



PRÁCTICO 1

PUNTEROS LETALES

Jorge Luis Esteves Salas

Fernando Navia

Joel Dalton Montero

Ana Laura Cuellar

Weimar Valda

Leonel Eguez Camargo

Alejandro Hurtado Rodas

Dirigido por el docente:

JIMMY REQUENA LLORENTTY

Materia:

Programación II

ÍNDICE

MIS PRIMERAS PRUEBAS DE REPLIT	3
APRENDIENDO A LLAMAR FUNCIONES.....	4
Versión original de la tarea.....	4
Función creada a partir el código original	4
Prueba llamando a la función desde otro archivo de código de Python:	5
Demostración del Funcionamiento en la terminal de Replit:.....	5
PRUEBAS DE LISTAS O VECTORES.....	6
USANDO FUNCIONES Y LISTAS INVERTIDAS	7
Ejemplo 1:	9
Ejemplo 2:	10
Ejemplo 3.....	11
Ejemplo 4:	11
Ejemplo 5:	12
Ejemplo 6:	13
Ejemplo 7:	15
OPERACIONES CON LAS LISTAS.....	16
Sumar Elementos	16
Encontrar Mayor	17
Pruebas lógicas de funcionamiento.....	18
BUSQUEDA BINARIA.....	19

MIS PRIMERAS PRUEBAS DE REPLIT

Después de seguir los pasos del texto guía en la plataforma pude configurar mi Replit con mi repositorio tuve unas complicaciones y todavía no puedo configurar el token de acceso correctamente pero la sincronización de archivos entre funciona bien:

```
~/workspace: git pull

~/workspace/pruebasProg2$ git commit -m "prueba de subida 2"
On branch main
Your branch is ahead of 'origin/main' by 5 commits.
(use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   ../.replit
    deleted:    ../PruebasProg2/clese02_prueba.py

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    clese02_prueba.py

no changes added to commit (use "git add" and/or "git commit -a")
~/workspace/pruebasProg2$ git push origin main
Enumerating objects: 18, done.
Counting objects: 100% (18/18), done.
Delta compression using up to 8 threads
Compressing objects: 100% (11/11), done.
Writing objects: 100% (15/15), 1.76 KiB | 1.76 MiB/s, done.
Total 15 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/Alejandro26/prog2-alejandrohurtado
   2dab68a..d6c3f11  main -> main
~/workspace/pruebasProg2$ ls
clese02_prueba.py  pruebasubida.md
~/workspace/pruebasProg2$ cd..
bash: cd.: command not found
~/workspace/pruebasProg2$ cd.
```

Añadí y eliminé archivos tanto en el repositorio como en el Replit y funciona bien :

```
~/workspace: git pull

~/workspace$ git add linkUtiles.md
~/workspace$ git commit -m "update del archivo"
[main c161965] update del archivo
 1 file changed, 5 insertions(+)
 create mode 100644 linkUtiles.md
~/workspace$ git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 446 bytes | 446.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Alejandro26/prog2-alejandrohurtado
   d6c3f11..c161965  main -> main
~/workspace$ git pull
Already up to date.
~/workspace$ git pull
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 2 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (2/2), 874 bytes | 874.00 KiB/s, done.
From https://github.com/Alejandro26/prog2-alejandrohurtado
   c161965..39836c0  main      -> origin/main
Updating c161965..39836c0
Fast-forward
 Links Utiles | 3 ---
 1 file changed, 3 deletions(-)
 delete mode 100644 Links Utiles
~/workspace$
```

APRENDIENDO A LLAMAR FUNCIONES

La clase de centro en aprender a usar las funciones y llamarlas para poder optimizar el código use de base los ejercicios que nos dio de tarea para para modificarlos y crear funciones que posteriormente llame

Versión original de la tarea

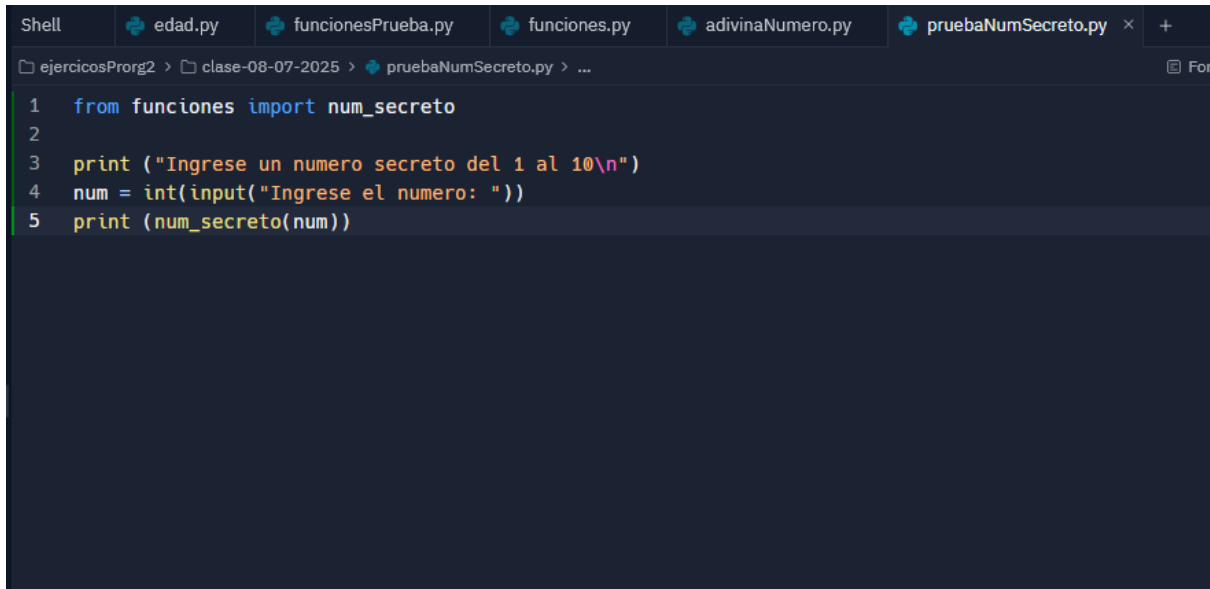
```
Shell  edad.py  funcionesPrueba.py  funciones.py  adivinaNumero.py x  pruebaNumSecreto.py  +  ⋮
ejerciciosProrg2 > acti1 > adivinaNumero.py > ...  Format
1  #Adivina el numero secreto
2
3  num = 7
4
5  print("Adivina el numero secreto del 1 al 10\n")
6  num_user = int(input("\nIngrese el numero: "))
7  if num_user == num:
8      print("\nFelicidades adivinaste el numero secreto")
9  else:
10     print("\nLo siento no adivinaste el numero secreto\n")
11
12  print ("____Alejandro Hurtado____")
13
14
```

Función creada a partir el código original

```
17
18  #Adivina el numero secreto
19
20  def num_secreto(num):
21      print("Adivina el numero secreto del 1 al 10\n")
22      num_user = int(input("Ingrese el numero: "))
23      if num_user == num:
24          return "\nFelicidades, adivinaste el numero secreto"
25      else:
26          return "\nLo siento, no adivinaste el numero secreto\n"
27
28  print ("____Alejandro Hurtado____")
```

Prueba llamando a la función desde otro archivo de código de Python:

Para comprobar el funcionamiento de la función cree un nuevo archivo de Python y llame a la función que anteriormente cree, de esta manera también aprendí que podía crear archivos propios de manera externa e ir construyendo una librería de mis funciones



```
Shell | edad.py | funcionesPrueba.py | funciones.py | adivinaNumero.py | pruebaNumSecreto.py x +
ejercicosProrg2 > clase-08-07-2025 > pruebaNumSecreto.py > ...
1 from funciones import num_secreto
2
3 print ("Ingrese un numero secreto del 1 al 10\n")
4 num = int(input("Ingrese el numero: "))
5 print (num_secreto(num))
```

Demostración del Funcionamiento en la terminal de Replit:

demostración de que la función fue llamada y ejecutada exitosamente en la terminal de Replit.



```
Shell | edad.py | funcionesPrueba.py | funciones.py | adivinaNumero.py | pruebaNumSecreto.py +
~/workspace/ejercicosProrg2/clase-08-07-2025: python pruebaNumSecreto.py
~/workspace/ejercicosProrg2/clase-08-07-2025$ cd ejercicioProrg2/clase-08-07-2025/
~/workspace/ejercicosProrg2/clase-08-07-2025$ python pruebaNumSecreto.py
____Alejandro Hurtado____
Ingrese un numero secreto del 1 al 10

Ingrese el numero: 3
Adivina el numero secreto del 1 al 10

Ingrese el numero: 7

Lo siento, no adivinaste el numero secreto
~/workspace/ejercicosProrg2/clase-08-07-2025$
```

PRUEBAS DE LISTAS O VECTORES

Alejandro Hurtado Rodas – 10/07/2025 – 17:34

Aprendí a utilizar y guardar datos en una lista para que sean más fáciles de manejar y cambiar utilizando las listas en Python, para el primer ejercicio cree un código simple en el que una lista contiene mis tres comidas favoritas y luego las imprime para posteriormente solicitar al usuario que cambie la primera comida de la lista y la vuelva a imprimir en la terminal.

```
listaNombreEstu.py listaNotas.py listaComidaFav.py X
ejerciciosProrg2 > acti2 > listaComidaFav.py > ...
1  #Lista donde se guardan las comidas
2  comida_fav = ["Salteña", "Cuñape", "Sopa de Mani"]
3  print(comida_fav)
4
5  print("\nMi segunda comida favorita es " + comida_fav[1])
6  print("\nCambiar tu primera comida favorita")
7  nueva_comida = input("\nIngresa tu nueva comida favorita:")
8  comida_fav[0] = nueva_comida
9  print(comida_fav)
10 cantidad_comida = len(comida_fav)
11 print(f"\nLa cantidad de comidas favoritas que tienes es de {cantidad_comida}")
```

El segundo ejercicio consiste en tener una lista de nombres e imprimir los nombres con un mensaje de bienvenida usando un bucle For

```
listaNombreEstu.py X listaNotas.py listaComidaFav.py
ejerciciosProrg2 > acti2 > listaNombreEstu.py > ...
1  lista_nom = ["nom1", "nom2", "nom3", "nom4", "nom5"]
2
3  for i in lista_nom:
4      print(f"Bienvenido al equipo {i} ")
5
```

El tercer ejercicio consiste en tener una lista de notas de estudiantes, en mi caso puse 10 en todo para que sea muy fácil saber los resultados, y con esas notas sumar para conseguir el total y sacar la media de las notas pero sin usar las funciones de suma de Python en su lugar se debe usar un bucle for e imprimir los resultados

```

ejercicosProrg2 > acti2 > listaNotas.py > ...
1  mis_notas = [10, 10, 10, 10, 10]
2  sum_total = 0
3  for i in mis_notas:
4      sum_total = sum_total + i
5
6  promedio = sum_total / len(mis_notas)
7  print(f"La suma total de todas las notas es de: {sum_total}")
8  print(f"El promedio de las notas es de: {promedio}")

```

USANDO FUNCIONES Y LISTAS INVERTIDAS

Alejandro Hurtado – 10/07/2025 – 20:43

Los ejercicios de la clase de hoy se trataron de hacer diversas cosas con las listas una de ellas fue tomar una lista existente y darle la vuelta o invertir su contenido para posteriormente imprimirlo usando unas simples líneas de Código de una manera moderna:

```

ejercicosProrg2 > clase3 > listaInvertida.py > ...
1  #Crear una Lista invertia
2
3  def invertir_lista (list_var):
4      return list_var[::-1]
5
6  lista = [1, 2, 3]
7
8  print(invertir_lista(lista))
9
10


```


```

● PS C:\clone\prog2-alejandrohurtado> & C:/Users/HP/AppData/Local/Programs/Python/P
clone/prog2-alejandrohurtado/ejercicosProrg2/clase3/listaInvertida.py
[1, 2, 3]
● [3, 2, 1]
● PS C:\clone\prog2-alejandrohurtado>

```

También realizamos un contador de caracteres para una palabra (hola mundo) en este caso usando un bucle for y un contador simple:

```
ejercicosProrg2 > clase3 >  contadorElementos.py > ...  
1  #Contador de cuntas veces aparece un elementon de una lista  
2  
3  palabra = "hola mundo"  
4  contador = 0  
5  
6  for i in palabra:  
7      if i == "o":  
8          contador = contador + 1  
9  print(f"El numero de veces que se repite la o es de: {contador}")
```

```
• PS C:\clone\prog2-alejandrohurtado> & C:/Users/HP/AppData/Local/Programs/Python/Python39-64/Python.exe C:\clone\prog2-alejandrohurtado\ejercicosProrg2\clase3\contadorElementos.py  
El numero de veces que se repite la o es de: 2  
• PS C:\clone\prog2-alejandrohurtado> 
```



UNIVERSIDAD PRIVADA
DOMINGO SAVIO

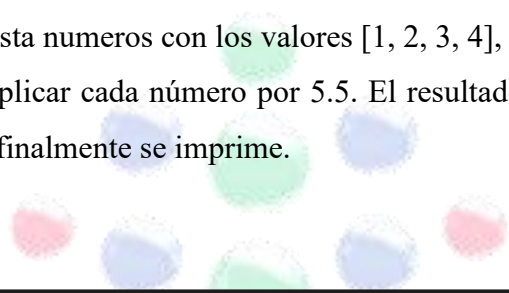
PROBANDO DIFERENTES FUNCIONES DE PYTHON

Alejandro Hurtado Rodas – 10/07/2025 – 21:35

Continuando con la clase probamos diferentes funciones así como vimos que son las funciones recursivas y como usarlas con las factoriales para aprender su funcionamiento:

Ejemplo 1:

En este código se muestran dos formas de manipular listas en Python. Primero, se crea una lista vacía llamada `numeros`. Con el método `append`, se agregan elementos al final de la lista, y con `insert` se añade un valor (15) en una posición específica (índice 1). Luego, en la segunda parte del código, se redefine la lista `numeros` con los valores [1, 2, 3, 4], y se usa `map` junto con una función `lambda` para multiplicar cada número por 5.5. El resultado se convierte en lista y se guarda en `duplicados`, que finalmente se imprime.



```
# Creamos una lista vacía
numeros = []

# append agrega al final
numeros.append(10)
numeros.append(20)
print("Usando append:", numeros) # [10, 20]

# insert agrega en una posición específica
numeros.insert(1, 15) # insertar en índice 1 el valor 15
print("Usando insert:", numeros) # [10, 15, 20]

# Queremos duplicar cada número-----
numeros = [1, 2, 3, 4]
duplicados = list(map(lambda x: x * 5.5, numeros))
print("Usando map:", duplicados) # [2, 4, 6, 8]
```

Ejemplo 2:

Este código define una función llamada `contar_elemento` que recibe una lista y un elemento a buscar. La función recorre la lista usando un bucle `for` y, por cada vez que encuentra el elemento buscado, incrementa un contador. Al final, retorna cuántas veces aparece dicho elemento. Luego se realizan pruebas utilizando `assert` para comprobar que la función devuelve los resultados correctos en distintos casos, incluyendo listas de números, cadenas, valores booleanos y listas vacías. Si todas las pruebas se cumplen, se imprime un mensaje confirmando que la función funciona correctamente.

```
def contar_elemento(lista, elemento_buscado):
    contador = 0
    for elemento in lista:
        if elemento == elemento_buscado:
            contador += 1
    return contador

# -----
# ✏️ Casos de prueba con assert
# -----

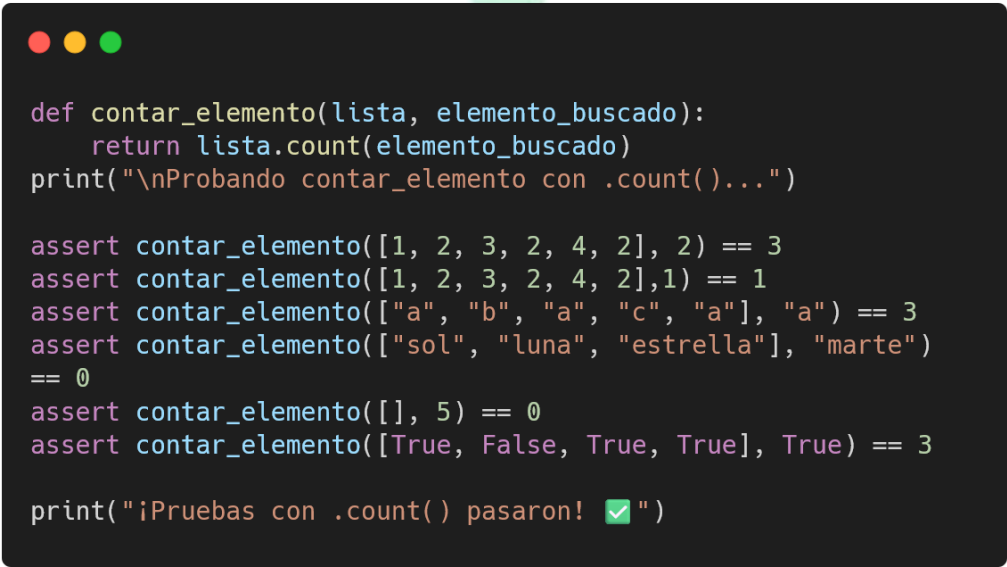
print("\nProbando contar_elemento...")

assert contar_elemento([1, 2, 3, 2, 4, 2], 2) == 3
assert contar_elemento(["a", "b", "a", "c", "a"], "a") == 3
assert contar_elemento(["sol", "luna", "estrella"], "martes") == 0
assert contar_elemento([], 5) == 0
assert contar_elemento([True, False, True, True], True) == 3

print("¡Pruebas para contar_elemento pasaron! ✅")
```

Ejemplo 3

Este código define una versión simplificada de la función `contar_elemento` utilizando el método incorporado `.count()` de las listas en Python. Esta función retorna directamente la cantidad de veces que aparece un elemento específico en la lista. A continuación, se realizan varias pruebas con `assert` para verificar que la función funciona correctamente en distintos casos: listas de números, cadenas, listas vacías y valores booleanos. Si todas las pruebas se cumplen sin errores, se imprime un mensaje indicando que las pruebas con `.count()` pasaron exitosamente.

A screenshot of a code editor with a dark background and light-colored text. The code defines a function `contar_elemento` that uses `lista.count(elemento_buscado)` to count occurrences. Below the function definition, there are several `assert` statements testing the function with various inputs: a list of numbers, a list of strings, an empty list, and a list of booleans. Finally, a `print` statement outputs a success message with a green checkmark icon.

```
def contar_elemento(lista, elemento_buscado):  
    return lista.count(elemento_buscado)  
print("\nProbando contar_elemento con .count()...")  
  
assert contar_elemento([1, 2, 3, 2, 4, 2], 2) == 3  
assert contar_elemento([1, 2, 3, 2, 4, 2], 1) == 1  
assert contar_elemento(["a", "b", "a", "c", "a"], "a") == 3  
assert contar_elemento(["sol", "luna", "estrella"], "marte")  
== 0  
assert contar_elemento([], 5) == 0  
assert contar_elemento([True, False, True, True], True) == 3  
  
print("¡Pruebas con .count() pasaron! ✅")
```

Ejemplo 4:

La función **`encontrar_mayor`** recibe una lista de números y devuelve el valor más grande encontrado. Primero, verifica si la lista está vacía y, en ese caso, retorna `None`. Si no está vacía, asume que el primer elemento es el mayor y luego recorre el resto de la lista comparando cada elemento con el mayor actual. Si encuentra uno más grande, lo actualiza. Al finalizar, devuelve el número más grande de la lista. Se incluyen pruebas con `assert` para validar el comportamiento de la función en diferentes situaciones: listas con varios números, negativos, elementos repetidos, una lista vacía y una lista con un solo elemento. Si todas las pruebas son correctas, se imprime un mensaje confirmando que pasaron exitosamente.

```

def encontrar_mayor(lista_numeros):
    # Caso especial: lista vacía
    if not lista_numeros:
        return None

    # Paso 1: El primer "campeón"
    mayor_temporal = lista_numeros[0]

    # Paso 2-4: Recorrer la lista para buscar al más grande
    for elemento in lista_numeros:
        if elemento > mayor_temporal:
            mayor_temporal = elemento

    # Paso 5: Devolver el campeón
    return mayor_temporal

# -----
# 🛠️ Casos de prueba con assert
# -----

print("Probando encontrar_mayor...")

assert encontrar_mayor([1, 5, 3, 9, 2]) == 9
assert encontrar_mayor([-10, -5, -3, -20]) == -3
assert encontrar_mayor([7, 7, 7, 7]) == 7
assert encontrar_mayor([]) == None # lista vacía
assert encontrar_mayor([42]) == 42 # un solo elemento

print("¡Pruebas para encontrar_mayor pasaron! ✅")

```

DOMINGO SAVIO

Ejemplo 5:

Este código define una función recursiva llamada factorial que calcula el factorial de un número entero no negativo. Si el número es negativo, lanza una excepción ValueError con un mensaje de error. Si el número es 0 o 1, retorna 1 como caso base. En otros casos, la función se llama a sí misma multiplicando n por el factorial de n - 1. Luego se realizan pruebas con assert para asegurar que la función devuelve los resultados correctos para ciertos valores. Finalmente, el programa solicita al usuario un número entero mediante input, e intenta calcular e imprimir su factorial. Si el usuario ingresa un número inválido (como uno negativo o texto), se captura la excepción y se muestra un mensaje de error.

```

def factorial(n):
    if n < 0:
        raise ValueError("❌ El factorial no está definido
para números negativos.")
    elif n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

assert factorial(0) == 1
assert factorial(1) == 1
assert factorial(5) == 120
assert factorial(6) == 720
print("¡Pruebas de factoriales pasaron! ✅")

# Solicita un número al usuario
try:
    numero = int(input("Ingrese un número entero para calcular
su factorial: "))
    resultado = factorial(numero)
    print(f"El factorial de {numero} es: {resultado}")
except ValueError as e:
    print(e)

```

Ejemplo 6:

Este código define una función llamada factorial que calcula el factorial de un número entero no negativo utilizando un enfoque iterativo. Si el número ingresado es negativo, se lanza una excepción ValueError con un mensaje de error. Para calcular el factorial, se inicializa una variable resultado en 1 y luego se multiplica sucesivamente por cada número desde 2 hasta n usando un bucle for. Al final, se retorna el resultado. El programa también solicita al usuario que ingrese un número entero y muestra el factorial correspondiente. Si se ingresa un valor no válido, el error es capturado y mostrado al usuario.

```
def factorial(n):
    if n < 0:
        raise ValueError("❌ El factorial no está definido
para números negativos.")

    resultado = 1
    for i in range(2, n + 1):
        resultado *= i
    return resultado

# Solicita un número al usuario
try:
    numero = int(input("Ingrese un número entero para calcular
su factorial: "))
    resultado = factorial(numero)
    print(f"El factorial de {numero} es: {resultado}")
except ValueError as e:
    print(e)
```



UNIVERSIDAD PRIVADA
DOMINGO SAVIO

Ejemplo 7:

Este programa calcula y muestra el factorial de un número entero no negativo, junto con los pasos de la multiplicación. La función factorial utiliza la función `math.prod` para calcular el producto de todos los números del 1 al `n`, y lanza un `ValueError` si el número es negativo. La función `mostrar_factorial` imprime visualmente la operación que se realiza para calcular el factorial (por ejemplo, $1 \times 2 \times 3 \times 4 = 24$). Si el número es negativo, muestra un mensaje de error. En la parte final, se pide al usuario un número entero, y si la entrada es válida, se muestra el cálculo del factorial paso a paso. Si el usuario ingresa un valor no válido, se muestra un mensaje de advertencia.

```
import math

def factorial(n):
    if n < 0:
        raise ValueError("❌ El factorial no está definido
para números negativos.")
    return 1 if n == 0 else math.prod(range(1, n + 1))

def mostrar_factorial(n):
    if n < 0:
        print("❌ El factorial no está definido para números
negativos.")
        return

    pasos = ' x '.join(str(i) for i in range(1, n + 1)) if n >
0 else "1"
    resultado = factorial(n)
    print(f"{pasos} = {resultado}")

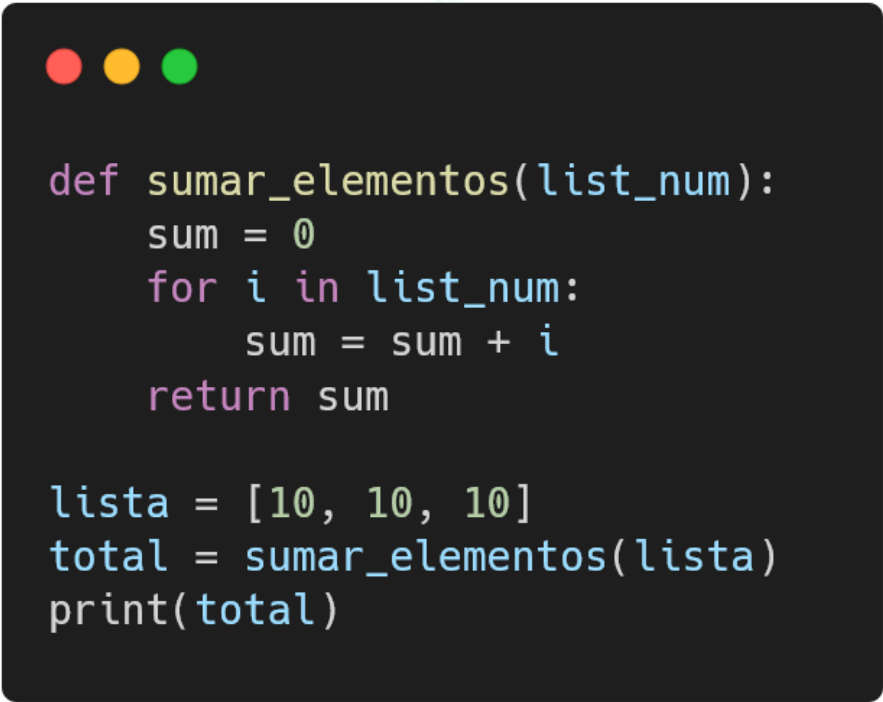
# -----
# Ejecución interactiva
# -----
try:
    numero = int(input("Ingrese un número entero para calcular
su factorial: "))
    mostrar_factorial(numero)
except ValueError:
    print("❌ Entrada inválida. Por favor ingresa un número
entero.")
```

OPERACIONES CON LAS LISTAS

códigos sencillos para ver el funcionamiento de las listas en diversos escenarios:

Sumar Elementos

Una función simple para sumar los elementos de una lista y poder imprimir el resultado:

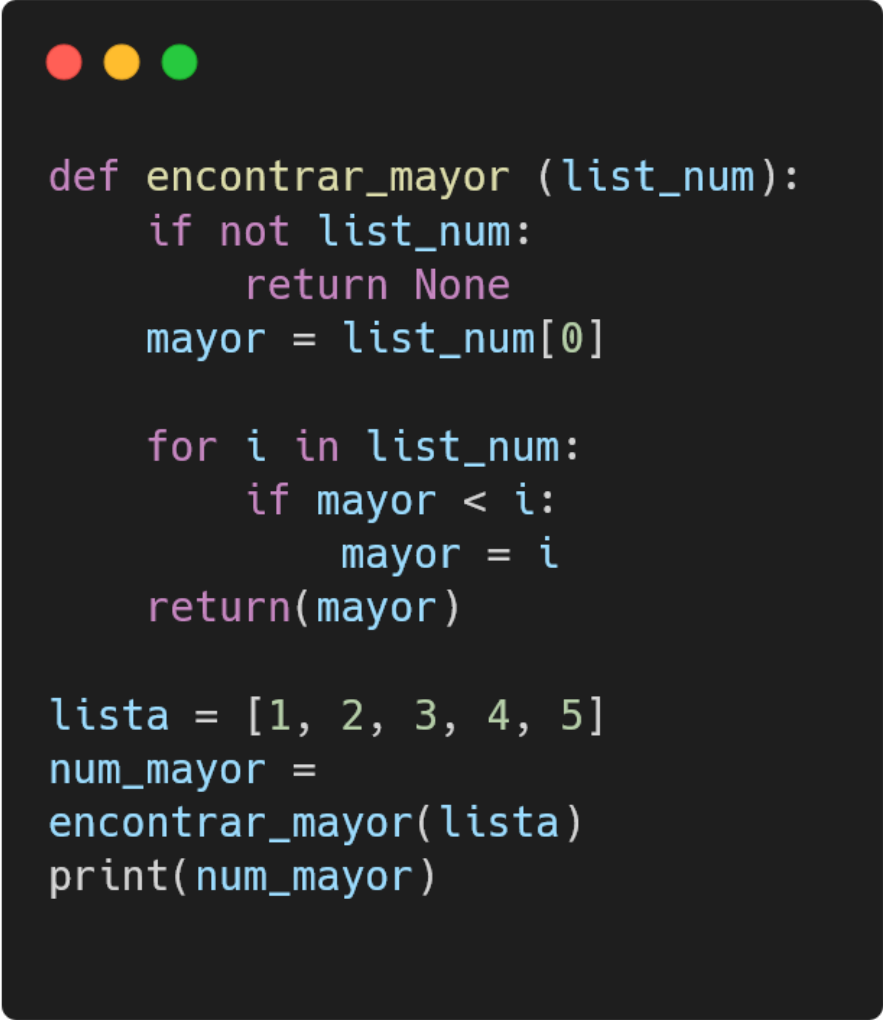


```
def sumar_elementos(list_num):  
    sum = 0  
    for i in list_num:  
        sum = sum + i  
    return sum  
  
lista = [10, 10, 10]  
total = sumar_elementos(lista)  
print(total)
```

Una función simple donde se usa un bucle for para recorrer la lista de elementos y cada vez que pase por uso se sume y guarde en la variable sum que se inicializo en 0 y retorne el valor total de la suma

Encontrar Mayor

función para encontrar el numero mayor en una lista:



```
def encontrar_mayor (list_num):  
    if not list_num:  
        return None  
    mayor = list_num[0]  
  
    for i in list_num:  
        if mayor < i:  
            mayor = i  
    return(mayor)  
  
lista = [1, 2, 3, 4, 5]  
num_mayor =  
encontrar_mayor(lista)  
print(num_mayor)
```

Tiene distintas condiciones por ejemplo en caso de que la lista este vacía retorna None y al inicializar la lista en el índice 0 no importa si se añaden numero negativos a la lista los tomara en cuenta y retornara siempre el numero mayor de la lista

Pruebas lógicas de funcionamiento

demostración del correcto funcionamiento de estas funciones usando el assert para diversas pruebas:

```
print("Probando encontrar mayor")
assert encontrar_mayor([1, 9, 1, 8, 3, 7]) == 9
assert encontrar_mayor([-1, -9, -2, -8]) == -1
assert encontrar_mayor([42, 42, 42]) == 42
assert encontrar_mayor([]) == None #
Prueba del caso especial
assert encontrar_mayor([5]) == 5
print("¡Pruebas para encontrar_mayor pasaron! ✅")

# Casos de Prueba con assert
print("\nProbando sumar_elementos...")
assert sumar_elementos([1, 2, 3, 4, 5]) == 15
assert sumar_elementos([10, -2, 5]) == 13
assert sumar_elementos([]) == 0 #
¡Importante probar con una lista vacía!
assert sumar_elementos([100]) == 100
print("¡Pruebas para sumar_elementos pasaron! ✅")
```

BUSQUEDA BINARIA

La **búsqueda binaria** es un método rápido para encontrar un elemento en una **lista ordenada**. Funciona dividiendo la lista a la mitad en cada paso y comparando el valor buscado con el número del medio. Dependiendo del resultado, se busca en la mitad izquierda o derecha hasta encontrar el valor o confirmar que no está. Es más eficiente que revisar uno por uno.

```
import math

def busqueda_binaria(lista, valor_a_buscar):
    izquierda = 0
    derecha = len(lista) - 1

    while izquierda <= derecha:
        medio = math.floor((izquierda + derecha) / 2)

        print(f"Iteración:")
        print(f"  Izquierda = {izquierda}, Derecha = {derecha}, Medio = {medio}")
        print(f"  lista[{medio}] = {lista[medio]}")

        if lista[medio] == valor_a_buscar:
            print(f"  ¡Encontrado! El valor {valor_a_buscar} está en el índice {medio}.")
            return medio
        elif valor_a_buscar > lista[medio]:
            print(f"  {valor_a_buscar} > {lista[medio]} → Buscamos en la mitad derecha")
            izquierda = medio + 1
        else:
            print(f"  {valor_a_buscar} < {lista[medio]} → Buscamos en la mitad izquierda")
            derecha = medio - 1
        print("-" * 50)

    print(f"El valor {valor_a_buscar} no se encuentra en la lista.")
    return None

# Lista ordenada de ejemplo
lista = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]

# Buscar un número que está
busqueda_binaria(lista, 23)

print("\n" + "=" * 50 + "\n")

# Buscar un número que no está
busqueda_binaria(lista, 80)
elementos([]) == 0 # ¡Importante probar con una lista vacía!
assert sumar_elementos([100]) == 100
print("¡Pruebas para sumar_elementos pasaron! ✅")
```