

## Introducción a la Programación

## Taller de Debugging

## 1. Indicaciones

El objetivo del taller es encontrar bugs en la implementación de los ejercicios utilizando las herramientas vistas. Podrán encontrar los materiales para este taller en el archivo (`template-alumnos.zip`). Una vez descomprimido encontrarán dentro de la carpeta `template-alumno/` los archivos para trabajar. A continuación detallamos el contenido de los mismos:

- `main.py`: Contiene el código con las llamadas a las funciones de testing (de prueba) de cada ejercicio, y que se encuentran en `cases.py`.
- `cases.py`: Contiene las funciones de testing de cada ejercicio, una función por ejercicio. Cada función de testing contiene varios llamados a la función que realiza el test (comparación de resultados) para cada caso de prueba establecido. La función de testing del ejercicio retorna `true` cuando todos los casos probados son exitosos.
- `ejercicios.py`: Contiene la implementación de las funciones que resuelven la problemática de cada ejercicio.
- `template-alumnos/datos/` carpeta con archivos de datos para 2 de los ejercicios.

Las fuentes de los errores pueden ser de diferentes tipos: errores de sintaxis, divisiones por cero, errores en los datos de entrada, etc.

Aclaración 1: Utilizar las herramientas de debugging para poder corregir los ejercicios y obtener el puntaje ideal. Mantener la estructura del código. No es válido cambiar todo el código por otra implementación completamente diferente. La idea es poder encontrar dónde está el error y corregir solamente esa parte. En todos los ejercicios hay al menos un error en una parte del código.

Aclaración 2: Las anotaciones de tipo utilizadas requieren usar la versión de python 3.9 o superior. En caso de no poder actualizar a esa versión, se pueden borrar las anotaciones de tipo de listas.

Aclaración 3: Lectura de archivos en python. Hay dos ejercicios donde se lee un archivo de texto en python. Para estos ejercicios basta con tener en cuenta lo siguiente:

`open` es una función built-in (no hace falta importar módulos) que recibe como parámetros (entre otros): la ruta del archivo a leer, el “modo” (lectura, escritura, etc) y el encoding (útil para evitar problemas en windows con, por ejemplo, tildes).

Supongamos que leemos un archivo de texto (`path_archivo`) con el contenido “Hola, Introducción a la Programación!”. ¿Qué imprime el siguiente código?

```
def ejemplo():
    miArchivo = open(path_archivo, mode="r", encoding="utf-8")
    for palabra in miArchivo.read().split():
        print(palabra)
    miArchivo.close() #importante: siempre cerrar luego de trabajar con archivos
```

`miArchivo.read().split()` va leyendo palabras (caracteres contiguos sin espacios). Entonces, el código imprimiría por pantalla:

```
Hola ,
Introducción
a
la
Programación!
```