y con `input` se solicita información al usuario.

Con estas dos operaciones se logra una interacción con los usuarios a partir de las aplicaciones que se desarrollen con Python.

En primer lugar, se va a ver la entrada de datos y posteriormente cómo mostrar información a los usuarios.

1.2. Entrada estándar

La función `input()` pausa el programa y espera a que el usuario introduzca algún texto. Una vez que Python recibe la entrada del usuario, la almacena en una variable y posteriormente hacer lo conveniente para que se pueda trabajar con la información capturada.

Esta función es la forma más sencilla de obtener información por parte del usuario. Por ejemplo, el siguiente fragmento de código solicita al usuario que escriba algo de texto, después muestra el mensaje por pantalla:

```
mensaje = input("¿Cómo te llamas? ")
print("Hola, me llamo", mensaje)
```

Si se necesita un entero como entrada en lugar de una cadena, habría que usar la función `int()` para convertir la cadena a entero. Para tratar adecuadamente este procedimiento, sería conveniente usar el control de excepciones para evitar que el programa se interrumpa:

```
try:
    edad = input("¿Cuántos años tienes?")
    dias = int(edad) * 365
    print("Has vivido " + str(dias) + " días")
except ValueError:
    print("El valor introducido no es un número")
```

1.3. Parámetros de línea de comando

Python cuenta con otros métodos para obtener datos del usuario. Como en otros lenguajes de programación como Java o C++, en Python también se dispone del uso de parámetros a la hora de llamar al programa en la línea de comandos. Por ejemplo:

```
python editor.py hola.txt
```

En este caso `hola.txt` sería el parámetro, al que se puede acceder a través de la lista `sys.argv`, aunque antes de poder usar esta variable hay que importar el módulo `sys`.

`sys.argv[0]` → contiene el valor del nombre del programa tal como lo ha ejecutado el usuario.

`sys.argv[1]` → en caso de existir, sería el primer parámetro y así sucesivamente.

```
import sys


if(len(sys.argv)>1):
    print("Abriendo " + sys.argv[1])
else:
    print("Debes indicar el nombre del archivo")
```

1.4. Salida estándar

Con el comando **`print()`** se muestra información por pantalla a los usuarios. Por ejemplo:

```
print("¡Hola Mundo!")
```

Muestra por consola:

```
¡Hola Mundo!
```

Después de imprimir la cadena pasada como argumento, el puntero pasa a la siguiente línea de la pantalla, de un modo igual que `println` en Java.

A continuación, se muestran los caracteres especiales:

Carácter especial	Función
\”	Comilla doble
\’	Comilla simple
\n	Salto de línea
\t	Tabulador

Por ejemplo:

```
print("Hola \nmundo")
```

Imprime:

```
Hola

mundo
```

Se pueden poner diferentes cadenas, separadas mediante comas como en el siguiente ejemplo:

```
for i in range(3):

    print(i, "Hola", "Mundo")
```

Imprimirá:

```
0 Hola Mundo

1 Hola Mundo

2 Hola Mundo
```

A diferencia del uso del operador `+` para concatenar las cadenas, la concatenación de cadenas mediante comas, introduce de manera automática un espacio entre cada una de ellas. Por ejemplo, modificando el anterior ejemplo:

```
for i in range(3):

    print(str(i) + "Hola"+"Mundo")
```

Imprime:

```
0HolaMundo

1HolaMundo

2HolaMundo
```

Como vemos en el anterior ejemplo, al usar el operador +, hay que convertir antes cada argumento que no sea *string*, como por ejemplo el entero, ya que si no, saltaría una excepción de tipo como vemos en el siguiente fragmento:

```
for i in range(3):
    print(i + "Hola"+"Mundo")
```

```
Traceback (most recent call last):

  File  "D:\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py",  line 3437, in run_code

    exec(code_obj, self.user_global_ns, self.user_ns)

  File "<ipython-input-14-b9164afe9d16>", line 2, in <module>

    print(i + "Hola"+"Mundo")

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

1.5. Formateo con print()

La sentencia `print` permite el uso de técnicas de formateo. Por ejemplo:

```
print("Hola %s" %"mundo")
```

Este fragmento de código introduce los valores a la derecha del símbolo `%` (la cadena "mundo") en las posiciones indicadas por los especificadores de conversión de la cadena a la izquierda del símbolo `%`, tras convertirlo al tipo adecuado.

Se pueden pasar varios valores mediante una lista:

```
print("Hola %s, qué tal %s" %("mundo", "andas"))
```

Otro ejemplo:

```
name = "Juan"

age = 23

print("%s tiene %d años." % (name, age))
```

En la siguiente tabla, se muestran los **especificadores** más comunes:

Especificador	Formato
%s	Cadena
%d	Entero
%f	Real
%o	Octal
%x	Hexadecimal

En el caso de los **reales**, se puede indicar la precisión a usar precediendo la '**f**' de un punto seguido del número de decimales que querremos mostrar.

```
import math

print('%.4f' %math.pi)
```

Imprime:

```
3.1416
```

Se puede indicar el número mínimo de caracteres que se quiere que ocupe la cadena, introduciendo un número entre **%** y el carácter que indica el tipo al que formatear. Si el tamaño de la cadena resultante es menor que este número, se añadirán espacios a la izquierda de la cadena hasta completar la cifra. En el caso de que el número sea negativo, los espacios se añadirán a la derecha de la cadena. Ejemplo:

```
print("%10s %s" %("Hola", "mundo"))
```

Imprime:

```
      Hola mundo
```

```
print("%-10s %s" %("Hola", "mundo"))
```

Imprime:

```
Hola       mundo
```

También se puede usar para indicar el número de caracteres de la cadena que se quiere mostrar:

```
print("%.3s" %"Hola mundo")
```

Imprime:

```
Hol
```

Hay un método más sofisticado de formatear el texto mediante la sentencia `print`, y es el de comenzar una cadena con `f` o `F` antes de las comillas. Dentro de la cadena, se pueden escribir expresiones entre los caracteres de llaves `{` y `}` que pueden referir a variables o valores literales.

```
name = "Juan"
age = 23
print(f"{name} tiene {age} años.")
```

Imprime:

```
Juan tiene 23 años.
```

1.5.1 Literales de cadena formateados

Permite el uso de expresiones en Python dentro de una cadena añadiendo un prefijo a la cadena con `f` o `F` y escribiendo expresiones como `{expresion}`.

Un especificador de formato opcional puede seguir a la expresión. Esto permite un mayor control sobre cómo se formatea el valor. El siguiente ejemplo redondea *pi* a cuatro decimales:

```
import math
print(f'El valor de pi es aproximadamente {math.pi:.4f}.')
```

Imprime:

```
El valor de pi es aproximadamente 3.1416.
```

Pasar un número entero después de `':'` hará que ese campo tenga un número mínimo de caracteres de ancho. Esto es útil para **alinear las columnas**.

```
tabla = {'Manuel': 4127, 'Juan': 4098, 'Pepe': 7678}
for nombre, telefono in tabla.items():
    print(f'{nombre:10} ==> {telefono:10d}')
```

Imprime:

```
Manuel      ==>       4127
Juan        ==>       4098
Pepe        ==>       7678
```

1.5.2 El método `format()` de String

El uso básico del método **`str.format()`** es:

```
print('Nosotros somos los {} que dicen "¡{}!"'.format('caballeros',
'Hola'))
```

Imprime:

```
Nosotros somos los caballeros que dicen "¡Hola!"
```

Los corchetes y caracteres dentro de ellos (llamados campos de formato) se reemplazan con los objetos pasados al método **`str.format()`**. Se puede usar un número entre corchetes para referirse a la posición del objeto pasado al método **`str.format()`**.

```
>> print('{0} y {1}'.format('Pan', 'Huevos'))
Pan y Huevos
```

```
>> print('{1} y {0}'.format('Pan', 'Huevos'))

Huevos y Pan
```

Si se utilizan argumentos de palabras clave en el método `str.format()`, se hace referencia a sus valores mediante el nombre del argumento.

```
print('Este {feed} es {adjetivo}.'.format(feed='spam',
adjetivo='absolutamente horrible'))
```

Imprime:

```
Este spam es absolutamente horrible.
```

Los argumentos posicionales y de palabras clave se pueden combinar arbitrariamente:

```
print('La historia de {0}, {1}, y {other}.'.format('Bill', 'Manfred',
other='Georg'))
```

Imprime:

```
La historia de Bill, Manfred, y Georg.
```

Si se tiene una cadena de formato muy larga que no se desea dividir, sería bueno que pudiera hacer referencia a las variables a formatear por nombre en lugar de por posición. Esto se puede hacer simplemente pasando el diccionario y usando corchetes `'[]'` para acceder a las claves.

```
tabla = {'Eva': 4127, 'Juan': 4098, 'Pepe': 8637678}

print('Juan: {0[Juan]:d}; Eva: {0[Eva]:d}; Pepe:
{0[Pepe]:d}'.format(tabla))
```

Imprime:

```
Juan: 4098; Eva: 4127; Pepe: 8637678
```

Esto también se puede hacer pasando la tabla como argumentos de palabras clave con la notación "`**`".

```
tabla = {'Eva': 4127, 'Juan': 4098, 'Pepe': 8637678}
print('Juan: {Juan:d}; Eva: {Eva:d}; Pepe: {Pepe:d}'.format(**tabla))
```

Imprime:

```
Juan: 4098; Eva: 4127; Pepe: 8637678
```

Como ejemplo, las siguientes líneas producen un conjunto de columnas ordenadamente alineadas que dan números enteros y sus cuadrados y cubos:

```
for x in range(1, 11):
    print('{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*x))
```

Imprime:

```
 1   1    1
 2   4    8
 3   9   27
 4  16   64
 5  25  125
 6  36  216
 7  49  343
 8  64  512
 9  81  729
10 100 1000
```

2 Persistencia con ficheros

Hay dos maneras de trabajar respecto al sistema de archivos y directorios. Uno es a través del módulo ***os***, es decir, desde el propio sistema operativo. El segundo nivel, más sencillo a través de Python, permite trabajar con archivos, manipulando su lectura y escritura a nivel de la aplicación y tratando a cada archivo como un **objeto**, que es el método que se verá en este curso.

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

Los ficheros en Python son objetos de tipo `File` creados mediante la función `open()`. Esta función tiene la siguiente sintaxis:

```
open(file, mode='r', buffering=-1, encoding=None, errors=None,
newline=None, closefd=True, opener=None)
```

Esta función devuelve un objeto que representa el fichero que se ha abierto.

Los 3 primeros parámetros son los más comunes, aunque el único obligatorio es el primero.

El *primer parámetro* (***file***) es una cadena con la ruta al fichero a abrir. Python busca primero en el mismo directorio donde se está ejecutando el programa, en caso que no se haya especificado más que el nombre del fichero.

El *segundo parámetro* (***mode***) es una cadena opcional indicando el modo de acceso (por defecto se accede en modo lectura '`r`'). Otro valor común es '`w`' para escribir (o truncar el fichero si ya existía), '`x`' para creación exclusiva de un fichero y '`a`' para añadir al final del fichero. En modo texto, si la codificación (***encoding***) no se especifica la codificación usada depende de la plataforma, por lo que conviene especificarlo de modo explícito. Los modos disponibles son:

Carácter	Significado
'r'	abrir para lectura (por defecto)
'w'	abrir para escritura, trunca primero el fichero
'x'	abre para creación exclusiva, falla si el fichero ya existía
'a'	abre para escritura, añadiendo al final del fichero si existe
'b'	modo binario
't'	modo texto (por defecto)
'+'	abre un fichero en disco para actualizar (lectura o escritura)
'U'	Modo de nuevas líneas universal (en desuso)

El *tercer parámetro* de tipo entero para especificar la política de ***buffer***.

En la práctica se suelen usar únicamente los dos primeros parámetros.

Ejemplo de uso de `open()`.

```
mi_lista = [i**2 for i in range(1,11)]
# Generá una lista con los números 1 - 10 al cuadrado

f = open("C:/Temp/salida.txt", "w", encoding="utf-8")
# abre el fichero en modo escritura

for item in mi_lista:
f.write(str(item) + "\n")
# escribe cada elemento de la lista en una línea

f.close()
# cierra el fichero
```

Siempre hay que cerrar los archivos cuando se termina de escribir en ellos.

Al terminar de trabajar con un archivo, es MUY recomendable cerrarlo, por diversos motivos. En algunos sistemas los archivos sólo pueden ser abiertos por un programa por la vez; en otros, lo que se haya escrito no se guardará realmente hasta no cerrar el archivo; o el límite de cantidad de archivos que puede manejar un programa puede ser bajo, etc. También puede ocurrir que se produzca un bug durante la ejecución del programa. En este caso, el fichero se podría quedar abierto y nunca se cerraría.

El no cerrar un fichero de manera correcta, podría ocasionar que los datos se corrompiesen, con la consecuente pérdida de información.

Durante el proceso de Entrada/Salida, los datos se **almacenan en un búfer,** esto significa que esos datos se guardan en una ubicación temporal antes de ser escritos en el archivo.

Python no **vacía el búfer**—es decir, no escribe los datos en el archivo— hasta que se asegura de que se terminó de escribir. La manera de hacer esto es **cerrar el archivo**. Si se escribe en un archivo sin cerrarlo bien, los datos no llegarán al archivo de destino.

En Python los objetos tipo `file` tienen dos métodos especiales **para cerrar de manera automática los archivos.** Estos métodos son `with` y `as`. La sintaxis es:

```
with open("archivo", "modo") as variable:
# Lee o escribe en el archivo
```

Por ejemplo:

```
with open("./texto.txt", "w", encoding="utf-8") as textofile:
    textofile.write("¡Éxito!")
```

Para saber si un objeto de tipo `file` se ha cerrado de manera correcta, dicho objeto tiene un atributo `closed` que indica mediante un booleano si está cerrado o no. Por ejemplo:

```
f = open("./bg.txt")
f.closed
# False

f.close()
f.closed
# True
```

2.1 Lectura de ficheros

Para la lectura de archivos se utilizan los métodos `read`, `readline` y `readlines`.

Para leer el contenido de un archivo se ejecuta **`f.read`**`(cantidad)`, el cual lee alguna cantidad de datos y los devuelve como una cadena de (en modo texto) o un objeto de bytes (en modo binario), *cantidad* es un argumento numérico opcional. Cuando se omite *cantidad* o es negativo, el contenido entero del archivo será leído y devuelto. Puede ser un problema si el archivo es muy

grande, ya que puede colapsar la memoria de la máquina. Si se alcanzó el fin del archivo, `f.read()` devolverá una cadena vacía (`""`).

El método `f.readline()` lee una sola línea del archivo; el carácter de fin de línea (`\n`) se deja al final de la cadena, y sólo se omite en la última línea del archivo si el mismo no termina en un fin de línea. Esto hace que el valor de retorno no sea ambiguo. Si `f.readline()` devuelve una cadena vacía, es que se alcanzó el fin del archivo, mientras que una línea en blanco es representada por `'\n'`, una cadena conteniendo sólo un único fin de línea.

```
f.readline()
'Esta es la primer linea del archivo.\n'


f.readline()
'Segunda linea del archivo\n'


f.readline()
''
```

Para leer líneas de un archivo, se puede iterar sobre el objeto archivo. Esto es eficiente en memoria, rápido, y conduce a un código más simple:

```
for linea in f:
        print(linea, end='')


Esta es la primera línea del archivo
Segunda línea del archivo
```

Si se quiere leer todas las líneas de un archivo en una lista también se puede usar `list(f)` o `f.readlines()`.

```
f.readlines()
['Esta es la primera línea del archivo\n', 'Segunda línea del archivo\n']
```

2.2 Puntero de lectura/ escritura

Hay situaciones en las que puede interesar mover el puntero de lectura/escritura a una posición determinada del archivo. Por ejemplo, si se quiere empezar a escribir en una posición determinada y no al final o al principio del archivo.

Para esto se utiliza el método **`seek(desplazamiento, desde_donde)`** que toma como parámetro un número positivo o negativo a utilizar como desplazamiento. También es posible utilizar un segundo parámetro (*desde_donde*) para indicar desde dónde queremos que se haga el desplazamiento: 0 indicará que el desplazamiento se refiere al principio del fichero (comportamiento por defecto), 1 se refiere a la posición actual, y 2, al final del fichero.

Para determinar la posición en la que se encuentra actualmente el puntero se utiliza el método **`tell()`**, que devuelve un entero indicando la distancia en bytes desde el principio del fichero.

2.3 Escritura de ficheros

Para la escritura de archivos se utilizan los métodos **`write`** y **`writelines`**. Mientras el primero funciona escribiendo en el archivo una cadena de texto que toma como parámetro, el segundo toma como parámetro una lista de cadenas de texto indicando las líneas que queremos escribir en el fichero.

`f.write(cadena)` escribe el contenido de la cadena al archivo, devolviendo la cantidad de caracteres escritos.

Ejemplo:

```
with open("texto.txt", "w") as f:

    f.write("Escritura de la primera línea.")
```

2.4 Tabla resumen de operaciones con ficheros objeto File

A continuación, se muestra una tabla resumen, con los métodos más comunes del objeto `File`.

Ejemplo:

```
file = open("<archivo>")
```

Función	Descripción
file.close()	Cierra un fichero. Un fichero cerrado no se puede leero o escribir más.
file.flush()	Limpia el buffer interno
file.fileno()	Devuelve un entero como descriptor del fichero que se usa para implementación subyacente para realizar peticiones de Entrada/Salida desde el sistema operativo.
File.isatty()	Devuelve true si el ficero está conectado a un dispositivo **tty** (teletipo), en caso contrrio false.
file.next()	Devuelve la siguiente línea de un fichero cada vez que es llamada.
file.read([size])	Lee como máximo el tamaño en bytes del fichero (menos si la lectura se encuentra con un EOF antes de llegar al tamaño en bytes especificado).
file.readline([size])	Lee una línea entera de un fichero. Un carácter de nueva línea al final se mantiene en la cadena.
file.readlines([sizehint])	Lee hasta un EOF usando readline() y devuelve una lista conteniendo las líneas. Si el argumento opcional sizehint es especificado, en lugar de leer hasta el EOF, se leen líneas completas que totalizan aproximadamente bytes de tamaño de letra (posiblemente después de redondear hasta un tamaño de búfer interno).
file.seek(offset[, whence])	Establece la posición actual del cursor en el fichero.
file.tell()	Devuelve la posición actual del cursor en el fichero.
file.truncate([size])	Trunca el tamaño del fichero. Si el argumento opcional size esta presente, el fichero se trunca (al menos) a ese tamaño.
file.write(str)	Escribe una cadena de texto al fichero. No se devuelve nada.
file.writelines(sequence)	Escribe una secuencia de cadenas de texto al fichero. La secuencia puede ser cualquier tipo de objeto iterable que produzca cadenas de texto, usualmente una lista de cadenas.
file.rstrip()	Elimina cualquier carácter en blanco del lado derecho de la cadena.
file.lstrip()	Elimina cualquier carácter en blanco del lado izquierdo de la cadena.

2.5 Ejercicios propuestos

Ejercicio 1

En este ejercicio se deberá crear un script llamado **`quijote.py`** que lea los datos del fichero de texto **`quijote.txt`**, que realice un bucle que lo recorra e imprima por consola todas las líneas.

El fichero de texto **`quijote.txt`** se creará previamente y se le añadirán las líneas:

```
En un lugar de la Mancha, de cuyo nombre no quiero acordarme,
no ha mucho tiempo que vivía un hidalgo de los de lanza en astillero,
adarga antigua, rocín flaco y galgo corredor.
```

Habrá que procurar generar el fichero .txt en codificación UTF-8 para, posteriormente, leerlo mediante el script usando el parámetro *encoding="utf-8"*.

La estructura del script será:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-



# TODO: Leer fichero "./curso/datos/quijote.txt"


# TODO: Bucle mostrando todas las líneas
```

Ejercicio 2

En este ejercicio se deberá abrir el fichero **`quijote.txt`** creado en el ejercicio anterior y añadirle las siguientes líneas a continuación de las líneas que ya existían previamente.

```
Una olla de algo más vaca que carnero, salpicón las más noches,
duelos y quebrantos los sábados, lentejas los viernes,
algún palomino de añadidura los domingos, consumían las tres partes de su
hacienda.
```

La estructura del script será:

```
#!/usr/bin/env python

# -*- coding: utf-8 -*-


# fichero a leer 'Quijote.txt'

f = "./curso/datos/quijote.txt"


# TODO: Abre el fichero en modo 'añadir'

""" Escribe las líneas:

 Una olla de algo más vaca que carnero, salpicón las más noches,

 duelos y quebrantos los sábados, lentejas los viernes,

 algún palomino de añadidura los domingos, consumían las tres partes de su hacienda.

"""


# TODO: Vuelve a leer el fichero para comprobar que ha escrito el texto correctamente
```

Ejercicio 3

En este ejercicio se ha de crear un fichero en modo escritura e ir almacenando en variables los datos del alumno. Posteriormente cerrar el fichero. Cada uno de los datos solicitados, se han de ir almacenando en diferentes líneas.

La estructura del script será:

```
#!/usr/bin/env python

# -*- coding: utf-8 -*-


# TODO: crear un fichero en modo escritura e ir almacenando en variables los datos del alumno.
```

```
# nombre, apellidos, ciudad, edad

# finalmente esos datos se escriben en líneas diferentes en el fichero.

# cerrar el fichero.


# Abre el fichero en modo 'escritura'


# pregunta por consola datos y los va introduciendo en variables


# Almacena los datos en líneas diferentes en el fichero
```

3 Solución ejercicios

Solución ejercicio 1

```
#!/usr/bin/env python

# -*- coding: utf-8 -*-




# Leer fichero Quijote.txt

f = open("./curso/datos/quijote.txt", "r")


# Bucle mostrando todas las líneas

for linea in f:

    print(linea)
```

Solución ejercicio 2

```
#!/usr/bin/env python

# -*- coding: utf-8 -*-


# fichero a leer 'Quijote.txt'

f = "./curso/datos/quijote.txt"
```

```
# Abre el fichero en modo 'añadir'
with open(f, 'a') as file:
    """ Escribe las líneas:
        Una olla de algo más vaca que carnero, salpicón las más noches,
        duelos y quebrantos los sábados, lentejas los viernes,
        algún palomino de añadidura los domingos, consumían las tres
partes de su hacienda.
       """
    file.write( '\n' + "Una olla de algo más vaca que carnero, salpicón
las más noches, ")
    file.write( '\n' +"duelos y quebrantos los sábados, lentejas los
viernes, ")
    file.write( '\n' +"algún palomino de añadidura los domingos,
consumían las tres partes de su hacienda.")

# Vuelve a leer el fichero para comprobar que ha escrito el texto
dcorrectamente
with open(f, 'r') as file:
    actual = file.read()
    print(actual)
```

Solución ejercicio 3

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

f = "datos_alumno.txt"

# Abre el fichero en modo 'escritura'
with open(f, 'w') as file:
    # pregunta por consola datos y los va introduciendo en variables
    nombre = input('Nombre: ')
    apellidos = input('Apellidos: ')
    ciudad=input('Ciudad nacimiento: ')
    edad=input('Edad: ')

    # Almacena los datos en lineas diferentes en el fichero
```

```
    file.write( "Nombre: " + nombre)
    file.write( "\nApellidos: " + apellidos)
    file.write( "\nCiudad: " + ciudad)
    file.write( "\nEdad: " + str(edad))

print("Fichero escrito correctamente.")
```