# Instituto de formación técnica superior N° 18

## Paradigmas de Programación - Ejercicios de programación 2

Profesor Bonini Juan Ignacio

### Ejercicios python avanzado

1. **Decoradores**
   - 1.1. Hacer un decorador para registrar las llamadas a una función, junto con sus argumentos y resultados.
   - 1.2. Hacer un decorador para verificar que los argumentos de una función sean del tipo correcto.
   - 1.3. Hacer un decorador para agrega un retardo antes de que se ejecute una función
   - 1.4. Hacer un decorador para verificar las precondiciones antes de ejecutar una función.
2. **Administradores de contexto**
   - 2.1. Hacer un administrador de contexto para notificar eventos al entrar y al salir de un bloque de código.
   - 2.2. Crea un administrador de contexto que permita cambiar el directorio de trabajo al entrar en un bloque y volver al directorio original al salir. Ejemplo:
3. **Imports**
   - 3.1. Crear dos módulos en el mismo directorio. Desde un módulo, importa una función o variable del otro utilizando una importación absoluta.
   - 3.2. Crear dos módulos en el mismo directorio. Desde un módulo, importa una función o variable del otro utilizando una importación relativa.
   - 3.3. Crear dos módulos en el mismo directorio. Desde un módulo, importar el otro sin usar from.
   - 3.4. Crear dos módulos en el mismo directorio. Desde un módulo, importa una función o variable del otro utilizando una importación absoluta y generar un error de importación circular.
   - 3.5. Crear dos módulos en el mismo directorio. Desde un módulo, importar el otro sin usar from y utilizando alias.

3.6. Crear dos módulos en el mismo directorio. Desde un módulo, importa una función o variable del otro utilizando una importación absoluta y utilizar un alias

3.7. Crear dos módulos en el mismo directorio. Desde un módulo, importa una función o variable del otro utilizando una importación relativa y utilizar un alias

**4. Funciones Lambda**

4.1. Dada una lista de números, utiliza map y una función lambda para crear una nueva lista que contenga el doble de cada número.

4.2. Toma una lista de cadenas y utiliza map con una función lambda para convertir todas las cadenas en mayúsculas.

4.3. Dada una lista de cadenas, utiliza map y una función lambda para crear una lista con la longitud de cada palabra.

4.4. Toma una lista de números y utiliza map con una función lambda para calcular la raíz cuadrada de cada número.

**5. Algoritmos**

5.1. Escribe una función que sume los dígitos de un número pares de un número entero. Si el número es impar, restarle 3 y sumarlo. Si el número da negativo, sumar 1.

**6. Recursividad**

6.1. Escribe una función recursiva para encontrar y sumar todos los números primos desde 1 hasta un número deseado.

6.2. Escribe una función recursiva para calcular el MCD de dos números enteros.

6.3. Escribe una función recursiva para invertir una cadena.

**7. Excepciones**

7.1. Manejo de excepciones

7.1.1. Escribe un programa que solicite al usuario dos números y realice la división de uno por el otro. Utiliza un bloque try y except para manejar la excepción que ocurre si el segundo número es cero.

7.1.2. Crea una lista de números y, a continuación, intenta acceder a un elemento en un índice especificado por el usuario. Utiliza un

bloque try y except para manejar la excepción que se produce si el índice está fuera de rango.

7.1.3. Solicita al usuario que ingrese una cadena que represente un número. Utiliza un bloque try y except para manejar la excepción que se produce si la cadena no se puede convertir a un número.

7.1.4. Escribe un programa que intente abrir un archivo que no existe y utilice un bloque try y except para manejar la excepción de "FileNotFoundError".

7.1.5. Crea un diccionario y luego intenta acceder a un valor utilizando una clave que no está en el diccionario. Utiliza un bloque try y except para manejar la excepción que se produce si la clave no existe.

7.2. Excepciones personalizadas

7.2.1. Para cada caso anterior del manejo de excepciones (7.1.1, 7.1.2, 7.1.3, 7.1.4, 7.1.5) crear una excepción personalizada.

7.3. Bloques try-except-finally

7.3.1. Escribe un programa que intente abrir un archivo, leer su contenido y luego cerrarlo. Utiliza bloques try, except y finally para asegurarte de que el archivo se cierre correctamente, incluso si ocurre una excepción durante la lectura.

7.3.2. Crea un programa que solicite al usuario dos números y una operación matemática (suma, resta, multiplicación, división) para realizar. Utiliza bloques try, except y finally para manejar cualquier excepción que pueda ocurrir durante la operación y asegurarte de que los recursos se liberen correctamente.

7.3.3. Escribe un programa que abra un archivo, lea su contenido y escriba el mismo contenido en otro archivo. Utiliza bloques try, except y finally para manejar cualquier excepción que pueda ocurrir durante la lectura o escritura, y asegúrate de que ambos archivos se cierren correctamente.

7.4. Lanzamiento de excepciones

7.4.1. Capturar las excepciones personalizadas en el punto 7.2, imprimir un mensaje en pantalla y lanzarlas nuevamente.

**8. Manejo de archivos**

8.1. Escribe un programa que abra un archivo de entrada, lea su contenido y luego escriba ese contenido en un nuevo archivo de salida. Asegúrate de cerrar ambos archivos al final.

8.2. Escribe un programa que abra un archivo de texto, cuente cuántas palabras contiene y muestre el resultado en la pantalla.

8.3. Lee un archivo CSV que contiene registros de datos y realiza alguna operación de procesamiento en los datos, cómo calcular promedios, encontrar valores máximos o mínimos, o filtrar registros que cumplan ciertas condiciones.

8.4. Escribe un programa que tome varios archivos de texto y los concatena en un solo archivo de salida. Asegúrate de cerrar todos los archivos correctamente.

8.5. Lee un archivo CSV que contiene registros de datos y convertirlo en un archivo JSON.

**9. Programación concurrente / paralelo**

9.1. Crea dos hilos que ejecuten dos funciones diferentes simultáneamente y muestran mensajes de salida.

9.2. Implementa el problema del productor-consumidor utilizando hilos, donde un hilo produce datos y otro hilo los consume desde una cola compartida.

9.3. Crea dos procesos utilizando la biblioteca multiprocessing y ejecuta funciones diferentes en cada proceso.

9.4. Utiliza un pool de procesos para realizar operaciones en paralelo en una lista de datos.