Ejercicio 1: Calculadora Simple

El programa solicita al usuario dos números y una operación matemática (suma, resta, multiplicación o división). Luego, realiza la operación seleccionada y muestra el resultado. Se repite hasta que el usuario decida salir.

Ejercicio 2: Juego de Adivinanza

El programa genera un número aleatorio entre 1 y 100. El usuario debe adivinar este número. El programa proporciona pistas (mayor o menor) y permite al usuario continuar adivinando hasta que lo adivine correctamente. **texto en negrita**

¿Que son las colecciones en Python?

Es un término general utilizado para referirse a estructuras de datos que permiten almacenar múltiples elementos en una sola variable. Estas estructuras de datos son fundamentales para trabajar con conjuntos de datos más complejos y flexibles.

En Python existen cuatro colecciones básicas:

- Listas
- Tuplas
- Conjuntos
- Diccionarios

Listas [list]

Son secuencias ordenadas de elementos accesibles a través de un índice, que pueden ser de cualquier tipo de dato. Se definen utilizando corchetes [ ] y los elementos se separan por comas. Las listas son mutables, lo que significa que se pueden modificar después de su creación mediante la adición, eliminación o modificación de elementos.

Las listas son el tipo de colección en Python equivalente a los arreglos.

```
mi_lista = [1, 2, 3, 4, 5]
print (type(mi_lista))
```

Creación e Inserción

Las listas utilizan corchetes para la creación y acceso de sus elementos

Podemos crear listas vacías, es decir, sin elementos, utilizando corchetes [] y luego llenarla añadiendo elementos con el **método append()**, que agrega un elemento al final de la lista.

```
lista_vacia = []
```

```
lista_vacia = []
print(lista_vacia)

#lista_vacia.append(10)
#lista_vacia.append(20)
#lista_vacia.append(30)
#print(lista_vacia)
```

O podemos insertar datos en una posición específica con el método **insert**

```
lenguajes = ["Python", "Java", "PHP", "C++", "Ruby"]
lenguajes.insert(2, "Visual Basic")
print (lenguajes)

#Que realiza este codigo??
#lenguajes.insert(round(len(lenguajes)/2),"Delphi")
#print (lenguajes)
```

Tambien podemos crear listas con elementos ya definidos e imprimir la lista completa o acceder a un elemento de la lista por índice.

```
lenguajes = ["Python", "Java", "PHP"]
print (lenguajes)  #Imprime la lista completa
print (lenguajes[0]) #Imprime el valor de la posición 0
```

Una lista en Python puede almacenar elementos de cualquier tipo de dato

```
persona = [1119999, "ana", 38, 5.0,True ]

print (persona)
print (persona[0])
persona [4] = "Python 3"
print (persona)
```

Incluso otras listas

```
mi_lista = [2, 3.5, True, "amigo", [3, 8, "a"]]
print (mi_lista)
```

¿De qué manera accederías al elemento con valor 6 en la siguiente lista?. Completa el código en la línea 2

```
mi_lista = [1, 2, 3, ['p', 'q', [5, 6, 7]]]
print(mi_lista)
```

¿Acertaste? De esa manera podemos acceder a cualquier elemento de una lista.

Modificación

Podemos modificar elementos de la lista simplemente asignando un nuevo valor a través de su índice.

```
lista = [1, 2, 3, 4, 5]
lista[2] = 10
print(lista)
```

¿Podrías indicar qué imprime el siguiente código?

```
numeros = [3,4,6,2,6,8,4,9]
res = numeros [-3] + numeros [5]
print (res)
```

Otra forma alternativa de crear listas es mediante la función **list** que recibe como argumento un objeto iterable

```
mi_lista = list(range(1,11))
print (type(mi_lista))
```

```
print (mi_lista)
print (mi_lista[3])
```

```
mi_lista = list("1234567")
print (type(mi_lista))
print (mi_lista)
```

Reto: En el ejemplo anterior nos genera una lista de str pero como los convierto en una lista de números enteros. pista funcion map, me ayudas??

```
mi_lista = list("1234567")
```

¿Recuerdas el concepto de **Slicing** que vimos con los datos tipo cadena?. Pues también es aplicable a las listas.

```
mi_lista = list (range (1,10))
print (mi_lista)
print (mi_lista [1:4])
print (mi_lista [:2])
print (mi_lista [7:])
print (mi_lista [-5:-1])
print (mi_lista [-3:])
```

Como vamos viendo, las listas en Python son muy flexibles. Aquí algunas otras operaciones nativas sobre las listas.

Asignación de varios valores

```
mi_lista = [1, 2, 3, 4, 5]
mi_lista [0:3] = ["a", "b", "c"]
print (mi_lista)
```

```
mi_lista = [1, 2, 3, 4, 5]
mi_lista = mi_lista + [6] #Agregar un valor con el operador +
print (mi_lista)
mi_lista = mi_lista + [7, 8, 9] #Agregar varios valores
print (mi_lista)
```

```
mi_lista = [1, 2, 3]
x, y, z = mi_lista #Asignar a varias variables el contenido de una lista
print (f"valor de x {x}")
print (f"valor de y {y}")
print (f"valor de z {z}")
```

Eliminación

Podemos eliminar un elemento de una lista por su índice utilizando la palabra reservada "**del**"

```
lenguajes = ["Python", "Java", "PHP"]
del lenguajes[1]
print (lenguajes)
```

O podemos eliminar un elemento de una lista utilizando el método "**remove()**" pasando el elemento a eliminar

```
lista = [1, 2, 3, 4, 5, 3, 5, 3]
lista.remove(5)
print(lista)
```

```
[1, 2, 3, 4, 3, 3]
```

```
lenguajes = ["Python", "Javax", 5, "Java"]
lenguajes.remove("Javax")
print (lenguajes)
```

Para saber si un elemento se encuentra dentro de una lista usamos la palabra reservada **in**

```
lenguajes = ["Python", "Java", "PHP"]
print ("Python" in lenguajes)
print ("C#" in lenguajes)
```

Recorrido de listas

Dado que cada lista es un objeto iterable, las podemos recorrer con nuestro ciclo for

```
 for variable in lista:
```

```
numeros = [1, 2, "papa", 3, 4, 0]
print(numeros)

for x in numeros:
  print (x, end= ",")
```

```
numeros = [1, 2, 3, 4, 5, "papa"]
for x in numeros:
  if type(x) is int:
    if x % 2 == 0:
      print (x)
```

```
lista =[]
x = int(input("Ingrese la cantidad de datos a almacenar en la lista "))
for i in range(x):
  lista.append(int(input("ingrese valor")))

print ("La lista es ", lista)
```

También nos las podemos recorrer una lista a través de sus índices como convencionalmente se hace en otros lenguajes. Para ello, necesitamos usar la función **len** que nos devuelve la longitud de la lista.

```
mi_lista = [4, 5, 6, 7]
for i in range(0, len(mi_lista)):
      print(mi_lista[i], end=",")
```

```
for i in mi_lista:
    print(i, end=",")
```

Python nos ofrece una forma alternativa de acceder al índice de cada elemento de una lista a través de la función **enumerate**

```
numeros = [3, 29, 34, 42, 54]
for x in enumerate(numeros):
  print (x)
```

Debido a que la función enumerate devuelve para cada elemento de la lista un par (índice - valor), podemos obtener ambos al tiempo en la misma variable como en el ejemplo anterior o acceder a cada uno a través de variables diferentes. Veamos:

```
numeros = ["A", "B", "C", "D", "E"]
for a, b in enumerate(numeros):
  print (f"En el indice {a} encontramos el valor {b}")
```

Con lo visto hasta ahora, muéstranos cómo recorrerías una lista de la cual no conoces su longitud usando el ciclo for y luego usando el ciclo while.

```
#recorrer la lista con el ciclo for
lista = [2,3,4,5,6,7,8,3,43,65,6,76,67,87,87]
```

```
#recorrer la lsiata con el ciclo while
lista = [2,3,4,5,6,7,8]
```

Concatenar listas

Podemos concatenar listas utilizando el operador "**+**"

```
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
lista_concatenada = lista1 , lista2
print(lista_concatenada)
```

Ten encuenta que si utilizas la coma, para concatenar y crear una nueva lista, Python trata cada elemento separado por coma como un elemento individual dentro de la lista (empaquetamiento de argumentos) Por lo tanto, si utilizas la coma, para crear una nueva lista, estarás creando una lista que contiene las listas originales como elementos individuales, no una lista que contenga los elementos de ambas listas combinadas.

```
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
lista_empaquetada = [lista1, lista2]
print(lista_empaquetada)
```

Se cuenta con dos listas y se desean recorrer de manera simultánea mostrando el valor de las dos en cada posición. ¿Ayudarías a terminar el código de tal manera que se obtenga el siguiente resultado?

```
 ma - pa
 me - pe
 mi - pi
 mo - po
 mu - pu
```

```
lista_1 = ["ma", "me", "mi", "mo", "mu"]
lista_2 = ["pa", "pe", "pi", "po", "pu"]

#continua aquí
```

Después de haberlo logrado, miremos otra alternativa para el recorrido de dos listas simultáneamente a través de la función **zip**

```
lista_1 = ["ma", "me", "mi", "mo", "mu", "ja"]
lista_2 = ["pa", "pe", "pi", "po", "pu", "jo"]
for x in zip (lista_1, lista_2):
    print (x)
```

A la función **zip** le podemos pasar cualquier número de listas y tendrá la capacidad de generar un iterador que las recorra simultáneamente.

```
nombres = ["maria", "luis", "carlos", "Juan", "Diego", "Lina", "Camilo"]
edades = [34, 20, 14, 18, 20]
notas = [4, 3.8, 2, 5]
for x in zip (nombres, edades, notas):
    print (x)
```

```
nombres = ["maria", "luis", "carlos", "Juan", "Diego", "Lina", "Camilo"]
edades = [34, 20, 14, 18, 20]
notas = [4, 3.8, 2, 5]

for nom, ed, no in zip(nombres, edades, notas):
    print("Nombre:", nom)
    print("Edad:", ed)
```

```
    print("Nota:", no)
```

Operaciones con listas

Comienza a programar o generar con IA.

Ordenar una lista

Puedes ordenar una lista usando el método sort() para ordenarla en su lugar o la función sorted() para obtener una nueva lista ordenada.

Ejemplo utilizando el método **sort()**

```
#mi_lista = [5, 6, 3, 8, 1]
mi_lista = ["s1", "a3", "p1", "b5", "x7","a1"]
mi_lista.sort()
print (mi_lista)
```

Ejemplo utilizando la función sorted()

```
palabras = ["manzana", "banana", "pera", "uva"]
palabras_ordenadas = sorted(palabras)
print(palabras_ordenadas)
```

Podemos pasarle el argumento **reverse=True** al método **sort()** para que ordene inversamente

```
mi_lista = ["d", "a", "b"]
mi_lista.sort(reverse=True)
print (mi_lista)
```

Invertir una lista

Se puede invertir una lista utilizando el método reverse() para invertirla en su lugar o el slicing para obtener una nueva lista invertida.

```
lenguajes = ["Python", "Java", "PHP"]
lenguajes.reverse()
print (lenguajes)
```

```
palabras = ["uno", "dos", "tres", "cuatro"]
palabras_invertidas = palabras[::-1]
print(palabras_invertidas)
```

Otros métodos o funciones aplicables sobre listas

Puedes contar la cantidad de elementos en una lista utilizando la función len().

```
lista = [10, 20, 30, 40, 50]
cantidad_elementos = len(lista)
print(cantidad_elementos)
```

Para sumar los elementos de una lista podemos utilizar la función sum()

```
numeros = list(range(20, 200,4))
print(numeros)
suma = sum(numeros)
print(f"la suma de los numeros de la lista es {suma}: ")
```

La función count() en Python se utiliza para contar cuántas veces aparece un elemento específico en la lista.

```
print(["Hola", "mundo", "mundo"].count("mundo"))
```

Encontrar el máximo y mínimo valor de una lista utilizando la función max() y utilizando la función min().

```
numeros = [10, 5, 20, 30, 15, 200, 6, 89]
print("Máximo:", max(numeros))
print("Mínimo:", min(numeros))
```

Esta seria la forma básica como se desarrollaria un ejercicio que genere el promedio de 4 notas e identificar la nota mayor. Ahora dime como lo realizariamos utilizando listas con sus métodos o funciones

```
n1 = 3.4
n2 = 5
n3 =4.2
n4 = 3.1

np = (n1+n2+n3+n4)/4

if n1 > n2 y n1 > n3 y n1 > n4 entonces
    mayor = n1
elif n2 > n1 y n2 > n3 y n2 > n4 entonces
    mayor = n2
elif n3 > n1 y n3 > n2 y n3 > n4 entonces
    mayor = n3
else
    mayor = n4

print("El promedio es:", np)
print("La nota mayor es:", mayor)


notas = [3.4, 5, 4.2, 3.1]
#.....
```

clear(): Este método se utiliza para eliminar todos los elementos de una lista, dejándola vacía. Por ejemplo:

```
lista = [1, 2, 3, 4, 5]
lista.clear()
print(lista)
```

extend(): Este método se utiliza para agregar los elementos de una lista al final de otra lista. Ejemplo:

```
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
lista1.extend(lista2)
print(lista1)
```

index(): Este método se utiliza para encontrar el índice de la primera aparición de un elemento en una lista. Por ejemplo:

```
lista = [1, 2, 3, 4, 5]
indice = lista.index(3)
print(indice)
```

pop(): Este método se utiliza para eliminar y devolver el elemento en la posición especificada de una lista. Por ejemplo:

```
lista = [1, 2, 3, 4, 5]
elemento = lista.pop(3)
print(elemento)
print(lista)
```

Reto: Eliminar todas las ocurrencias de un número en una lista

```
lista = [30,20,30,40,30,50,30,20,30,30]
```

Tuplas

Generalidades

Las Tuplas son similares a las listas pero con la característica de ser inmutables, es decir, sus valores no pueden cambiar. Las Tuplas se crean con paréntesis **()** en lugar de corchetes **[]** o tambien se pueden crear sin parentesis, los elementos se separan por comas y al igual que las listas, se accede a sus posiciones con corchetes **[]** Veamos:

```
mi_tupla = (2,3,4,5,6,9)
print (type(mi_tupla))
```

```
mi_tupla = (2,3,4,5,6,9)
print (mi_tupla)
print (mi_tupla[1])
print (mi_tupla[1:3])
print (mi_tupla[:])
```

```
tupla_sin_parentesis = 1, 2, 3
print (type(tupla_sin_parentesis))
print (tupla_sin_parentesis)
```

Intenta cambiar el contenido de la posición 1 en la siguiente tupla por el valor de "f":

```
mi_tupla = ("a","b","c","d")
print (type(mi_tupla))
#mi_tupla[1] = "f"
#print (mi_tupla[1])
#print(mi_tupla)
```

Como puedes ver, el contenido de una **tupla es inmutable**

Las Tuplas al igual que las Listas pueden almacenar diferentes tipos de elementos y pueden conformarse por otras listas o tuplas internas. El acceso a sus elementos funciona igual que con las listas.

```
mi_tupla = (100, 'Hola', [1, 2, 3], -50)
print (mi_tupla[2][-1])
```

```
mi_tupla = (100, 'Hola', (1, (200, 'D', ["x","y","z"])), (1, 2, 3), -50)
print (mi_tupla [2][1][2][1])
```

Las Tuplas comparten varios métodos de las listas, sin embargo, por ser inmutables, las Tuplas no usan ningún método en cuyo resultado sea un cambio de contenido. Ejemplo append(), insert(), remove(), sort(), reverse(), etc.

¿Que opinan con el método **count** funciona?

```
mi_tupla = (1,2,5,2,6,9,3,2,78,45,3,2,0)
print (mi_tupla.count (2))
```

¿Que opinan con el método **sort** funciona?

```
mi_tupla = (1,2,5,2,6,9,3,2,78,45,3,2,0)
mi_tupla.sort()
print(mi_tupla)
```

Las tuplas también las podemos crear a partir de la función **tuple** que recibe un objeto iterable

```
cadena = [2,3,54,65,76,4]
t = tuple (input("valor1:")+""+input("valor2:")+""+input("valor3:"))
print (t)
```

```
x =(2,3,4,5,6,2,3,7,9)
print (x.count(2))
```

Por lo demás, podemos resumir diciendo que las listas y las tuplas son semejantes, con la diferencia que las segundas no pueden cambiar su contenido.

Paso de Tupla a Lista y Lista a Tupla

Las funciones **list** y **tuple** son las que nos permiten realizar este tipo de conversiones. miremos

Paso de Lista a Tupla

```
mi_lista = [1,2,3,4,5]
print (f"{mi_lista} es de tipo {type(mi_lista)}")
mi_tupla = tuple (mi_lista)
print (f"{mi_tupla} es de tipo {type(mi_tupla)}")
```

Paso de Tupla a Lista

```
mi_tupla = (1,2,3,4,5)
print (f"{mi_tupla} es de tipo {type(mi_tupla)}")
mi_lista = list (mi_tupla)
print (f"{mi_lista} es de tipo {type(mi_lista)}")
```

```
mi_tupla = (3,8,5,1,9)
print (mi_tupla)
mi_lista = list (mi_tupla)
mi_lista.sort()
mi_tupla = tuple (mi_lista)
print (mi_tupla)
```

```
t1 = (input("ingrese numero:"),input("ingrese numero:"),input("ingrese numero:"))
print (t1)
```

Apropiación

1. Pregunte al usuario cuántos elementos desea ingresar en una lista, luego solicite cada uno de ellos y presente el contenido de la lista y su contenido invertido.

2. Solicite al usuario dos frases y devuelva una lista con todas las letras que se repiten en la misma posición de ambas frases. Ejemplo:

   f1: hola mundo

   f2: como estas

   r=["o"]

3. Leer una frase y almacenar en una tupla la frase leída pero sin espacios. Mostrar el contenido de la tupla.

4. Crear tres lista con la siguiente información (nombre, edad, sexo), imprimir cual es la persona más joven y cual es la persona de más edad contar cuantas son hombres y cuantas son mujeres.

5. Cree un Juego de Adivinanza. El programa seleccionará aleatoriamente una palabra de una lista de frutas. El jugador debe adivinar la palabra en un número limitado de intentos. Después de cada intento, se indicará si la respuesta es correcta o incorrecta. Si la respuesta es incorrecta, se dará una pista como la longitud de la palabra, la primera y última letra de la fruta.

6. Cree una aplicación que presente un menú con las siguientes opciones:

```
Aplicaciones con Listas

1. Ingresar lista nueva
2. Ordenar lista
3. Promedio lista
4. Buscar elemento
5. Salir
```

Ingresar lista solicita los elementos de una lista, terminando con el ingreso de un número negativo el cual no formará parte de la lista

Ordenar lista Presenta los valores de la lista ordenados.

Promedio lista Muestra el promedio de los valores de la lista

Buscar elemento solicita un número a buscar en la lista e indica si se encuentra en ella y cuántas veces aparece.

Salir Termina la ejecución del programa