Universidad Autónoma de Baja California

Alumno: Colin Reyes Brian Gabriel

Matrícula: 379494

Materia: Phyton

Docente: Pedro Nuñez Yepiz

Periodo: Agosto - Diciembre

Grado/semestre: 3er semestre

Grupo: 432

Tema: Actividad 5

Fecha: 16/09/25

INTRODUCCIÓN

El presente trabajo reúne una serie de programas en Python desarrollados como práctica para reforzar los fundamentos de la programación estructurada. Cada ejercicio aborda un problema distinto, desde el manejo de listas y cadenas de texto hasta la generación de números aleatorios, la eliminación de duplicados y el cálculo de estadísticas básicas como la media y la mediana. Estos programas no solo permiten afianzar la lógica de programación, sino también comprender la importancia de validar entradas, manejar errores de manera adecuada y aprovechar librerías y estructuras de datos propias de Python.

TEORÍA

Listas: estructuras de datos que permiten almacenar y manipular colecciones de elementos de manera ordenada y dinámica.

Ciclo for: herramienta que facilita la iteración sobre los elementos de una lista u otra secuencia.

Función len(): devuelve la longitud de una lista o cadena, indispensable para trabajar con posiciones y tamaños.

Biblioteca random: proporciona funciones para generar números aleatorios, útiles en simulaciones o pruebas.

Conjuntos (set): estructuras que no permiten duplicados y son ideales para depuración de listas repetidas.

Manejo de excepciones (try-except): mecanismo para controlar errores de ejecución y evitar que el programa se interrumpa de manera abrupta.

Ordenamiento (sort / sorted): proceso de organizar elementos en orden creciente o decreciente, fundamental para el cálculo de la mediana.

Media y mediana: medidas estadísticas que permiten resumir un conjunto de datos, la primera como promedio aritmético y la segunda como el valor central de la distribución.

EJERCICIO 1: Nombres de Mascotas o Artistas Favoritos

Función que utilice una lista con los nombres de tus mascotas o artistas favoritos (mínimo 5, máximo 10). Imprimir cada cadena junto con la cantidad de caracteres que contiene.

```
            break
        else:
            print("Rango incorrecto. Intenta de nuevo.")

    lista = []
    for n in range(i):
        nombre = input(f"Ingresa el nombre número {n}: ").strip()
        lista.append(nombre)

    for idx, nombre in enumerate(lista):
        longitud = len(nombre)
        print(f"[{idx}] {nombre.upper()} --> {longitud} CARACTERES")

if __name__ == "__main__":
    main()
```

```
Ingresa el número de mascotas o artistas que añadirás a la lista (5 a 10): perro
Por favor ingresa un número entero.
Ingresa el número de mascotas o artistas que añadirás a la lista (5 a 10): 6
Ingresa el nombre número 0: perro
Ingresa el nombre número 1: gato
Ingresa el nombre número 2: avion
Ingresa el nombre número 3: movil
Ingresa el nombre número 4: 44
Ingresa el nombre número 5: luis
[0] PERRO --> 5 CARACTERES
[1] GATO --> 4 CARACTERES
[2] AVION --> 5 CARACTERES
[3] MOVIL --> 5 CARACTERES
[4] 44 --> 2 CARACTERES
[5] LUIS --> 4 CARACTERES
```

EJERCICIO 2: Generación de Números Aleatorios

Crear dos funciones:
1. Una función que genere y regrese una lista con 10 números aleatorios entre 30 y 50 (sin repetidos).
2. Una función que reciba una lista e imprima cada elemento junto con su índice.

```
import random

def generar_numeros_aleatorios(cantidad=10, inicio=30, fin=50):

    if cantidad < 1:
        raise ValueError("La cantidad debe ser al menos 1.")
    if cantidad > (fin - inicio + 1):
        raise ValueError("Rango insuficiente para generar números únicos.")
    return random.sample(range(inicio, fin + 1), cantidad)

def imprimir_con_indices(lista):
    for idx, val in enumerate(lista):
        print(f"[{idx}] {val}")

if __name__ == "__main__":
    numeros = generar_numeros_aleatorios()
    imprimir_con_indices(numeros)
```

```
[0] 39
[1] 36
[2] 34
[3] 43
[4] 50
[5] 46
[6] 38
[7] 45
[8] 40
[9] 30
```

EJERCICIO 3: Suma de Elementos Correspondientes

Escribir una función que reciba dos listas de números del mismo tamaño y calcule la suma de los elementos correspondientes de cada lista. Si las listas no son del mismo tamaño, usar el tamaño de la lista más pequeña.

```
        except Exception as e:
            try:
                suma = float(list1[i]) + float(list2[i])
            except Exception:
                raise TypeError(f"No se pudo sumar los elementos en la posición {i}: {list1[i]!
        resultado.append(suma)
    return resultado

if __name__ == "__main__":
    print("Ingresar la primera lista (ejemplo: 1 2 3 4 o 1,2,3,4 o 1 2.5 3):")
    lista1 = parsear_lista_desde_input("Lista 1: ")
    print("Ingresar la segunda lista (mismo formato):")
    lista2 = parsear_lista_desde_input("Lista 2: ")

    try:
        resultado = sumar_elementos_correspondientes(lista1, lista2)
        print("Lista 1:", lista1)
        print("Lista 2:", lista2)
        print("Resultado:", resultado)
    except Exception as err:
        print("Ocurrió un error al sumar las listas:", err)
```

```
Ingresar la primera lista (ejemplo: 1 2 3 4 o 1,2,3,4 o 1 2.5 3):
Lista 1: 4 5 32 5
Ingresar la segunda lista (mismo formato):
Lista 2: 2 42 2 4 4
Advertencia: las listas tienen distinto tamaño. Se usará el tamaño de la lista más pequeña.
Lista 1: [4, 5, 32, 5]
Lista 2: [2, 42, 2, 4, 4]
Resultado: [6, 47, 34, 9]
```

EJERCICIO 4: Eliminar Duplicados

Escribir una función llamada eliminar_duplicados que reciba una lista como parámetro y elimine los elementos duplicados. El resultado debe ser una nueva lista sin duplicados.

```
def parsear_lista_enteros(prompt="Ingresa números enteros separados por espacios o comas: "):
    while True:
        linea = input(prompt).strip()
        if not linea:
            print("No ingresaste nada. Intenta de nuevo.")
            continue
        tokens = [t for t in linea.replace(",", " ").split() if t != ""]
        try:
            numeros = [int(tok) for tok in tokens]
            return numeros
        except ValueError:
            print("Entrada inválida: asegúrate de ingresar sólo números enteros separados por espacios o comas.")


if __name__ == "__main__":
    print("Eliminar duplicados — se conservará el primer orden de aparición.")
    lista = parsear_lista_enteros("Lista original: ")
    try:
        sin_duplicados = eliminar_duplicados(lista)
        print("Lista original:", lista)
        print("Lista sin duplicados:", sin_duplicados)
    except TypeError as err:
        print("Error:", err)
```

```
Eliminar duplicados — se conservará el primer orden de aparición.
Lista original: 2 4 2 43
Lista original: [2, 4, 2, 43]
Lista sin duplicados: [2, 4, 43]
```

EJERCICIO 5: Media y Mediana

Escribir una función que calcule la media y la mediana de una lista de números enteros.

```python
def parsear_lista_enteros(prompt="Ingresa números enteros separados por espacios o comas: "):
    while True:
        linea = input(prompt).strip()
        if not linea:
            print("No ingresaste nada. Intenta de nuevo.")
            continue
        tokens = [t for t in linea.replace(",", " ").split() if t != ""]
        try:
            numeros = [int(tok) for tok in tokens]
            return numeros
        except ValueError:
            print("Entrada inválida: asegúrate de ingresar sólo números enteros (ej. 1 2 3 o 1,2,3).")


if __name__ == "__main__":
    print("Cálculo de media y mediana (lista de enteros).")
    lista = parsear_lista_enteros("Lista: ")
    try:
        media, mediana = calcular_media_mediana(lista)
        print("Lista:", lista)
        print("Media:", media)
        print("Mediana:", mediana)
    except Exception as e:
        print("Ocurrió un error:", e)
```

```
Cálculo de media y mediana (lista de enteros).
Lista: 1  4 4 3 5 3
Lista: [1, 4, 4, 3, 5, 3]
Media: 3.3333333333333335
Mediana: 3.5
```

CONCLUSIÓNES

A lo largo de estos ejercicios se reforzaron conocimientos fundamentales de programación en Python, como el manejo de listas, ciclos y funciones, así como la aplicación de estructuras de control y librerías externas. Cada programa permitió poner en práctica habilidades distintas: desde manipular cadenas y números hasta organizar datos, evitar duplicados y calcular estadísticas simples. Además, se aprendió la importancia de validar entradas y manejar errores con try-except, lo cual hace que los programas sean más seguros y confiables.

Como posibles mejoras, se podrían optimizar algunos programas para aceptar diferentes tipos de datos, agregar interfaces gráficas o trabajar con archivos externos para leer y guardar información. También sería útil implementar pruebas automáticas para verificar que las funciones trabajen correctamente en distintos escenarios. En conjunto, estas prácticas contribuyen a desarrollar una lógica de programación más sólida y a preparar el camino para resolver problemas más complejos en el futuro.