Checkpoint 3 – Preguntas teóricas

1. ¿Cuáles son los tipos de Datos en Python?

Los principales tipos de datos en Python son los siguientes:

- String: cadenas de texto que se ponen entre comillas tanto simples como dobles. Un ejemplo sería “Hola mundo”.

- Números. Hay dos tipos de números en Python:
  - Integers: Son números enteros, sin decimales. Por ejemplo 1,2,3…
  - Floats: Son números con decimales. Por ejemplo, tanto 2.5 como 3.00 serían tipos de número float.

- Booleans: Estos son valores o verdadero (True) o falso (False).

- Bytes y bite arrays. Son tipos de datos reservados para desarrollos más complejos en Python.
  - Bytes es un tipo de dato que no se puede cambiar. Son para almacenar datos binarios.
  - Byte arrays, a diferencia de los bytes son mutables, es decir, se pueden cambiar o modificar a posteriori. También se usan para almacenar datos binarios.

- None: Es un tipo de dato especial que representa que no hay valor.

- Listas: Las listas son colecciones ordenadas de distintos tipos de elementos. Es decir, una lista puede contener por ejemplo strings [“Tomates”, “Patatas”], números [1, 2, 3.14] y otros tipos de datos como otras listas, tuplas, diccionarios, etc. En las listas también se pueden combinar los tipos de datos, es decir, no tienen que ser completamente strings o números, pueden contener distintos tipos de datos como por ejemplo [“Tomates”, 3, [1,2,3]].

- Tuples: Estas son colecciones de elementos ordenados, pero a diferencia de las listas, no pueden ser modificados. Se representan entre paréntesis, por ejemplo (1, 2, 3)

- Sets: Estas son colecciones no ordenadas y sus elementos son únicos, no se repiten. En este caso se representan entre llaves {1,2,3}.

- Dictionaries: Los diccionarios son combinaciones de claves con valores. Este tipo de dato se representa entre llaves {“key”: ”value”}. Las claves o keys son únicas. Este tipo de dato también se puede modificar. Un ejemplo de un diccionario:

```
coche = {
        “marca”: “opel”,
        “año_matricula”: 2017,
        “itv_pasada”: True
}
```

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

La documentación de la guía de estilo activa de Python actual es PEP 8. Respecto a las variables, al escribir el nombre de las variables hay que evitar usar o y O ya que se pueden confundir con un 0 y las letras l y L porque se pueden confundir con el número 1. Además, se aconseja que estén en minúscula y separadas por barras bajas (snake_case) para no confundirlas con clases, ya que las clases están en FormatoCapital (CamelCase). Un ejemplo de una variable en formato adecuado sería: mi_variable

3. ¿Qué es un Heredoc en Python?

Un heredoc se le llama en Python a la escritura multilínea. Es decir, escribir un comentario en varias líneas. En Python esto se realiza mediante triples comas que pueden ser dobles triples comas (“ “ “) o triple comas simples (‘ ‘ ‘). Estas se colocan al inicio del comentario y al final. Se utilizan cuando se necesita escribir bloques de texto largos con saltos de línea.

```
"""
Este es un ejemplo de un Heredoc en Python.
Como se puede observar, es un solo comentario que se divide en distintas líneas.
El Heredoc comienza y finaliza con triples comillas. En este caso dobles, pero también
se pueden usar simples.
"""
```

4. ¿Qué es una interpolación de cadenas?

La interpolación de cadenas es cuando se inserta una variable dentro de una string. Hay dos formas de poder realizar esta interpolación, mediante el uso de f antes de la cadena (la más utilizada), o mediante el método de format().

- Usando f-string. Escribiendo f delante de una cadena de texto esto le da un formato al texto donde escribiendo entre las llaves {} el nombre de la variable, en vez de imprimir literalmente “{variable}” imprimiría directamente el valor almacenado en dicha variable. Por ejemplo:

```
nombre = “Pepito”
edad = 25
print(f”Hola, me llamo {nombre} y tengo {edad} años.”)
```

Consola: Hola, me llamo Pepito y tengo 25 años.

- Usando format(). En este caso, también se utilizan las llaves {} para mostrar las variables, pero a diferencia de f-string, no se nombra la variable dentro de éstas. En este caso se usa .format() después de la string y entre paréntesis se escriben por orden las variables que se quieren mostrar. Es decir, la primera variable se mostraría en las primeras llaves, la segunda en las segundas y así sucesivamente. Por ejemplo, usando las variables anteriores:

```
print(”Mi vecino {} tiene {} años.”.format(nombre, edad)
```

Consola: Mi vecino Pepito tiene 25 años.

```
print(“Hace {} años que conozco a {}.”.format(edad, nombre)
```

Consola: Hace 25 años que conozco a Pepito

5. ¿Cuándo deberíamos usar comentarios en Python?

Los comentarios de texto son para que los vean las personas que están trabajando con el código, es decir, son apartados que el código ignora (no se imprimen, no se usan para funcionalidad del código). El código debería ser suficientemente comprensible para las personas que trabajen con él, es decir, las variables, clases, funciones, etcétera deberían tener nombres completamente descriptivos para evitar el uso de los comentarios y añadir demasiadas líneas o texto que se puede evitar y así tener un código más limpio para trabajar. Aun así, en ocasiones es necesario escribir comentarios para explicar bloques de código complejos y así describir su función principal. También se pueden utilizar para documentar decisiones tomadas o incluso dejar alguna nota para ti mismo u otros desarrolladores que trabajan contigo, por ejemplo, si hay algún apartado que se necesita modificar, si es necesario añadir alguna función en un punto concreto del código, etc. Además, también se pueden utilizar para organizar el código cuando es muy largo (aunque esto es más utilizado en otro tipo de lenguajes como CSS). Aun así, como anteriormente he mencionado, lo óptimo es usar la mínima cantidad de comentarios, porque un buen código debe ser descriptivo sin necesidad de explicaciones adicionales.

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

En una aplicación monolítica la arquitectura de la aplicación está toda junta, es decir, está todo en una sola aplicación. Este tipo de arquitectura tiene la ventaja de ser más rápido de desarrollar, ya que se puede componer de funciones más básicas. Además, debido a que está todo en el mismo servicio, son más rápidas que las aplicaciones de microservicios. Aun así, como está todo compactado en una misma aplicación, editarlo es más complicado, ya que al realizar un cambio se pueden modificar otras partes del código creando un efecto dominó, lo cual lo hace más difícil de mantener. Un ejemplo de una aplicación monolítica podría ser por ejemplo una web de un blog personal, donde todo se encontraría en una misma aplicación ya que es una aplicación más simple.

Por lo contrario, una aplicación de microservicios está estructurada en distintos procesos. Es decir, está separada en distintas partes independientes. Cada parte se encarga de una funcionalidad específica. Un ejemplo de esto sería por ejemplo plataformas como Amazon, ya que tiene distintos apartados como el de mostrar los productos, el apartado de pagos, el apartado del usuario etcétera. Estas aplicaciones, al tener bloques independientes son más sencillas de mantener, ya que, si se comete un error en un apartado específico, al ser independiente a los demás, no afectaría negativamente a los demás bloques. Cuando la aplicación está terminada, se ejecutan los procesos de forma separada. Esto permite por ejemplo aumentar los componentes individuales de forma más sencilla, ya que en las aplicaciones monolíticas habría que escalarlo de forma global, pero en este caso se pueden crear componentes individuales que no afectan negativamente al resto en su desarrollo.