Ejercicios: Clase 4

1. **Debugging**
    a. Cree un nuevo proyecto el cual pueda ejecutar con un entorno virtual aislado. Puede utilizar tanto `poetry` como `venv` o la herramienta que prefiera.
    b. Dentro de su proyecto, genere un script que contenga una función que reciba uno o más argumentos y produzca algún resultado particular. La función tiene que fallar para ciertos valores del argumento de entrada (por ejemplo, división por cero, raíz de un valor negativo, o puede considerar también tipo de argumentos de entrada incorrectos).
    c. Utilice statements del tipo `assert` para asegurarse que no tiene valores indeseados en algunas de sus variables.
    d. Genere una función adicional que llame a la función que creó anteriormente. Esta función puede ser un decorador. Utilice algún debugger (`pdb`, `pdbpp` o cualquier otro), navegue en las diferentes funciones y en los diferentes niveles de su script, viendo cómo cambia en contexto global y local.
2. **Docstrings y typing**
    a. Agregue a su proyecto el paquete `pydocstyle` y ejecútelo sobre su script.
    b. Escriba los docstrings necesaria para su script. Para esto, elija uno los estilos de documentación posibles y aplíquelo.
    c. Investigue y utilice alguna aplicación para Visual Studio Code (o su IDE de preferencia) que le facilite la tarea de escribir los docstrings.
    d. Ejecute `pydocstyle` sobre su script hasta que este no muestre ninguna corrección pendiente.
    e. Agregue type hints en su script. Utilice el módulo `typing` cuando sea necesario (aclarando argumentos opcionales `Optional`, union entre más de un argumento `Union`, de acuerdo a lo que necesite.
    f. Opcionalmente, investigue y utilice la biblioteca `sphinx` para generar documentación en formato HTML a partir de los docstrings escritos.

g. Utilicé la biblioteca `mypy` o `pyright` para hacer enforcing del tipado especificado.

3. **Manejo de configuraciones**

   a. A partir del script que utilizó (o creando uno nuevo) genere un archivo de configuración con los argumentos que necesita para ejecutarlo, a partir de uno de los métodos vistos en clase.

   b. Opcionalmente, investigue y utilice otros métodos para importar su configuración. Se recomienda importar configuración a partir de un archivo `.yml` utilizando `PyYaml`, o utilizando `ConfigParser` para importar la configuración de un archivo `.ini`.

   c. Opcionalmente, investigue como insertar los argumentos por línea de comandos cuando ejecuta su script.

   d. Trabajando con el módulo `os`, obtenga e imprima, en un script o utilizando el intérprete de Pyhon, las variables de entorno.

4. **Profiling**

   a. Cree un nuevo script donde exista una función que trabaje con un número alto de iteraciones, o que contenga el método `time.sleep()`.

   b. Calcule el tiempo de ejecución de la misma utilizando la función `timeit` del módulo homónimo. Varíe entre corridas la cantidad de iteraciones de la función, o el valor del `sleep`.

   c. Utilice la función `time` del módulo homónimo, llamándola en la primer línea de su función y antes del final de la misma, para obtener e imprimir el tiempo de ejecución a partir de la diferencia de estos valores. Compare el valor obtenido con este método con el que obtiene con el módulo `timeit`.

   d. Opcionalmente, cree un decorador que le permita medir el tiempo de ejecución de funciones a través la función `time`, de manera similar al punto anterior.

   e. Opcionalmente, utilice la función `wraps` del módulo `functools` y utilícela en su decorador. ¿Cómo se modifican los atributos `__name__` y `__doc__` de su función aplicando esta función?

   f. En su script, o en uno nuevo, genere en una función diferentes listas con diferentes valores y de diferentes tamaños, y mida el espacio en memoria de

las mismas a través del decorador `@profile` del módulo `memory_profiler`.

g. Opcionalmente, utilice el comando de line magic `%memit` para el mismo objetivo. Esto deberá realizarlo dentro de una notebook de Jupyter ejecutada dentro de su proyecto (para instalar el kernel de Jupyter deberá ejecutar el siguiente comando: `poetry run python -m ipykernel install --user --name <kernel_name>`).