Entradas y salidas de datos

En esta sección aprenderemos sobre las entradas y salidas de información, es decir, las formas de capturar información desde el exterior (entrada) y la visualización de esos datos (salida).

Entrada por teclado

Ya conocemos la función *input()* que lee una cadena por teclado. Su único inconveniente es que debemos transformar el valor a numérico si deseamos hacer operaciones con él:

decimal = float( input("Introduce un número decimal con punto: ") )

Introduce un número decimal con punto: 3.14

Una forma interesante de leer varios valores es almacenar los datos en una lista e irlos añadiendo ahí para poderlos manipular en grupo:

valores = []

print("Introduce 3 valores")

for x in range(3):

valores.append( input("Introduce un valor: ") )

print(valores)

Introduce 3 valores

Introduce un valor: 10

Introduce un valor: sdkjsdk

Introduce un valor: skdjs

['10', 'sdkjsdk', 'skdjs']

Por ahora no hay mucho más que comentar sobre las entradas en tiempo de ejecución. También es posible leer datos de ficheros pero eso es algo que veremos más adelante.

# **Entrada por script**

Hasta ahora todo lo que hemos hecho ha sido escribir código en el intérprete, pero los programas informáticos no funcionan así. Se basan en escribir todas las instrucciones en ficheros llamados scripts (o guiones de instrucciones). Luego se envía este fichero al intérprete desde la terminal (si es un lenguaje interpretado como Python) y éste ejecutará todas las instrucciones en bloque.

A parte de ser la base de los programas, la gracia de los Scripts es que pueden recibir datos desde la propia terminal en el momento de la ejecución, algo muy útil para añadirles dinamismo.

Para poder crear y ejecutar scripts hace falta un editor y una terminal, por suerte Anaconda trae el editor Spyder y el intérprete es accesible a través de la terminal Anaconda Prompt, ambos programas accesibles desde Inicio (o con Anaconda Navigator si utilizáis Linux/MAC).

## **Script de prueba**

Utilizando Spyder deberíais ser capaces de crear un fichero hola.py con este contenido:

print("Hola, bienvenido a tu primer script")

Lo podéis guardar en el directorio que queráis, yo lo guardaré en una carpeta del disco, en C:\Scripts para tenerlo a mano.

A continuación para ejecutarlo abríramos Anaconda Prompt y navegaríamos con el comando cd (change directory) al directorio donde tenemos los scripts:

cd C:\Scripts

Por último ejecutaremos el script pasándolo al intérprete de python y veremos el resultado del print() en la terminal:

python hola.py

Hola, bienvenido a tu primer script

## **Scripts con argumentos**

Para poder enviar información a un script y manejarla, tenemos que utilizar la librería de sistema sys. En ella encontraremos la lista argv que almacena los argumentos enviados al script. Cread el siguiente script y ejecutadlo:

import sys

print(sys.argv)

python argumentos.py

['argumentos.py']

Al ejecutarlo veréis que devuelve una lista con una cadena que contiene el nombre del script. Quedaros con la idea pues, de que el primer argumento de la lista sys.argv (sys.argv[0]) es el propio nombre del script.

Ahora intentad ejecutarlo de nuevo pasando algunos valores como números y cadenas de texto entre comillas dobles, todo separado por espacios:

python argumentos.py 100 "Hola" 3.14

['argumentos.py', '100', 'Hola', '3.14']

Cada valor que enviamos al script durante la llamada se llama argumento e implica una forma de entrada de datos alternativa sin usar el **input()**.

El siguiente ejemplo es un script que recibe dos argumentos: un texto y un número entero. Lo que hace es imprimir la cadena de texto tantas veces como le indiquemos en el número. Tomaos el tiempo necesario para analizar y entender cada línea:

import sys

# Comprobación de seguridad, ejecutar sólo si se reciben 2 argumentos reales

if len(sys.argv) == 3:

texto = sys.argv[1]

repeticiones = int(sys.argv[2])

for r in range(repeticiones):

print(texto)

else:

print("Error - Introduce los argumentos correctamente")

print('Ejemplo: escribir\_lineas.py "Texto" 5')

python escribir\_lineas.py "Hola Mundo!!!" 5

Hola Mundo!!!

Hola Mundo!!!

Hola Mundo!!!

Hola Mundo!!!

Hola Mundo!!!

# **Salida por pantalla**

La función print() es la forma general de mostrar información por pantalla. Generalmente podemos mostrar texto y variables separándolos con comas:

v = "otro texto"

n = 10

print("Un texto",v,"y un número",n)

Un texto otro texto y un número 10

## **El método .format()**

Es una funcionalidad de las cadenas de texto que nos permite formatear información en una cadena (variables o valores literales) cómodamente utilizando identificadores referenciados:

c = "Un texto '{}' y un número '{}'".format(v,n)

print(c)

Un texto 'otro texto' y un número '10'

También podemos referenciar a partir de la posición de los valores utilizando índices:

print( "Un texto '{1}' y un número '{0}'".format(v,n) )

print(c)

Un texto '10' y un número 'otro texto'

O podemos utilizar identificador con una clave y luego pasarlas en el format:

print( "Un texto '{v}' y un número '{n}'".format(n=n,v=v) )

Un texto 'otro texto' y un número '10'

print("{v},{v},{v}".format(v=v))

otro texto,otro texto,otro texto

## **Formateo avanzado**

Este método soporta muchas técnicas de formateo, aquí algunos ejemplos.

Alineamiento a la derecha en 30 caracteres:

print( "{:>30}".format("palabra") )

palabra

Alineamiento a la izquierda en 30 caracteres (crea espacios a la derecha):

print( "{:30}".format("palabra") )

palabra

Alineamiento al centro en 30 caracteres:

print( "{:^30}".format("palabra") )

palabra

Truncamiento a 3 caracteres:

print( "{:.3}".format("palabra") )

pal

Alineamiento a la derecha en 30 caracteres con truncamiento de 3:

print( "{:>30.3}".format("palabra") )

pal

Formateo de números enteros, rellenados con espacios:

print("{:4d}".format(10))

print("{:4d}".format(100))

print("{:4d}".format(1000))

10

100

1000

Formateo de números enteros, rellenados con ceros:

print("{:04d}".format(10))

print("{:04d}".format(100))

print("{:04d}".format(1000))

0010

0100

1000

Formateo de números flotantes, rellenados con espacios:

print("{:7.3f}".format(3.1415926))

print("{:7.3f}".format(153.21))

3.142

153.210

Formateo de números flotantes, rellenados con ceros:

print("{:07.3f}".format(3.1415926))

print("{:07.3f}".format(153.21))

003.142

153.210

## **Format simplificado**

La actualización de Python 3.6 trajo la novedad de poder concatenar variables y cadenas de una forma muy cómoda sin utilizar el format().

Hasta ahora para concadenar hacíamos lo siguiente:

nombre = "Héctor"

texto = "Hola {}".format(nombre)

print(texto)

Hola Héctor

La nueva sintaxis nos permite ahorrarnos el método:

nombre = "Héctor"

texto = f"Hola {nombre}"

print(texto)

Hola Héctor

Sólo tenemos que indicar **f** antes de la cadena y sustituir las variables por sus nombre.