Checkpoint nº 3: teoría

1. **¿Cuáles son los tipos de Datos en Python?**
   - Boolean
   - Number
   - String
   - Bytes and byte array
   - None
   - List
   - Tuple
   - Dictionary
   - Set

2. **¿Qué tipo de convención de nomenclatura deberíamos utilizar para las variables en Python?**
   De acuerdo a PEP 8, debemos usar el formato Snake Case: palabras en minúsculas separadas por guiones bajos. Ejemplos:
   - interes_compuesto
   - tasa_de_error

3. **¿Qué es un Heredoc en Python?**
   Un heredoc es un string que tiene varias líneas. Estamos hablando no de un string de pocos caracteres, sino de un string más extenso en el que pueden existir además caracteres no imprimibles como saltos de línea, retornos de carro,…

4. **¿Qué es una interpolación de cadenas?**
   La interpolación de cadenas (o f-string) es una función de Python que permite insertar código Pyton dentro de una cadena de texto. Se usar para generar cadenas de texto con información variable dependiendo de la situación. Esto nos puede servir, por ejemplo, para personalizar mensajes: nuestra aplicación puede, partiendo de una estructura de string común presentar información específica adecuada al contexto. El siguiente código mostraría en pantalla el siguiente mensaje:

   Sr. Recesvinto De Rua.
   A día de hoy, 27-03-2025, su saldo en nuestor banco es de 100.000 euros.

   Código python

```
Tratamiento = 'Sr.'
Nombre = 'Recesvinto'
Apellido = "De Rua"
Fecha = "27-03-2025"
Saldo = "100.000"
print(f'''{Tratamiento} {Nombre} {Apellido}. \nA día de hoy, {Fecha}, su saldo
en nuestro banco es de {Saldo} euros.''')
```

5. **¿Cuándo deberíamos usar comentarios en Python?**

El uso de comentarios en Python debiera restringirse al máximo, ya que, dado que el código suele sufrir modificaciones, un comentario que explicaba una funcionalidad concreta del código asociado, puede quedar obsoleto si el código se modifica y el comentario no.

Esto suele ser bastante habitual, especialmente si la modificación se realiza tiempo después o la hace otro programador: es común no fijarse en el comentario asociado y dejarlo sin actualizar. Acabamos obteniendo comentarios que no tienen nada (o poco) que ver con el código, y que en realidad entorpecen más que ayudan.

El objetivo de un programador es que el código sea auto explicativo: para ello es muy importante la elección correcta de los nombres de los elementos del lenguaje (constantes, variables, clases, objetos, atributos y métodos), de tal modo que con su lectura ya nos indiquen su semántica. Esto no siempre es posible y a veces, porciones de código muy intrincadas necesitan de comentarios, pero hay que ser cuidadosos al sincronizar los cambios en el código y los comentarios asociados.

Los comentarios más generales (como por ejemplo los que puedan describir la funcionalidad principal de un método, una función o la semántica de una clase) sí son aconsejables. Ejemplo:

```
“””
Esta función permite establecer la conexión a la base de
datos MySQL de usuarios de la aplicación
”””
def conexion_bbdd (……):
    …
    return ….
```

Aunque esa función cambie la forma concreta de conectarse, seguirá siendo una función que se encarga de la conexión a la base de datos de usuarios.

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

En una aplicación monolítica toda la funcionalidad perseguida se implementa en un gran archivo de código.

En un microservicio, la funcionalidad completa se descompone en diferentes partes que realizan acciones más sencillas, pero que juntas obtienen la funcionalidad total. Cada una de estas partes se denominan microservicios (por ser más pequeños que el global) y se implementa en un único archivo. Obviamente, cada uno de estos microservicios genera un archivo más pequeño que en una monolítica.

Es por tanto más fácil de implementar y depurar que un único archivo con toda la funcionalidad. Además, un error en alguno de estos microservicios provoca el mal funcionamiento de ése, pero no tiene por qué afectar a toda la aplicación, como sí ocurre en una monolítica.

Otra enorme ventaja es que, si descomponemos una aplicación en microservicios, podemos asignar esos a desarrolladores diferentes, posibilitando el desarrollo colaborativo y de forma paralela. Esto nos permite hacer que el ciclo de desarrollo pueda ser muchísimo más rápido.

Un ejemplo. Queremos desarrollar una aplicación de compra online (B2C). Podemos descomponer esta aplicación es estos microservicios:

- Acceso de usuarios. Se encarga de dar de alta y validar usuarios.
- Carrito de compra. Permite recorrer el ecommerce, añadir y eliminar ítems de un carrito virtual.
- Pagos. Se encarga de los pagos de los clientes.
- Histórico. Se encarga de recuperar el historial de compras de un usuario.
- Lista de deseos. Permite crear listas de ítems que queremos guardar para quizá comprarlos en un futuro.

  (Habrá seguro más microservicios, pero no pretendo ser exhaustivo, solo poner un ejemplo sencillo)

En una aplicación basada en microservicios, si el microservicio que se encarga de 'Lista de deseos' no funcionase por cualquier causa, el cliente podría realizar toda la operativa de compra sin problemas, aunque no pudiese consultar ni añadir ítems a sus listas de deseos.

Si la aplicación fuese monolítica, probablemente no podría realizar ninguna acción.