

**NTER TECH SERVICES**

**Python – Ejercicios: Día 3**

Contenido

[7. La Biblioteca Estándar 3](#_heading=h.gjdgxs)

[8. Errores y Excepciones 4](#_heading=h.30j0zll)

[9. Concurrencia en Python 5](#_heading=h.2et92p0)

# La Biblioteca Estándar

**Ejercicio 1:**

1. ¿Cuál es la diferencia entre os.path.join() y os.path.abspath?
2. ¿Cuál es el propósito del módulo os? Da ejemplos de su uso.
3. ¿Cuál es el propósito del módulo logging? ¿Cómo se usa?

**Ejercicio 2:**

En este ejercicio, utilizarás el módulo *random* para seleccionar frutas aleatorias de una lista y crear un batido:

frutas = ['manzana', 'plátano', 'cereza', 'mango', 'piña', 'fresa']

1. Selecciona una fruta aleatoria de la lista para el batido:
2. Mezcla la lista de frutas (usa random.shuffle) y selecciona las dos primeras frutas de la lista mezclada para añadir al batido:
3. Elige una muestra de tres frutas de la lista de frutas para decorar el batido.
4. Usa itertools para generar todas las combinaciones posibles de tres frutas.

**Ejercicio 3:**

En este ejercicio, utilizarás los módulos os y re de Python para buscar un patrón de expresiones regulares (regex) en todos los archivos TXT de un directorio específico. El objetivo es encontrar todas las palabras que terminan en un signo de exclamación (!) y mostrarlas en la consola.

1. Usa os.listdir para obtener todos los archivos en el directorio actual.
2. Filtra los archivos para obtener solo los archivos con extensión .txt.
3. Define una expresión regular que coincida con todas las palabras que terminan en !.
4. Abre cada archivo TXT y lee su contenido.
5. Busca coincidencias del patrón regex en el contenido del archivo.
6. Imprime las coincidencias encontradas en la consola.

**Ejercicio 4 - logging:**

1. Crea un logger llamado ‘t4\_logger’.
2. Configura el nivel del logger a DEBUG para capturar todos los niveles de mensajes.
3. Crea un manejador de archivo (FileHandler) para registrar mensajes de nivel WARNING y superiores en un archivo llamado ‘t4\_warning.log’.
4. Crea otro manejador de archivo (FileHandler) para registrar mensajes de nivel CRITICAL en un archivo llamado ‘t4\_critical.log’.
5. Asegúrate de aplicar un formateador (Formatter) a ambos manejadores que incluya la fecha y hora, el nivel de registro y el mensaje.

# Errores y Excepciones

**Ejercicio 1:**

1. ¿Cuál es el propósito de la declaración ‘raise’?
2. ¿Cómo puedes crear y usar una excepción personalizada?
3. Uso de la declaración ‘assert’

**Ejercicio 2:**

Implementa un programa que solicite al usuario que ingrese un número entero. Debes asegurar que el usuario ingresa un valor válido. Si el usuario ingresa un valor no numérico, se mostrará un mensaje de error y se le pedirá que lo intente nuevamente.

**Ejercicio 3:**

1. Implementa una función para leer números desde un archivo de texto:

Texto

Descripción generada automáticamente

1. Crea una excepción personalizada llamada InvalidDataError para manejar casos de datos no válidos.
2. Valida que todos los elementos en la lista de datos sean números y que la lista no esté vacía. Lanza una excepción InvalidDataError si los datos no son válidos.
3. Realiza algunas operaciones básicas en los datos, como el cálculo de la media, si los datos son válidos.
4. Usa bloques try, except, else y finally para manejar las excepciones y asegurar que el código se ejecute correctamente.
5. Configura el módulo logging para registrar mensajes en un archivo llamado data\_analysis.log con un nivel de registro DEBUG y un formato que incluya la fecha, el nivel de registro y el mensaje.

# Concurrencia en Python

**Ejercicio 1:**

1. ¿Qué es el Global Interpreter Lock (GIL)?
2. Diferencias entre *multithreading* y *multiprocessing*

**Ejercicio 2 - Multithreading**

1. Define una función lector\_archivos\_multihilo() que lea múltiples archivos en paralelo utilizando hilos.
2. La función debe tomar como parámetro una lista de rutas de archivos.
3. Crea una lista de hilos e inicia un hilo para cada archivo en la lista de rutas de archivos.
4. En el bloque principal del programa (if \_\_name\_\_ == "\_\_main\_\_":), define una lista de rutas de archivos (rutas\_archivos) y llama a la función lector\_archivos\_multihilo para leer los archivos.

**Ejercicio 3 - Multiprocessing:**

En este ejercicio deberás crear un programa que sume todos los números del 0 al 100.000.000, tres veces, utilizando multiprocessing y también de forma secuencial.

Una captura de pantalla de un celular

Descripción generada automáticamente con confianza media

Demuestra cómo el uso de multiprocessing (con, por ejemplo, 3 procesos) puede mejorar el rendimiento de una tarea intensiva en CPU al aprovechar los múltiples núcleos.