Bottega DevCamp

Full Stack Developer

Checkpoint 3: Ejercicios teóricos

Fecha: 22 de febrero de 2025

Autor: Domínguez Becerril, Jorge DNI: 72607195J

Índice

1. Cuestiones teóricas

1.1. ¿Cuáles son los tipos de datos en Python?

Los principales tipos de datos con los que se puede trabajar en Python son los siguientes:

- **Booleanos**: este tipo de dato solo adopta dos valores, **True** o **False** y resulta especialmente útil en estructuras condicionales de tipo **if**.
- **Números reales** y **enteros**: estos se utilizan para hacer todo tipo de cálculos. Python permite trabajar con distintos tipos de precisión, en función del tipo de cálculos que se deseen llevar a cabo.
- **Cadenas**: son conjuntos de caracteres alfanuméricos y se representan entre comillas simples o dobles. Presentan la característica de inmutabilidad, es decir, no se pueden alterar.
- **Listas**: son conjuntos de cualquier tipo de dato. Se representan por corchetes, separando cada elemento con una coma y son mutables.
- **Tuplas**: este tipo de dato permite almacenar cualquier tipo de datos de manera similar a las listas, pero, a diferencia de estas, son inmutables.
- **Diccionarios**: un diccionario es un conjunto de elementos, donde cada uno tiene una llave key y un valor value. Se representan con llaves, separando con una coma cada par.

La existencia de toda esta variedad de datos hace de Python un lenguaje de programación multidisciplinar, usado tanto para machine learning, finanzas, biología, matemáticas, etc.

1.2. ¿Qué tipo de convención de nomenclatura deberíamos utilizar para las variables en Python?

La nomenclatura que debería utilizarse para las variables en Python o en cualquier otro lenguaje de programación debería seguir las siguientes recomendaciones:

- **Auto-explicativa**: el nombre de la variable debe ser lo más explicativo posible en relación a su contenido. Por ejemplo, una variable que almacene una cadena que contiene un nombre podría llamarse **name** o **nombre**. Una variable que almacene un conjunto de datos podría ser llamada **data_set** o **conjunto_datos**.
- **Uso del guión bajo**: cuando el nombre de la variable es largo, es recomendable el uso del guión bajo para mejorar la legibilidad y evitar errores. Por ejemplo, el nombre **conjuntodatos** se debería sustituir por **conjunto_datos**.
- **Nombres a evitar**: nunca se deben utilizar los caracteres 'l' (letra ele minúscula), 'O' (letra o mayúscula) o 'I' (letra i mayúscula) como nombre de una variable. Esto se debe a que en algunas fuentes de letras estos caracteres son indistinguibles de los números 0 y 1.

Estas recomendaciones permiten una programación más ágil y libre de errores.

1.3. ¿Qué es un Heredoc en Python?

Un Heredoc (abreviatura de here document, que significa aquí documento) es una sintaxis especial que permite definir cadenas de texto multilínea. En Python, los heredoc se definen haciendo uso de las comillas triples. En la figura 1 se muestra un heredoc.

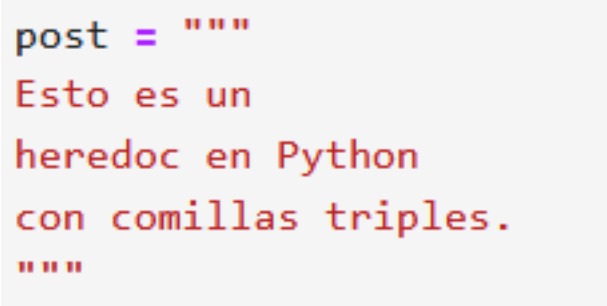

Figura 1: Ejemplo de Heredoc en Python.

1.4. ¿Qué es una interpolación de cadenas?

La interpolación de cadenas en Python permite procesar código de Python dentro de cadenas de texto. Existen diferentes maneras de hacer esto. La más moderna y sencilla es con el uso de cadenas-f. En la figura 2 se muestra un ejemplo de interpolación de cadenas haciendo uso de cadenas-f. Nótese como al imprimir la variable **mensaje** las variables **name** y **age** dentro de la cadena de texto se han sustituido por sus valores.

```
name = 'Jorge'
age = '27'
mensaje = f'Me llamo {name} y tengo {age} años'
print(mensaje)
```

```
Me llamo Jorge y tengo 27 años
```

Figura 2: Ejemplo de interpolación de cadenas.

En la figura 3 se muestra otro ejemplo. Nótese como al imprimir la variable **mensaje** las variables **peras** y **platanos** se han sustituido por sus valores y se ha computado la suma de ambas variables.

```
peras = 7
platanos = 5
mensaje = f'''Tengo {peras} peras y {platanos} platanos,
o {peras + platanos} piezas de fruta'''
print(mensaje)
```

```
Tengo 7 peras y 5 platanos,
o 12 piezas de fruta
```

Figura 3: Otro ejemplo de interpolación de cadenas.

1.5. ¿Cuándo deberíamos usar comentarios en Python?

Los comentarios en Python y, en general, en cualquier lenguaje de programación siempre han sido objeto de acalorado debate entre los desarrolladores. Las recomendaciones sobre cuando comentar el código son las siguientes:

- **Organización del código**: los comentarios resultan útiles para organizar el código. Por ejemplo, si en un archivo de código hay un bloque de funciones que ejecutan una determinada acción, esto se puede indicar mediante un comentario para poder localizarlas fácilmente.
- **Marcar código pendiente o mejoras**: se pueden usar comentarios para indicar partes del código pendientes de desarrollo o que precisan mejoras. De esta manera, resulta más fácil identificarlas.

En ningún caso se deberían utilizar para explicar la funcionalidad del código. Esto se debe a que si se cambia el código pero no el comentario, este resultaría confuso e inútil. Resulta preferible hacer el código lo más autoexplicativo posible.

1.6. ¿Cuáles son las diferencias entre aplicaciones monolíticas y de microservicios?

Los enfoque monolítico y de microservicios son dos formas distintas de estructurar una aplicación. Una aplicación **monolítica** es una aplicación en la que todos los componentes de la misma están integrados en el mismo código base y se ejecutan juntos. Por el contrario, una aplicación de **microservicios** está dividida en componentes más pequeños e independientes que se comunican entre sí a través de APIs (acrónimo de Application Programming Interface). Cada enfoque presenta sus ventajas e inconvenientes.

- **Enfoque monolítico**
  - Ventajas
    - Más fácil de desarrollar, depurar, probar y desplegar
    - La comunicación entre componentes no es una fuente de errores
  - Inconvenientes
    - No es escalable
    - Se vuelve difícil de mantener con el paso del tiempo
    - Cualquier cambio o actualización precisa desplegar la totalidad de la aplicación nuevamente.

- **Enfoque de microservicios**
  - Ventajas
    - Es escalable
    - Es más fácil de mantener
  - Inconvenientes
    - Es más complicado de desarrollar
    - La comunicación entre los distintos componentes de la aplicación puede ser una fuente de errores