Python

Guía para ser un Pythonista

Juan José Lozano Gómez – j2logo.com

©Juan José Lozano Gómez. Todos los derechos reservados

6 Entrada/Salida

©Juan José Lozano Gómez. Todos los derechos reservados

Índice de contenidos

©Juan José Lozano Gómez. Todos los derechos reservados

Introducción

Terminamos el segundo bloque del curso con el tema de Entrada/Salida de datos.

Este tema es fundamental y muy práctico a la hora de realizar aplicaciones de consola en Python.

En él descubrirás cómo recibir información por parte de un usuario y cómo formatear de manera apropiada el texto en la consola o terminal.

©Juan José Lozano Gómez. Todos los derechos reservados

Entrada de datos

En muchas ocasiones es necesario en una aplicación solicitar datos al usuario que este debe introducir a través del teclado. Para ello, en Python podemos usar la función `input()`.

`input()` es una función que proporciona el lenguaje y su funcionamiento es muy sencillo: Lee una línea de la entrada, la convierte en un `string` (eliminando el carácter `\n`) y la devuelve.

A la función se le puede pasar un texto como argumento. Este texto será utilizado como *prompt*, mostrando el mensaje al usuario.

Cuando se llama a la función `input()` en una aplicación, el flujo de esta se detiene hasta que el usuario introduzca una línea y pulse la tecla `Intro/Enter`.

Veámoslo con un ejemplo:

```
>>> nombre = input('¿Cómo te llamas?')
¿Cómo te llamas?>? j2logo
>>> print(nombre)
j2logo
```


©Juan José Lozano Gómez. Todos los derechos reservados

¡Cuidado cuando utilices esta función con valores numéricos! La función siempre devuelve un `string`.

Si necesitas un valor numérico debes hacer la conversión correspondiente:

```
>>> año = input('¿En qué año naciste?')
¿En qué año naciste?>? 1960
>>> edad = 2020 - año
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: unsupported operand type(s)
for -: 'int' and 'str'
```

```
>>> año = input('¿En qué año naciste?')
¿En qué año naciste?>? 1960
>>> edad = 2020 - int(año)
>>> edad
60
```


©Juan José Lozano Gómez. Todos los derechos reservados

Salida de datos

Ya hemos visto anteriormente cuál es la función para mostrar datos por consola: `print()`

La función `print()` muestra un texto por pantalla:

```
>>> print('¡Hola mundo!')
¡Hola mundo!
```

Esta es la forma más sencilla de usar la función. Pero, en realidad, se le puede pasar cualquier tipo de argumento a la función:

```
>>> nombre = 'J2logo'
>>> print(nombre)
J2logo
```

Y no solo un argumento, sino varios. Cuando esto ocurre, `print()` concatena los diferentes argumentos separándolos con el carácter espacio.

©Juan José Lozano Gómez. Todos los derechos reservados

```
>>> nombre = 'J2logo'
>>> print('Hola', nombre)
Hola J2logo
```

Si se quiere utilizar un texto diferente como separador, se debe indicar el argumento `sep` (es un `string`):

```
>>> nombre = 'J2logo'
>>> print('Hola', nombre, sep="+")
Hola+J2logo
```

Por último, la función `print()` finaliza siempre con el carácter de nueva línea `\n`. Es posible modificar este comportamiento especificando el argumento `end` (es un `string`):

```
>>> print('Hola', end='')
Hola>>> print('Hola', end=' :)')
Hola :)
```

En definitiva, la cadena de salida es la concatenación de todos los argumentos, separados por el valor indicado en `sep` (si no se indica, se utiliza el carácter espacio) y al

©Juan José Lozano Gómez. Todos los derechos reservados

final se concatena el valor indicado por `end` (si no se indica, se utiliza el carácter de nueva línea)

```
>>> nombre = 'J2logo'
>>> print('Hola', nombre, sep='+', end='!')
Hola+J2logo!
```

Dando formato al texto

Una vez que hemos visto cómo obtener datos por teclado con `input()` y cómo mostrar información al usuario con `print()`, te voy a explicar cómo puedes darle formato al texto para mostrar la información de la manera más adecuada posible.

En principio, en Python hay tres formas de dar formato a un texto: Utilizando cadenas formateadas (conocidas como *f-strings*), usando el método `format()` o manualmente con el operador de concatenación de cadenas `+`.

©Juan José Lozano Gómez. Todos los derechos reservados

F-Strings

Aparecieron en la versión 3.6 de Python y es la forma recomendada y más eficiente para dar formato a un texto.

Estas cadenas permiten embeber expresiones completas de la forma `{expresión}` dentro de un string que comienza por el carácter `f`:

```
>>> x = 1
>>> y = 2
>>> res = f'El resultado de {x} + {y} es {x + y}'
>>> print(res)
El resultado de 1 + 2 es 3
```

`{expresión}` puede ser una variable, un literal o, como has visto en el ejemplo anterior, una expresión.

Adicionalmente, a la expresión le puede seguir una directiva de formato para tener un mayor control de cómo los datos son formateados. Para ello se utiliza la siguiente estructura `{expresión:formato}`.

Por ejemplo, para mostrar solo dos cifras decimales de un `float`, se puede usar la siguiente expresión:

©Juan José Lozano Gómez. Todos los derechos reservados

```
>>> pi = 3.1416
>>> print(f'Número PI: {pi:.2f}')
Número PI: 3.14
```

En la sección del final del tema puedes ver una miniguía del lenguaje usado en las directivas de formato.

Método format() de la clase String

Esta manera de formatear cadenas requiere un poco más de esfuerzo manual que la anterior. También usa `{}` para indicar dónde se sustituirá una variable y puede especificar directivas de formato detalladas.

El método `format()` está definido en la clase `String`, por tanto, es accesible por cualquier cadena de caracteres.

A continuación, te explico diferentes formas de uso.

La más básica es especificando con llaves las posiciones donde se incluirán los valores de las variables:

©Juan José Lozano Gómez. Todos los derechos reservados

```
>>> var1 = 'J2logo'
>>> var2 = 'Hola'
>>> print('{} {}, ¿cómo estás?'.format(var2, var1))
Hola J2logo, ¿cómo estás?
```

También se puede especificar el nombre de las variables, de manera que el orden en el que se pasen los argumentos al llamar a `format()` no importa:

```
>>> var1 = 'J2logo'
>>> var2 = 'Hola'
>>> '{v2} {v1}, ¿cómo estás?'.format(v1=var1,
v2=var2)
'Hola J2logo, ¿cómo estás?'
```

Por último, puedes especificar el índice de los argumentos:

```
>>> var1 = 'J2logo'
>>> var2 = 'Hola'
>>> '{1} {0}, ¿cómo estás?'.format(var1, var2)
'Hola J2logo, ¿cómo estás?'
```


©Juan José Lozano Gómez. Todos los derechos reservados

Formato manual

El tercer modo de formatear texto es hacerlo manualmente, utilizando el operador de concatenación de cadenas, `+`, junto con métodos de la clase string (que veremos en el tema dedicado a este tipo de datos).

Siguiendo el último ejemplo anterior, podemos obtener el mismo resultado de la siguiente manera:

```
>>> var1 = 'J2logo'
>>> var2 = 'Hola'
>>> saludo = var2 + ' ' + var1 + ', ¿cómo estás?'
>>> print(saludo)
Hola J2logo, ¿cómo estás?
```

Lleva cuidado cuando trates de concatenar un string con una variable de otro tipo. Como ya sabes, En Python no se pueden realizar operaciones con objetos de distinto tipo. Realiza primero una conversión de la variable a string.

```
>>> edad = 22
>>> print('Julia tiene ' + edad + ' años')
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: can only concatenate str (not "int")
to str
```


©Juan José Lozano Gómez. Todos los derechos reservados

El ejemplo anterior da un error porque no se hizo una conversión de la variable edad a string. El modo correcto sería así:

```
>>> edad = 22
>>> print('Julia tiene ' + str(edad) + ' años')
Julia tiene 22 años
```

Como puedes apreciar, este modo de formatear texto es más engorroso que los anteriores y más proclive a errores.

Directivas de formato

En esta última parte del tema te voy a mostrar algunas de las directivas de formato más comunes que puedes usar con las *f-strings* o con el método `format()`.

Las especificaciones o directivas de formato se utilizan dentro de los campos de reemplazo, justo después de los dos puntos, para definir cómo se presentan los valores individuales.

Los diferentes tipos de datos pueden definir cómo se debe interpretar la especificación de formato. No

©Juan José Lozano Gómez. Todos los derechos reservados

obstante, algunas de las opciones de formato solo son compatibles con los tipos numéricos.

Cuando no se especifica una directiva de formato, se produce el mismo resultado que si hubiera llamado a `str(variable)`. Cuando `variable` pertenece a un tipo de datos básico, ya sabes lo que ocurre. Sin embargo, cuando `variable` es un objeto de otro tipo, por defecto se llama al método `__str__()` del mismo. Veremos esto con más detenimiento en el tema de *Programación Orientada a Objetos*.

Volviendo a las directivas de formato, los usos más comunes son los siguientes:

Alineación y ancho mínimo

```
>>> num1 = 2020
>>> num2 = 100320
>>> print(f'{num1:*>10}')
******2020
>>> print(f'{num2:*>10}')
****100320
>>> print(f'{num2:*^10}')
**100320**
>>> print(f'{num2:^10}')
  100320
```


©Juan José Lozano Gómez. Todos los derechos reservados

Separador de miles

```
>>> num1 = 1239765
>>> print(f'{num1:,}')
1,239,765
>>> print(f'{num1:_}')
1_239_765
```

Precisión de números decimales para floats

```
>>> num = 23.873124
>>> print(f'{num:.3f}')
23.873
```


©Juan José Lozano Gómez. Todos los derechos reservados

Representación de números enteros

```
>>> num = 25

# En binario
>>> print(f'{num:b}')
11001

# En octal
>>> print(f'{num:#o}')
0o31

# En hexadecimal
>>> print(f'{num:#X}')
0X19


>>> num = 0o31 # 25 en octal

# Representación en decimal
print(f'{num:d}')
25
```


©Juan José Lozano Gómez. Todos los derechos reservados

Representación de números float

```
>>> num = 23.87899

# Notación científica usando la letra e
>>> print(f'{num:e}')
2.387899e+01

# Notación científica usando la letra E
>>> print(f'{num:E}')
2.387899E+01

# Notación en punto flotante
# con 6 dígitos decimales por defecto
>>> print(f'{num:f}')
23.878990

# Porcentaje
>>> num = 0.5
>>> print(f'{num:%}')
50.000000%
```


©Juan José Lozano Gómez. Todos los derechos reservados

©Juan José Lozano Gómez. Todos los derechos reservados

j2logo.com - © Todos los derechos reservados