

UNIVERSIDAD PRIVADA DOMINGO SAVIO



PROYECTO EN TINKERCAD

Materia: Programación II

Docente: Jimmy Nataniel Requena LLorentty

Estudiantes:

- Adaniel Balderrama Orellana

Santa Cruz- Bolivia

```
edad de votacion.py > ...
1  edad_str = input("Bienvenido a la elecciones de nuestro proximo tirano, saqueador: Ingrese su edad... ")
2  edad = int(edad_str)
3  if edad >=18:
4      print ("Eres mayor de edad, puedes votar por el Capitan Lara :)")
5  elif edad >= 13:
6      print ("Lo lamento, no puedes votar eres menor de edad")
7  elif edad >=5:
8      print ("Eres un niño o niña.")
9  else :
10     print ("No puedes votar...")
11
12  print ("Adaniel Balderrama")

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicass python> & C:/Users/danny/AppData/Local/Programs/Python/Python313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/practicass python/edad de votacion.py"
Bienvenido a la elecciones de nuestro proximo tirano, saqueador: Ingrese su edad... 18
Eres mayor de edad, puedes votar por el Capitan Lara :)
Adaniel Balderrama
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicass python> |
```

En este aprendemos de manera creativa el uso del (if, elif, else) el código se enfoca en poder ingresar la edad, según a nuestra edad podremos votar con unos textos divertidos.

```
buque for.py X
buque for.py > ...
1  print("Contando hasta 3(sin incluirlo):")
2  for numero in range(3):
3      print(numero)
4  print("\nRecorriendo un string:")
5  nombre = "PYTHON"
6  for letra in nombre:
7      print(letra)
8
9  print ("Adaniel Balderrama")

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS  Python + v [ ]

PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicass python> & C:/Users/danny/AppData/Local/Programs/Python/Python313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/practicass python/buque for.py"
Contando hasta 3(sin incluirlo):
0
1
2

Recorriendo un string:
P
Y
T
H
O
N
Adaniel Balderrama
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicass python>
```

Aquí aplicamos el tema de los bucles exactamente el for, el código contará según el número que ingresemos, como también imprimirá la palabra letra por letra.

```
bluque while.py X
bluque while.py > ...
1  contador =0
2  print ("\nBucle while")
3  while contador <10:
4      print(f"Contador es: {contador}")
5      contador = contador + 1
6  print("¡Bucle while terminado!")
7  print ("Adaniel Balderrama")

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicas python> & C:/Users/danny/AppData/Local/Programs/Python/Python313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/practicas python/bluque while.py"

Bucle while
Contador es: 0
Contador es: 1
Contador es: 2
Contador es: 3
Contador es: 4
Contador es: 5
Contador es: 6
Contador es: 7
Contador es: 8
Contador es: 9
¡Bucle while terminado!
Adaniel Balderrama
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicas python>
```

Aquí aplicamos el tema de los bucles exactamente el while, el código contará según el número que ingresemos. En este caso es <10.

```
edad cine.py X
edad cine.py > ...
1  edad_str = input("Bienvenido al cine, ¿cuál es tu edad?: ")
2  edad = int(edad_str)
3
4  if edad < 0:
5      print("Edad no válida. Por favor, ingresa un número positivo.")
6  elif edad >= 18:
7      print("¡Puedes ver películas clasificadas R!")
8  elif edad >= 13:
9      print("Puedes ver películas clasificadas PG-13.")
10 else:
11     print("Te recomendamos películas clasificadas G o PG.")
12 print ("Adaniel Balderrama")
13

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicas python> & C:/Users/danny/AppData/Local/Programs/Python/Python313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/practicas python/edad cine.py"
Bienvenido al cine, ¿cuál es tu edad?: 18
¡Puedes ver películas clasificadas R!
Adaniel Balderrama
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicas python>
```

En este código recordamos el uso del (if, elif, else) pero en esta ocasión lo usamos para pedirnos ingresar nuestra edad y según a lo ingresado nos recomendará qué categorías se recomienda para ver.

```
tabla de multiplicar.py > ...
1  num_tabla = int(input("Ingrese un numero para ver su tabla de multiplicar :"))
2  print(f"--- Tabla del {num_tabla}--- ")
3  for i in range(1,11):
4      resultado = num_tabla * i
5      print(f"{num_tabla} x {i} = {resultado}")
6  print("Adaniel Balderrama")

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicass python> & C:/U
n313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/practicass p
Ingrese un numero para ver su tabla de multiplicar :15
--- Tabla del 15---
15 x 1 = 15
15 x 2 = 30
15 x 3 = 45
15 x 4 = 60
15 x 5 = 75
15 x 6 = 90
15 x 7 = 105
15 x 8 = 120
15 x 9 = 135
15 x 10 = 150
Adaniel Balderrama
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicass python> |
```

Este código Python genera tablas de multiplicar. Pide al usuario el número deseado, almacenando en la variable `num_tabla`. Luego, un bucle `for` con `range` itera para calcular y mostrar la tabla completa utilizando f-strings para un formato claro.

```
numero_secreto.py > ...
1  # Paso 1: Definir el número secreto
2  numero_secreto = 7
3
4  # Paso 2: Pedir al usuario que adivine
5  adivinanza = int(input("Adivina el número secreto entre 1 y 10: "))
6
7  # Paso 3: Bucle hasta que lo adivine
8  while adivinanza != numero_secreto:
9      if adivinanza < numero_secreto:
10         print("Demasiado bajo. Intenta de nuevo.")
11     else:
12         print("Demasiado alto. Intenta de nuevo.")
13
14     # Volver a pedir el número
15     adivinanza = int(input("Adivina otra vez: "))
16
17 # Paso 4: Mensaje final
18 print(f"¡Correcto! El número era {numero_secreto}. 🎉")
19 print("Adaniel Balderrama")

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

n313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/practicass python
Adivina el número secreto entre 1 y 10: 8
Demasiado alto. Intenta de nuevo.
Adivina otra vez: 6
Demasiado bajo. Intenta de nuevo.
Adivina otra vez: 7
¡Correcto! El número era 7. 🎉
Adaniel Balderrama
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicass python> ^B|
```

Este programa de adivinanza define un `numero_secreto`. Utiliza un bucle `while` para repetir la solicitud de adivinanza (`input` y `int`) hasta que el usuario acierte. Dentro del bucle, usa un `if-else` para dar pistas ("demasiado alto" o "bajo"), demostrando control de flujo.

```
1 # Ejercicio: Verificador de Edad para Películas usando Funciones
2
3 def obtener_clasificacion_pelicula(edad_persona):
4     """
5     Función que determina qué tipo de películas puede ver una persona según su edad.
6
7     Parámetro:
8     | edad_persona (int): La edad de la persona
9
10    Retorna:
11    | str: Mensaje con la clasificación de película recomendada
12    """
13
14    # Validar que la edad sea válida
15    if edad_persona < 0 or edad_persona > 120:
16        return "Edad no válida."
17
18    # Lógica de clasificación por edad
19    if edad_persona < 13:
20        return "Te recomendamos películas G (General Audiences)."
```

Este programa modulariza la lógica del verificador de edad utilizando **funciones**: `obtener_clasificacion_pelicula` (con parámetros y retorno) y `main`. Introduce manejo de errores con `try-except` para entradas no numéricas, y una función `probar_múltiples_edades` para demostración, evidenciando **reutilización de código** y **estructura de programa**.

```
refactorizacion.py > ...
1 # Ejemplo de código a refactorizar:
2 # Rectangulo 1
3 base1 = 10
4 altura1 = 5
5 area1 = base1 * altura1
6 print(f"El area del rectangulo 1 ({base1}x{altura1}) es: {area1}")
7
8 # Rectangulo 2
9 base2 = 7
10 altura2 = 3
11 area2 = base2 * altura2
12 print(f"El area del rectangulo 2 ({base2}x{altura2}) es: {area2}")
13
14 # Definir función para calcular área
15 def calcular_area_rectangulo(base, altura):
16     return base * altura
17
18 # Mostrar resultado
19 def mostrar_area_rectangulo(numero, base, altura):
20     area = calcular_area_rectangulo(base, altura)
21     print(f"El area del rectangulo {numero} ({base}x{altura}) es: {area}")
22
23 # Ahora por rectángulo
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS Python

```
n313/python.exe "d:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicas python\refactorizacion.py"
El area del rectangulo 1 (10x5) es: 50
El area del rectangulo 2 (7x3) es: 21
El area del rectangulo 1 (10x5) es: 50
El area del rectangulo 2 (7x3) es: 21
Adaniel Balderrama
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicas python>
```

Este ejemplo demuestra la **refactorización** del código para calcular el área de rectángulos. Inicialmente repetitivo, se introduce una **función `calcular_area_rectangulo`** para reutilizar la lógica y otra **función `mostrar_area_rectangulo`** para la presentación. Esto mejora la **legibilidad, mantenibilidad y eficiencia**, reduciendo la duplicación de código.

```
promedio de las notas.py > ...
1  # Crear una lista con notas numericas
2  mis_notas = [85.5, 92, 78, 88.5, 95, 82]
3  # Inicializar variable para la suma
4  suma_total = 0
5  # Usar bucle (variable) mis_notas: list total sin usar sum()
6  for nota in mis_notas:
7      suma_total += nota
8  # Calcular el promedio
9  promedio = suma_total / len(mis_notas)
10 # Imprimir resultados de forma clara
11 print(f"Suma total de las notas: {suma_total}")
12 print(f"Promedio de las notas: {promedio: .2f}")
13 print(f"La nota promedio de Adaniel: {promedio: .1f}")
14
15 print ("Adaniel Balderrama")

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicasython> & C:\Users\adaniel\AppData\Local\Microsoft\Windows\Inet
n313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/practic
Suma total de las notas: 521.0
Promedio de las notas: 86.83
La nota promedio de Adaniel: 86.8
Adaniel Balderrama
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicasython>
```

Este código calcula el promedio de un **vector (lista)** de notas numéricas. Utiliza un bucle **for** para iterar y sumar los elementos de la lista **mis_notas**. Luego, divide la **suma_total** por la cantidad de elementos (**len**) para obtener el promedio, mostrando operaciones fundamentales con listas.

```
Lista de comidas favoritas.py > ...
1  Lista_de_comida_favoritas = ["Charquecan", "Pique macho", "Sopa de mani", "Silpancho"]
2  print(Lista_de_comida_favoritas)
3  Mi_comida_favorita = Lista_de_comida_favoritas [0]
4  print(f"Mi comida favorita es: {Mi_comida_favorita}")
5
6
7  print(f"Adaniel Balderrama-FIN DEL PROGRAMA")

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicasython> & C:\Users\adaniel\AppData\Local\Microsoft\Windows\Inet
on313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/practicasython
['Charquecan', 'Pique macho', 'Sopa de mani', 'Silpancho']
Mi comida favorita es: Charquecan
Adaniel Balderrama-FIN DEL PROGRAMA
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicasython>
```

Este código demuestra correctamente el uso de **listas** en Python para almacenar elementos. Se inicializa la lista y se accede al **primer elemento ('Charquecan')** utilizando el índice **[0]**, asignándole a **Mi_comida_favorita**. Esto ilustra la indexación de listas para obtener valores específicos.

```
listahobbys.py > ...
1  lista_hobbys = ['musica', 'juegos', 'amigos', 'mi pareja']
2  primer_hobby = lista_hobbys [0]
3  print (f"Mi priner hobby es: {primer_hobby}")
4
5  segundo_hobby = lista_hobbys [1]
6  print (f"Mi segundo hooby es: {segundo_hobby}")
7
8  hobby_favorito = lista_hobbys [3]
9  print (f"Mi hobby favorito es : {hobby_favorito}")
10
11 print(f"Adaniel Balderrama-FIN DEL PROGRAMA")

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicass python> & C:/Python313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/practicass python.py"
Mi priner hobby es: musica
Mi segundo hooby es: juegos
Mi hobby favorito es : mi pareja
Adaniel Balderrama-FIN DEL PROGRAMA
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicass python>
```

Este código reitera el uso de **listas** en Python para almacenar y organizar datos. Demuestra claramente cómo **acceder a elementos individuales de la lista utilizando sus índices** (0, 1, 3), mostrando la versatilidad de las listas para recuperar valores específicos.

```
listagrupos.py > ...
1  # 1. Crear una lista con nombres de estudiantes
2  Death_Service = ['Adaniel_Balderrama', 'Carlos_Suarez', 'Marco_Barriga']
3  # 2 y 3. Usar bucle for para recorrer la lista e imprimir mensaje personalizado
4  for nombre in Death_Service:
5  | print(f"¡Bienvenido al equipo Death Services, {nombre}!")
6  print(f"Adaniel Balderrama-FIN DEL PROGRAMA")

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicass python> & C:/Python313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/practicass python.py"
¡Bienvenido al equipo Death Services, Adaniel_Balderrama!
¡Bienvenido al equipo Death Services, Carlos_Suarez!
¡Bienvenido al equipo Death Services, Marco_Barriga!
Adaniel Balderrama-FIN DEL PROGRAMA
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicass python>
```

Este código muestra la **iteración sobre una lista de nombres** (`Death_Service`) usando un bucle `for`. Por cada nombre en la lista, imprime un mensaje personalizado, demostrando cómo recorrer y **procesar individualmente cada elemento** de una colección.

```
notas.py > ...
1 # Accediendo a Elementos (Leer un "Cajón"): Usamos el nombre de la lista seguido de corchetes con índice deseado.
2 notas_parciales = [80,95,73,60,88]
3 primera_nota=notas_parciales[0] # Índice 0 para el PRIMER elemento
4 print(f"La primera nota fue: {primera_nota}") # Imprimir 80
5 tercera_nota = notas_parciales[2] # Índice 2 para el TERCER elemento
6 print(f"La tercera nota fue: {tercera_nota}") # Imprimir 73
7
8 # Modificando Elementos (Cambiar el contenido de un "cajón"):
9 print(f"Lista original: {notas_parciales}")
10 # Supongamos que se recalifico el 4to (índice 3)
11 notas_parciales[3]=65 # Asignamos un nuevo valor al índice 3
12 print(f"Lista modificada: {notas_parciales}")
13
14 # Obteniendo el Tamaño (len()):
15 cantidad_de_notas = len(notas_parciales)
16 print (f"Tenemos un total de {cantidad_de_notas} notas.") #Imprimira 5
17
18 print(f"Adaniel Balderrama-FIN DEL PROGRAMA")
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS Python +

```
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicas python> & C:/Users/danny/AppData/Local/Programs/Python/Python313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/practicas python/notas.py"
La primera nota fue: 80
La tercera nota fue: 73
Lista original: [80, 95, 73, 60, 88]
Lista modificada: [80, 95, 73, 65, 88]
Tenemos un total de 5 notas.
Adaniel Balderrama-FIN DEL PROGRAMA
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicas python>
```

Este código profundiza en la manipulación de **listas**. Muestra cómo **acceder a elementos por índice** (0 para el primero, 2 para el tercero), **modificar un elemento específico** mediante su índice (`notas_parciales[3]=65`), y cómo obtener el **tamaño de la lista** utilizando la función `len()`, ilustrando operaciones fundamentales.

```
vector misterioso.py > ...
1 # Lista secreta (vector misterioso)
2 vector_misterioso = [3, 7, 2, 9, 5] # Puedes cambiar los números si deseas
3
4 # Puedes hacer preguntas como estas:
5 print(len(vector_misterioso)) # ¿Cuántos elementos tiene la lista?
6 print(vector_misterioso[0]) # ¿Cuál es el primer número?
7 print(vector_misterioso[2]) # ¿Cuál es el tercer número?
8 print(vector_misterioso[-1]) # ¿Cuál es el último número?
9 print(vector_misterioso[0] > 10) # ¿El primer número es mayor a 10?
10 print(sum(vector_misterioso)) # ¿Cuál es la suma de todos los números?
11
12 # BONUS: Recorre la lista y muestra todos los números (una vez descubiertos)
13 for numero in vector_misterioso:
14     print(numero)
15 print(f"Adaniel Balderrama-FIN DEL PROGRAMA")
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

```
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicas python> & C:/Users/danny/AppData/Local/Programs/Python/Python313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/practicas python/vector misterioso.py"
5
3
2
5
False
26
3
7
2
9
5
Adaniel Balderrama-FIN DEL PROGRAMA
```

Este código explora diversas operaciones con **listas (vectores)**. Demuestra cómo obtener su **longitud** (`len`), acceder al **primer elemento** (`[0]`), al **tercero** (`[2]`), y al **último** (`[-1]`). También muestra una **comparación de valores** (`>`) y la **suma total de elementos** (`sum()`), finalizando con un bucle `for` para **recorrer y mostrar cada elemento**.


```
problema1.py > ...
1 def sumar_elementos_lista(lista_numeros):
2     # 1. Inicialización: Crea una "caja" para la suma (una variable) y ponle el valor 0.
3     # Llamémosla acumulador_suma.
4     acumulador_suma = 0
5
6     # 2. Iteración y Repetición: Recorre cada número en la lista.
7     # Por cada número, súmalo a tu acumulador_suma.
8     for numero in lista_numeros:
9         acumulador_suma += numero # Esto es equivalente a acumulador_suma = acumulador_suma + numero
10
11     # 5. Resultado: El valor final en acumulador_suma es tu respuesta.
12     return acumulador_suma
13
14 # --- Ejemplos de uso ---
15
16 # Ejemplo 1: Lista simple
17 numeros_ejemplo_1 = [5, 2, 8]
18 resultado_1 = sumar_elementos_lista(numeros_ejemplo_1)
19 print(f"La suma de los elementos en {numeros_ejemplo_1} es: {resultado_1}") # Salida: La suma de los elementos en [5,
20
21 # Ejemplo 2: Lista con más números
22 numeros_ejemplo_2 = [10, 4, 7, 1]
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS Python + v

PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicas python> & C:/Users/danny/AppData/Local/Programs/Python/Python313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/practicas python/problema1.py"

La suma de los elementos en [5, 2, 8] es: 15
La suma de los elementos en [10, 4, 7, 1] es: 22
La suma de los elementos en [] es: 0
La suma de los elementos en [1, -5, 3, -2] es: -3
Adaniel Balderrama-FIN DEL PROGRAMA

Este código define una **función `sumar_elementos_lista`** que calcula la suma de números en una lista. Demuestra la **inicialización de un acumulador** (**`acumulador_suma`**) y el uso de un bucle **`for`** para **iterar y acumular valores**. Ilustra la reutilización de funciones y el concepto de **iteración para agregación**.

```
problema2.py > encontrar_numero_mas_grande
1 def encontrar_numero_mas_grande(lista_numeros):
2
3     # 1. Suposición Inicial: Asume que el primer elemento de la lista es el más grande.
4     # ¡Cuidado! ¿Qué pasa si la lista está vacía?
5     if not lista_numeros: # Verifica si la lista está vacía
6         print("La lista está vacía, no se puede encontrar el número más grande.")
7         return None # Retorna None para indicar que no hay un número más grande
8
9     mayor_temporal = lista_numeros[0] # Asigna el primer elemento como el mayor temporal
10
11     # 2. Iteración: Recorre los elementos restantes de la lista.
12     # Comenzamos desde el segundo elemento (índice 1) porque el primero ya lo usamos.
13     for i in range(1, len(lista_numeros)):
14         elemento_actual = lista_numeros[i]
15
16         # 3. Comparación: Compara este elemento con tu mayor_temporal.
17         # Si el elemento actual es más grande, actualiza mayor_temporal.
18         if elemento_actual > mayor_temporal:
19             mayor_temporal = elemento_actual
20
21     # 5. Resultado: El valor final en mayor_temporal es el número más grande.
22     return mayor_temporal
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS Python

En la lista [12, 5, 23, 8], el número más grande es: 23
En la lista [30, 10, 20, 5], el número más grande es: 30
En la lista [1, 7, 3, 40], el número más grande es: 40
En la lista [-10, -3, -15, -1], el número más grande es: -1
En la lista [99], el número más grande es: 99
La lista está vacía, no se puede encontrar el número más grande.
En la lista [], el número más grande es: None
Adaniel Balderrama-FIN DEL PROGRAMA

Este código define una **función `encontrar_numero_mas_grande`** que halla el mayor valor en una lista. Incluye **validación para listas vacías**, y utiliza una **variable `mayor_temporal`** inicializada con el primer elemento. Un **bucle `for`** itera desde el segundo elemento, **comparando cada uno** para actualizar **`mayor_temporal`** si encuentra un valor superior, demostrando un algoritmo de búsqueda.

```
problema3.py > contar_apariciones
1 def contar_apariciones(lista, elemento_buscado):
2
3     # 2. Iteración: Recorrer cada elemento de la lista, uno por uno.
4     # 3. Comparación: En cada paso, pregunta: "¿Es este elemento igual al que estoy buscando?"
5     # Si la respuesta es "Sí", incrementa tu contador en 1.
6     # Si la respuesta es "No", no hagas nada.
7     for elemento in lista:
8         if elemento == elemento_buscado:
9             contador += 1 # Incrementa el contador si el elemento coincide
10
11     # 4. Condición de Fin: Continúa hasta haber revisado todos los elementos. (El bucle 'for' se encarga de esto)
12     # 5. Resultado: El número final en tu contador es la respuesta.
13     return contador
14
15 # --- Ejemplos de uso ---
16
17 # Ejemplo 1: Contar números
18 mi_lista_numeros = [1, 2, 3, 2, 4, 2, 5, 2]
19 elemento_a_contar_1 = 2
20 veces_aparece_1 = contar_apariciones(mi_lista_numeros, elemento_a_contar_1)
21 print(f"El número {elemento_a_contar_1} aparece {veces_aparece_1} veces en {mi_lista_numeros}")
22 # Salida: El número 2 aparece 4 veces en [1, 2, 3, 2, 4, 2, 5, 2]
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS Python

```
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicas python> & C:\Users\danny\AppData\Local\on313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/practicas python/problema3.py"
El número 2 aparece 4 veces en [1, 2, 3, 2, 4, 2, 5, 2]
'manzana' aparece 3 veces en ['manzana', 'banana', 'manzana', 'pera', 'manzana', 'kiwi']
'amarillo' aparece 0 veces en ['rojo', 'verde', 'azul']
'10' aparece 0 veces en []
Adaniel Balderrama-FIN DEL PROGRAMA
```

Este código define la **función contar_apariciones** para determinar cuántas veces un **elemento_buscado** aparece en una **lista**. Inicializa un **contador en cero** y utiliza un **bucle for** para **iterar sobre cada elemento**. Si hay coincidencia, el contador se incrementa, demostrando un algoritmo de **búsqueda y conteo** simple pero efectivo.

```
problema4.py > ...
25 # --- Ejemplos de uso ---
26
27 # Ejemplo 1: Lista de números
28 numeros = [10, 20, 30, 40, 50]
29 numeros_invertidos = invertir_lista_nueva(numeros)
30 print(f"Lista original: {numeros}")
31 print(f"Lista invertida: {numeros_invertidos}")
32 # Salida: Lista original: [10, 20, 30, 40, 50]
33 # Lista invertida: [50, 40, 30, 20, 10]
34
35 # Ejemplo 2: Lista de cadenas de texto
36 letras = ["a", "b", "c", "d"]
37 letras_invertidas = invertir_lista_nueva(letras)
38 print(f"Lista original: {letras}")
39 print(f"Lista invertida: {letras_invertidas}")
40 # Salida: Lista original: ['a', 'b', 'c', 'd']
41 # Lista invertida: ['d', 'c', 'b', 'a']
42
43 # Ejemplo 3: Lista con un solo elemento
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

```
on313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/practicas python/problema4.py"
Lista original: [10, 20, 30, 40, 50]
Lista invertida: [50, 40, 30, 20, 10]
Lista original: ['a', 'b', 'c', 'd']
Lista invertida: ['d', 'c', 'b', 'a']
Lista original: ['único']
Lista invertida: ['único']
Lista original: []
Lista invertida: []
Adaniel Balderrama-FIN DEL PROGRAMA
```

Este código define la **función invertir_lista_nueva** que crea una nueva lista con los elementos de la original en **orden inverso**. Utiliza un **bucle for con range** para **iterar de forma descendente** (desde el último índice hasta el primero), añadiendo cada elemento a la nueva lista. También maneja el caso de **listas vacías**.

```
Binaria.py > búsqueda_binaria
3 def búsqueda_binaria(lista_ordenada, clave):
24     # Si el bucle termina, return -1
25     return -1
26
27 # Prueba tu función:
28 if __name__ == "__main__":
29     lista_ordenada = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]
30
31     print("\nProbando búsqueda_binaria...")
32
33     # Test cases using assert
34     assert búsqueda_binaria(lista_ordenada, 23) == 5, "Test Falló: 23 debería estar en el índice 5"
35     assert búsqueda_binaria(lista_ordenada, 91) == 9, "Test Falló: 91 debería estar en el último índice"
36     assert búsqueda_binaria(lista_ordenada, 2) == 0, "Test Falló: 2 debería estar en el primer índice"
37     assert búsqueda_binaria(lista_ordenada, 3) == -1, "Test Falló: 3 no debería existir"
38     assert búsqueda_binaria(lista_ordenada, 100) == -1, "Test Falló: 100 no debería existir (fuera de rango mayor)"
39     assert búsqueda_binaria([], 5) == -1, "Test Falló: Búsqueda binaria en lista vacía"
40     assert búsqueda_binaria([1], 1) == 0, "Test Falló: Búsqueda binaria con un solo elemento encontrado"
41     assert búsqueda_binaria([1], 99) == -1, "Test Falló: Búsqueda binaria con un solo elemento no encontrado"

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS Python
PS D:\ingeniería en sistemas\Ingeniería de sistemas\tercer semestre\Programacion II\practicas python> & C:/Users/danny/AppData/Local/Programs/Python/Python313/python.exe "d:/ingeniería en sistemas/Ingeniería de sistemas/tercer semestre/Programacion II/practicas python/Binaria.py"

Probando búsqueda_binaria...
¡Pruebas para búsqueda_binaria pasaron! ✓

Ejemplos adicionales:
Buscando 12 en [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]: 3
Buscando 5 en [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]: 1
Buscando 70 en [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]: -1
Adaniel Balderrama-FIN DEL PROGRAMA
PS D:\ingeniería en sistemas\Ingeniería de sistemas\tercer semestre\Programacion II\practicas python> |
```

Este código implementa la **búsqueda binaria**, un algoritmo eficiente para encontrar un elemento en una **lista_ordenada**. Reduce el espacio de búsqueda a la mitad en cada paso al comparar la **clave** con el elemento **medio**. Utiliza **izquierda**, **derecha** y **medio** para ajustar el rango, y **assert** para **validar la correcta funcionalidad** en diversos escenarios, incluyendo listas vacías.

```
secuencial.py > búsqueda_lineal
14 if __name__ == "__main__":
15     mi_lista_desordenada = [10, 5, 42, 8, 17, 30, 25]
16
17     print("Probando búsqueda_lineal...")
18
19     # Test cases using assert
20     assert búsqueda_lineal(mi_lista_desordenada, 42) == 2, "Test Falló: 42 debería estar en el índice 2"
21     assert búsqueda_lineal(mi_lista_desordenada, 10) == 0, "Test Falló: 10 debería estar en el índice 0 (Al inicio)"
22     assert búsqueda_lineal(mi_lista_desordenada, 25) == 6, "Test Falló: 25 debería estar en el índice 6 (Al final)"
23     assert búsqueda_lineal(mi_lista_desordenada, 99) == -1, "Test Falló: 99 no debería existir"
24     assert búsqueda_lineal([], 5) == -1, "Test Falló: Búsqueda en lista vacía debería retornar -1"
25
26     print("¡Pruebas para búsqueda_lineal pasaron! ✓")
27
28     # Optional: Additional test cases
29     print("\nEjemplos adicionales:")
30     print(f"Buscando 8 en {mi_lista_desordenada}: {búsqueda_lineal(mi_lista_desordenada, 8)}")
31     print(f"Buscando 17 en {mi_lista_desordenada}: {búsqueda_lineal(mi_lista_desordenada, 17)}")

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS Python +
PS D:\ingeniería en sistemas\Ingeniería de sistemas\tercer semestre\Programacion II\practicas python> & C:/Users/danny/AppData/Local/Programs/Python/Python313/python.exe "d:/ingeniería en sistemas/Ingeniería de sistemas/tercer semestre/Programacion II/practicas python/secuencial.py"

Probando búsqueda_lineal...
¡Pruebas para búsqueda_lineal pasaron! ✓

Ejemplos adicionales:
Buscando 8 en [10, 5, 42, 8, 17, 30, 25]: 3
Buscando 17 en [10, 5, 42, 8, 17, 30, 25]: 4
Buscando 100 en [10, 5, 42, 8, 17, 30, 25]: -1
Adaniel Balderrama
```

Este código implementa la **búsqueda lineal**, un algoritmo que recorre cada elemento de una lista de forma secuencial. Un bucle **for** itera a través de los índices; si encuentra la **clave**, retorna su índice. Si no la halla tras revisar toda la lista, retorna -1, demostrando un enfoque directo para la búsqueda.

```
Lista desordenada.py > ...
8
9 # 3. Prueba tu función con assert
10 mi_lista_desordenada = [10, 5, 42, 8, 17, 30, 25]
11 print("Probando busqueda_lineal...")
12
13 assert busqueda_lineal(mi_lista_desordenada, 42) == 2
14 assert busqueda_lineal(mi_lista_desordenada, 10) == 0 # Al inicio
15 assert busqueda_lineal(mi_lista_desordenada, 25) == 6 # Al final
16 assert busqueda_lineal(mi_lista_desordenada, 99) == -1 # No existe
17 assert busqueda_lineal([], 5) == -1 # Lista vacía
18
19 print("¡Pruebas para busqueda_lineal pasaron! ✅")
20 print("Adaniel Balderrama - FIN DEL PROGRAMA")
21
22 # clase06_busquedas.py (continuación)
23 # 2. Definir la función busqueda_binaria
24 def busqueda_binaria(lista_ordenada, clave):

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS >

on313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/practicas python/Lista des
Probando busqueda_lineal...
¡Pruebas para busqueda_lineal pasaron! ✅
Adaniel Balderrama - FIN DEL PROGRAMA

Probando busqueda_binaria...
¡Pruebas para busqueda_binaria pasaron! ✅
Adaniel Balderrama - FIN DEL PROGRAMA

Realizando el experimento del caos...
Búsqueda binaria de '5' en lista desordenada devolvió: 4
Adaniel Balderrama - FIN DEL PROGRAMA
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicas python>
```

Este código compara dos algoritmos de búsqueda: **lineal** y **binaria**. La **búsqueda lineal** (`busqueda_lineal`) es simple, recorriendo cada elemento secuencialmente, útil para listas desordenadas. La **búsqueda binaria** (`busqueda_binaria`) es más eficiente, dividiendo la lista ordenada a la mitad en cada paso. El "experimento del caos" crucialmente demuestra que la **búsqueda binaria falla** en **listas desordenadas**, enfatizando la **precondición de ordenamiento** para su correcto funcionamiento y eficiencia.

```

22
23 def Ordenamiento_burbuja(lista):
24     n = len(lista)
25     for i in range(n - 1):
26         hubo_intercambio = False
27         for j in range(n - 1 - i):
28             if lista[j] > lista[j + 1]:
29                 lista[j], lista[j + 1] = lista[j + 1], lista[j]
30                 hubo_intercambio = True
31         if not hubo_intercambio:
32             break
33
34 if __name__ == "__main__":
35     numeros = [6, 3, 8, 2, 5]
36     print("Antes:", numeros)
37     Ordenamiento_burbuja(numeros)
38     print("Después:", numeros)
39
40 print ("Adaniel Balderrama")
41

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

Python

PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicas python> & C:/Users/danny/AppData/Local/Programs/Python/Python313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/practicas python/Ordenamiento.p
Antes: [6, 3, 8, 2, 5]
Después: [2, 3, 5, 6, 8]
Adaniel Balderrama
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicas python>

Ordenamiento burbuja.py > ...

```

1 def ordenamiento_burbuja(lista):
2     n = len(lista)
3     for i in range(n):
4         for j in range(0, n - i - 1):
5             if lista[j] > lista[j + 1]:
6                 lista[j], lista[j + 1] = lista[j + 1], lista[j]
7     return lista
8
9 # Prueba con impresión
10 lista_a_ordenar = [54, 34, 25, 12, 22, 11, 90]
11 print(f"Lista original: {lista_a_ordenar}")
12 ordenamiento_burbuja(lista_a_ordenar)
13 print(f"Lista ordenada: {lista_a_ordenar}")
14
15 # Casos de prueba
16
17 # caso 1: lista desordenada
18 lista1 = [6, 3, 8, 2, 5]
19 ordenamiento_burbuja(lista1)
20 assert lista1 == [2, 3, 5, 6, 8]

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicas python> & C:/Users/danny/AppData/Local/Programs/Python/Python313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/practicas python/Ordenamiento.p
Lista original: [54, 34, 25, 12, 22, 11, 90]
Lista ordenada: [11, 12, 22, 25, 34, 54, 90]
¡Todas las pruebas pasaron!
Adaniel Balderrama
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\practicas python>