Bucles en Python (while y for)

1. Introducción a los bucles

1.1 ¿Qué son los bucles?

Un **bucle** (o loop) es una estructura que permite **repetir un bloque de código múltiples veces** sin tener que escribirlo una y otra vez.

En la vida cotidiana usamos bucles constantemente:

- "Toma una manzana de la canasta mientras haya manzanas"
- "Repite este ejercicio 10 veces"
- "Mientras no llegues a casa, sigue caminando"

En programación, los bucles son fundamentales porque permiten:

- Procesar datos repetitivos de forma eficiente.
- Automatizar tareas.
- Reducir código duplicado.
- Trabajar con colecciones de datos.

1.2 Ejemplo introductorio

Sin bucles (tedioso):

```
print("Hola")
print("Hola")
print("Hola")
print("Hola")
print("Hola")
```

Con bucles (elegante):

```
for i in range(5):
    print("Hola")
```

El resultado es el mismo, pero el segundo es mucho más limpio y escalable.

1.3 Tipos de bucles en Python

Python tiene dos tipos principales de bucles:

- **`while`** : Repite mientras una condición sea verdadera.
- **`for`** : Repite un número conocido de veces o sobre los elementos de una colección.

2. El bucle while (mientras)

2.1 Sintaxis del while

```
while condición:
    # Bloque de código que se repite mientras la condición sea True
    print("Esto se repite")
```

Cómo funciona:

1. Python evalúa la condición.
2. Si es `True`, ejecuta el bloque de código.
3. Vuelve al paso 1.
4. Si es `False`, sale del bucle.

2.2 Ejemplo simple: Contar hasta 5

```
contador = 1

while contador <= 5:
    print(f"Contador: {contador}")
    contador += 1  # Incrementar contador

print("¡Fin!")
```

Salida:

```
Contador: 1
Contador: 2
Contador: 3
Contador: 4
Contador: 5
¡Fin!
```

¿Qué sucede paso a paso?

1. `contador = 1` (inicializamos)
2. Verifica: `1 <= 5` ? Sí → Ejecuta el bloque
3. Imprime: `Contador: 1`

4. `contador += 1` → ahora `contador = 2`
5. Verifica: `2 <= 5` ? Sí → Ejecuta el bloque
6. (continúa hasta que contador sea 6)
7. Verifica: `6 <= 5` ? No → Sale del bucle

2.3 Punto importante: Evitar bucles infinitos

Un **bucle infinito** ocurre cuando la condición nunca se vuelve falsa:

```
contador = 1

while contador <= 5:
    print("Hola")  # ❌ PROBLEMA: contador nunca cambia
    # contador nunca se incrementa, así que el bucle nunca termina
```

Este código imprimirá "Hola" infinitamente. Para evitarlo, siempre asegúrate de que algo cambie dentro del bucle para que la condición eventualmente sea falsa.

Forma correcta:

```
contador = 1

while contador <= 5:
    print("Hola")
    contador += 1  # ✅ Ahora sí termina
```

2.4 Operadores de incremento y decremento

Python proporciona operadores útiles para cambiar variables:

```
# Incrementar
contador = 5
contador += 1  # contador = contador + 1 → 6
contador += 3  # contador = contador + 3 → 9

# Decrementar
numero = 10
numero -= 1  # numero = numero - 1 → 9
numero -= 2  # numero = numero - 2 → 7

# Multiplicar
valor = 4
valor *= 2  # valor = valor * 2 → 8

# Dividir
```

```
cantidad = 20
cantidad /= 4  # cantidad = cantidad / 4 → 5.0
```

2.5 Ejemplo: Contar hacia atrás

```
numero = 5

while numero > 0:
    print(numero)
    numero -= 1

print("¡Despegue!")
```

Salida:

```
5
4
3
2
1
¡Despegue!
```

2.6 Ejemplo: Sumar números del usuario

```
print("=== SUMADOR ===")
print("Ingresa números (digita 0 para terminar)")
print()

suma = 0

while True:
    numero = float(input("Ingresa un número: "))

    if numero == 0:
        break  # Salir del bucle

    suma += numero

print(f"La suma total es: {suma}")
```

Explicación:

- `while True` crea un bucle que se ejecuta indefinidamente.
- `break` detiene el bucle cuando el usuario digita 0.
- Acumulamos la suma con `suma += numero`.

2.7 Palabra clave: break

La palabra `break` **termina inmediatamente** el bucle:

```
contador = 1

while contador <= 10:
    if contador == 5:
        break  # Sale del bucle cuando contador llega a 5
    print(contador)
    contador += 1

print("Fin")
```

Salida:

```
1
2
3
4
Fin
```

2.8 Palabra clave: continue

La palabra `continue` **salta a la siguiente iteración** del bucle:

```
contador = 0

while contador < 5:
    contador += 1

    if contador == 3:
        continue  # Salta esta iteración

    print(contador)
```

Salida:

```
1
2
4
5
```

Nota que el 3 no se imprime porque `continue` salta esa iteración.

3. El bucle for (para)

3.1 Sintaxis del for con range

```
for variable in range(inicio, fin, paso):
    # Bloque de código que se repite
    print(variable)
```

El bucle `for` es especialmente útil cuando sabes exactamente cuántas veces quieres repetir algo.

3.2 Función range()

La función `range()` genera una secuencia de números:

```
# range(fin): Desde 0 hasta fin-1
for i in range(5):
    print(i)  # Imprime: 0, 1, 2, 3, 4
```

Salida:

```
0
1
2
3
4
```

```
# range(inicio, fin): Desde inicio hasta fin-1
for i in range(2, 7):
    print(i)  # Imprime: 2, 3, 4, 5, 6
```

Salida:

```
2
3
4
5
6
```

```
# range(inicio, fin, paso): Desde inicio hasta fin-1, aumentando de paso en paso
for i in range(0, 10, 2):
    print(i)  # Imprime: 0, 2, 4, 6, 8
```

Salida:

```
0
2
4
6
8
```

```
# Contar hacia atrás
for i in range(5, 0, -1):
    print(i)  # Imprime: 5, 4, 3, 2, 1
```

Salida:

```
5
4
3
2
1
```

3.3 Ejemplo: Tabla de multiplicar

```
numero = 5

print(f"Tabla de multiplicar del {numero}")
print()

for i in range(1, 11):
    resultado = numero * i
    print(f"{numero} × {i} = {resultado}")
```

Salida:

```
Tabla de multiplicar del 5

5 × 1 = 5
5 × 2 = 10
5 × 3 = 15
5 × 4 = 20
5 × 5 = 25
5 × 6 = 30
5 × 7 = 35
5 × 8 = 40
5 × 9 = 45
5 × 10 = 50
```

3.4 Ejemplo: Suma de números del 1 al N

```
n = int(input("¿Hasta qué número deseas sumar? "))

suma = 0

for numero in range(1, n + 1):
    suma += numero

print(f"La suma de 1 hasta {n} es: {suma}")
```

Si el usuario ingresa 10:

```
¿Hasta qué número deseas sumar? 10
La suma de 1 hasta 10 es: 55
```

3.5 Ejemplo: Patrón visual (triángulo)

```
n = 5

for i in range(1, n + 1):
    print("*" * i)
```

Salida:

```
*
**
***
****
*****
```

¿Cómo funciona?

- `"*" * 1` → `"*"`
- `"*" * 2` → `"**"`
- `"*" * 3` → `"***"`
- Y así sucesivamente...

4. Bucles anidados

4.1 ¿Qué son los bucles anidados?

Un **bucle anidado** es un bucle dentro de otro bucle. Son muy útiles para trabajar con estructuras bidimensionales.

4.2 Ejemplo simple: Tabla de multiplicar completa

```
print("=== TABLA DE MULTIPLICAR (1-5) ===")
print()

for i in range(1, 6):
    for j in range(1, 6):
        resultado = i * j
        print(f"{i} × {j} = {resultado:2d}", end="  ")
    print()  # Salto de línea después de cada fila
```

Salida:

```
=== TABLA DE MULTIPLICAR (1-5) ===

1 × 1 =  1  1 × 2 =  2  1 × 3 =  3  1 × 4 =  4  1 × 5 =  5
2 × 1 =  2  2 × 2 =  4  2 × 3 =  6  2 × 4 =  8  2 × 5 = 10
3 × 1 =  3  3 × 2 =  6  3 × 3 =  9  3 × 4 = 12  3 × 5 = 15
4 × 1 =  4  4 × 2 =  8  4 × 3 = 12  4 × 4 = 16  4 × 5 = 20
5 × 1 =  5  5 × 2 = 10  5 × 3 = 15  5 × 4 = 20  5 × 5 = 25
```

¿Cómo funciona?

- El bucle externo (i) controla las filas.
- El bucle interno (j) controla las columnas.
- Para cada valor de i, j recorre completamente su rango.

4.3 Ejemplo: Rectángulo con asteriscos

```
ancho = 5
alto = 3

for fila in range(alto):
    for columna in range(ancho):
        print("*", end=" ")
    print()  # Salto de línea
```

Salida:

```
* * * * *
* * * * *
* * * * *
```

4.4 Ejemplo: Pirámide

```
n = 5

for i in range(1, n + 1):
    # Espacios en blanco
    for j in range(n - i):
        print(" ", end="")
    # Asteriscos
    for j in range(i):
        print("*", end="")
    print()
```

Salida:

```
    *
   **
  ***
 ****
*****
```

5. Bucles con listas (introducción)

5.1 Recorrer una lista con for

Una de las principales utilidades del bucle `for` es recorrer **listas**:

```
frutas = ["manzana", "plátano", "naranja", "uva"]

for fruta in frutas:
    print(fruta)
```

Salida:

```
manzana
plátano
naranja
uva
```

¿Qué sucede?

- La variable `fruta` toma el valor de cada elemento de la lista, uno por uno.
- En la primera iteración: `fruta = "manzana"`

- En la segunda iteración: `fruta = "plátano"`
- Y así sucesivamente.

5.2 Ejemplo: Sumar todos los números de una lista

```
numeros = [5, 10, 15, 20, 25]
suma = 0

for numero in numeros:
    suma += numero

print(f"La suma total es: {suma}")
```

Salida:

```
La suma total es: 75
```

5.3 Obtener el índice con enumerate()

A veces necesitas saber la posición (índice) de cada elemento:

```
frutas = ["manzana", "plátano", "naranja"]

for indice, fruta in enumerate(frutas):
    print(f"Posición {indice}: {fruta}")
```

Salida:

```
Posición 0: manzana
Posición 1: plátano
Posición 2: naranja
```

5.4 Crear una lista usando append()

```
numeros = []

for i in range(1, 6):
    numeros.append(i * 2)

print(numeros)
```

Salida:

```
[2, 4, 6, 8, 10]
```

6. Ejercicio guiado: Menú repetitivo

Crearemos un programa que muestre un menú y permita al usuario seleccionar opciones hasta que decida salir.

6.1 Objetivo

Crear un programa que:

1. Muestre un menú con opciones.
2. Permita al usuario elegir una opción.
3. Ejecute la acción correspondiente.
4. Repita el menú hasta que el usuario seleccione "Salir".

6.2 Planificación

```
1. Mostrar menú (while True)
2. Pedir opción al usuario
3. Validar opción
4. Ejecutar acción según opción
5. Repetir hasta que seleccione "Salir"
```

6.3 Código completo

```
# Menú de opciones repetitivo

print("=" * 40)
print("BIENVENIDO AL MENÚ INTERACTIVO".center(40))
print("=" * 40)
print()

opcion = 0

while opcion != 4:
    print("-" * 40)
    print("OPCIONES DISPONIBLES")
    print("-" * 40)
    print("1. Saludar")
    print("2. Despedirse")
    print("3. Ver información")
    print("4. Salir")
```

```
        print("-" * 40)

        # Pedir opción
        try:
            opcion = int(input("Elige una opción (1-4): "))
        except ValueError:
            print("❌ Error: Debes ingresar un número")
            continue

        # Ejecutar acción según opción
        if opcion == 1:
            print("👋 ¡Hola! Bienvenido al programa")
        elif opcion == 2:
            print("👋 ¡Hasta luego! Que tengas un excelente día")
        elif opcion == 3:
            print("ℹ Este es un programa educativo sobre bucles en Python")
        elif opcion == 4:
            print("✅ Saliendo del programa...")
        else:
            print("❌ Opción no válida. Por favor, elige entre 1 y 4")

        print()

print("=" * 40)
print("GRACIAS POR USAR EL PROGRAMA".center(40))
print("=" * 40)
```

6.4 Explicación

- **`while opcion != 4`** : Repite hasta que el usuario seleccione salir.
- **`try-except`** : Maneja errores si el usuario ingresa algo que no es un número.
- **`continue`** : Si hay un error, salta al inicio del bucle.
- **`if-elif-else`** : Ejecuta la acción correspondiente.

6.5 Salida del programa

```
========================================
  BIENVENIDO AL MENÚ INTERACTIVO
========================================

----------------------------------------
OPCIONES DISPONIBLES
----------------------------------------
1. Saludar
2. Despedirse
3. Ver información
4. Salir
----------------------------------------
Elige una opción (1-4): 1
```

```
👋 ¡Hola! Bienvenido al programa

----------------------------------------
OPCIONES DISPONIBLES
----------------------------------------
1. Saludar
2. Despedirse
3. Ver información
4. Salir
----------------------------------------
Elige una opción (1-4): 4
✅ Saliendo del programa...

========================================
   GRACIAS POR USAR EL PROGRAMA
========================================
```

7. Ejercicio de evaluación

7.1 Enunciado

Crea un programa que **gestione una lista de tareas** con las siguientes funcionalidades:

1. **Ver tareas**: Muestra todas las tareas guardadas.
2. **Agregar tarea**: Permite al usuario agregar una nueva tarea.
3. **Eliminar tarea**: Permite al usuario eliminar una tarea por su número.
4. **Contar tareas**: Muestra cuántas tareas hay en total.
5. **Salir**: Termina el programa.

Requisitos:

- Usa un bucle `while` para el menú principal.
- Usa bucles `for` para mostrar las tareas.
- Usa la lista para almacenar las tareas.
- Valida que las opciones sean válidas.
- Muestra mensajes claros en cada acción.

Ejemplo de ejecución:

```
=== GESTOR DE TAREAS ===

1. Ver tareas
2. Agregar tarea
3. Eliminar tarea
4. Contar tareas
```

```
5. Salir

Elige una opción: 2
Ingresa la nueva tarea: Estudiar Python
✅ Tarea agregada: "Estudiar Python"

Elige una opción: 2
Ingresa la nueva tarea: Hacer ejercicio
✅ Tarea agregada: "Hacer ejercicio"

Elige una opción: 1
Tareas:
1. Estudiar Python
2. Hacer ejercicio

Elige una opción: 4
Total de tareas: 2

Elige una opción: 5
¡Hasta luego!
```

7.2 Pistas para resolver

- Crea una lista vacía: `tareas = []`
- Para agregar: usa `tareas.append(nueva_tarea)`
- Para eliminar: usa `tareas.pop(indice)`
- Para mostrar: usa un bucle `for` con `enumerate()`
- Para contar: usa `len(tareas)`

8. Solución del ejercicio de evaluación

```
# Gestor de tareas

print("=" * 40)
print("GESTOR DE TAREAS".center(40))
print("=" * 40)
print()

tareas = []
opcion = 0

while opcion != 5:
    print("-" * 40)
    print("MENÚ PRINCIPAL")
    print("-" * 40)
    print("1. Ver tareas")
```

```
        print("2. Agregar tarea")
        print("3. Eliminar tarea")
        print("4. Contar tareas")
        print("5. Salir")
        print("-" * 40)

        try:
            opcion = int(input("Elige una opción (1-5): "))
        except ValueError:
            print("❌ Error: Debes ingresar un número")
            print()
            continue

        # OPCIÓN 1: Ver tareas
        if opcion == 1:
            print()
            if len(tareas) == 0:
                print("📭 No hay tareas. ¡Agrega una!")
            else:
                print("📋 LISTA DE TAREAS")
                for indice, tarea in enumerate(tareas, start=1):
                    print(f"  {indice}. {tarea}")
            print()

        # OPCIÓN 2: Agregar tarea
        elif opcion == 2:
            print()
            nueva_tarea = input("Ingresa la nueva tarea: ")
            tareas.append(nueva_tarea)
            print(f"✅ Tarea agregada: \"{nueva_tarea}\"")
            print()

        # OPCIÓN 3: Eliminar tarea
        elif opcion == 3:
            print()
            if len(tareas) == 0:
                print("📭 No hay tareas para eliminar")
            else:
                print("📋 LISTA DE TAREAS")
                for indice, tarea in enumerate(tareas, start=1):
                    print(f"  {indice}. {tarea}")

                try:
                    numero = int(input("Ingresa el número de la tarea a eliminar: "))
                    if 1 <= numero <= len(tareas):
                        tarea_eliminada = tareas.pop(numero - 1)
                        print(f"✅ Tarea eliminada: \"{tarea_eliminada}\"")
                    else:
                        print("❌ Número no válido")
                except ValueError:
                    print("❌ Error: Debes ingresar un número")
            print()
```

```
    # OPCIÓN 4: Contar tareas
    elif opcion == 4:
        print()
        cantidad = len(tareas)
        print(f"📊 Total de tareas: {cantidad}")
        print()

    # OPCIÓN 5: Salir
    elif opcion == 5:
        print()
        print("✅ ¡Hasta luego! Que tengas un excelente día")
        print()

    # Opción no válida
    else:
        print()
        print("❌ Opción no válida. Por favor, elige entre 1 y 5")
        print()

print("=" * 40)
print("FIN DEL PROGRAMA".center(40))
print("=" * 40)
```

8.1 Explicación de la solución

- **`tareas = []`** : Lista vacía para almacenar tareas.
- **Opción 1**: Itera sobre la lista con `enumerate()` para mostrar número y tarea.
- **Opción 2**: Agrega una nueva tarea con `append()` .
- **Opción 3**: Elimina una tarea usando `pop()` después de validar el índice.
- **Opción 4**: Cuenta las tareas con `len()` .
- **`try-except`** : Maneja errores de entrada del usuario.

8.2 Alternativa más compacta (sin try-except)

```
tareas = []
opcion = 0

while opcion != 5:
    print("\n--- MENÚ GESTOR DE TAREAS ---")
    print("1. Ver tareas")
    print("2. Agregar tarea")
    print("3. Eliminar tarea")
    print("4. Contar tareas")
    print("5. Salir")

    opcion = int(input("Opción: "))

    if opcion == 1:
```

```
            if tareas:
                for i, tarea in enumerate(tareas, 1):
                    print(f"{i}. {tarea}")
            else:
                print("Sin tareas")
        elif opcion == 2:
            tareas.append(input("Tarea: "))
        elif opcion == 3 and tareas:
            numero = int(input("Número: "))
            if 1 <= numero <= len(tareas):
                tareas.pop(numero - 1)
        elif opcion == 4:
            print(f"Total: {len(tareas)}")
        elif opcion == 5:
            print("¡Adiós!")

    print("Fin")
```

9. Ejercicios adicionales para practicar

9.1 Ejercicio: Calculadora simple con menú

Crea un programa que:

- Muestre un menú con operaciones (+, -, ×, ÷).
- Pida dos números.
- Realice la operación.
- Repita hasta que el usuario seleccione "Salir".

9.2 Ejercicio: Búsqueda en lista

Crea un programa que:

- Tenga una lista de nombres predefinida.
- Pida al usuario un nombre a buscar.
- Muestre si está en la lista y en qué posición.
- Repita hasta que ingrese "salir".

9.3 Ejercicio: Sistema de calificaciones

Crea un programa que:

- Pida notas de estudiantes en un bucle.
- Almacene las notas en una lista.

- Calcule el promedio.
- Muestre la nota máxima y mínima.

9.4 Ejercicio: Juego: Adivina el número (mejorado)

Crea un programa que:

- Tenga un número secreto.
- Permita múltiples intentos.
- Diga si es mayor, menor o igual.
- Cuente los intentos.
- Permita jugar múltiples rondas.

9.5 Ejercicio: Fibonacci

Crea un programa que:

- Pida cuántos números de la serie Fibonacci mostrar.
- Use un bucle para generar la serie (1, 1, 2, 3, 5, 8, 13...).
- Guarde los números en una lista.
- Muestre la lista completa.

10. while vs for: Cuándo usar cada uno

Aspecto	while	for
Uso	Repetir mientras una condición sea verdadera	Repetir un número conocido de veces
Mejor para	Bucles con condición desconocida	Iterar sobre listas o rangos
Inicialización	Manual (fuera del bucle)	Automática
Control	Manual (debes cambiar la variable)	Automático (i aumenta)
Ejemplo	`while usuario != "salir":`	`for i in range(5):`

11. Comparación de ejemplo: while vs for

Contar del 1 al 5 con while:

```
contador = 1
while contador <= 5:
    print(contador)
    contador += 1
```

Contar del 1 al 5 con for:

```
for contador in range(1, 6):
    print(contador)
```

El resultado es idéntico, pero `for` es más simple y limpio para este caso.

12. Errores comunes con bucles

Error 1: Bucle infinito por olvido de incremento

```
contador = 1
while contador <= 5:
    print(contador)
    # ❌ PROBLEMA: contador nunca cambia
```

Solución:

```
contador = 1
while contador <= 5:
    print(contador)
    contador += 1  # ✅ Incrementar
```

Error 2: Confundir range(5) con range(1, 6)

```
# ❌ Imprime 0, 1, 2, 3, 4
for i in range(5):
    print(i)

# ✅ Imprime 1, 2, 3, 4, 5
for i in range(1, 6):
    print(i)
```

Error 3: Olvidar el dos puntos

```
while contador < 5  # ❌ Falta el dos puntos
    print(contador)
```

Solución:

```
while contador < 5:  # ✅ Dos puntos presente
    print(contador)
```

Error 4: Indentación incorrecta

```
for i in range(5):
print(i)  # ❌ Sin indentación
```

Solución:

```
for i in range(5):
    print(i)  # ✅ Indentado
```

Error 5: Modificar la lista mientras se itera

```
numeros = [1, 2, 3, 4, 5]

for numero in numeros:
    if numero == 3:
        numeros.remove(3)  # ❌ Evita modificar listas mientras iteras
```

Solución:

```
numeros = [1, 2, 3, 4, 5]

# Opción 1: Crear una copia
for numero in numeros[:]:
    if numero == 3:
        numeros.remove(3)

# Opción 2: Usar una nueva lista
numeros = [n for n in numeros if n != 3]
```

13. Casos de uso del mundo real

13.1 Procesamiento de datos

```
# Calcular promedio de calificaciones
calificaciones = [85, 90, 78, 92, 88]
suma = 0

for calificacion in calificaciones:
    suma += calificacion

promedio = suma / len(calificaciones)
print(f"Promedio: {promedio:.2f}")
```

13.2 Validación de entrada

```
# Pedir edad válida
edad = 0

while edad < 1 or edad > 120:
    try:
        edad = int(input("Ingresa tu edad (1-120): "))
    except ValueError:
        print("Debe ser un número válido")

print(f"Tu edad es: {edad}")
```

13.3 Búsqueda en datos

```
# Encontrar el nombre más largo
nombres = ["Ana", "Roberto", "Leo", "Alejandra"]
nombre_mas_largo = nombres[0]

for nombre in nombres:
    if len(nombre) > len(nombre_mas_largo):
        nombre_mas_largo = nombre

print(f"Nombre más largo: {nombre_mas_largo}")
```

14. Resumen de conceptos clave del PDF 3

Concepto	Descripción	Ejemplo
Bucle	Repetir código múltiples veces	`while` o `for`

Concepto	Descripción	Ejemplo
while	Repetir mientras condición sea verdadera	`while contador < 5:`
for	Repetir sobre elementos o rangos	`for i in range(5):`
range()	Generar secuencia de números	`range(1, 6)` → 1,2,3,4,5
break	Salir del bucle	`break`
continue	Saltar a siguiente iteración	`continue`
+=, -=	Operadores de incremento/decremento	`contador += 1`
Bucle anidado	Bucle dentro de otro	`for i in range(n): for j in range(n):`
enumerate()	Obtener índice y elemento	`for i, valor in enumerate(lista):`
append()	Agregar a lista	`lista.append(elemento)`
pop()	Eliminar de lista	`lista.pop(indice)`
len()	Longitud de lista	`len(lista)`

15. Preguntas frecuentes del PDF 3

P: ¿Cuándo uso while y cuándo for?

R: Usa `for` cuando sabes cuántas iteraciones quieres. Usa `while` cuando quieres repetir hasta que ocurra algo.

P: ¿Qué es un bucle infinito?

R: Un bucle que nunca termina porque su condición siempre es verdadera. Se evita asegurando que algo cambie.

P: ¿Puedo usar break y continue en while?

R: Sí, ambas palabras funcionan en `while` y `for`.

P: ¿Qué es range()?

R: Una función que genera números. `range(5)` genera 0,1,2,3,4. `range(1,6)` genera 1,2,3,4,5.

P: ¿Puedo modificar una lista mientras la recorro?

R: No es recomendable. Crea una copia o usa una nueva lista.

P: ¿Cuál es la diferencia entre range(5) y range(1,6)?

R: `range(5)` genera 0-4. `range(1,6)` genera 1-5. El inicio por defecto es 0, y el fin es exclusivo.

Conclusión

Felicidades por completar el PDF 3. Ahora entiendes:

- Cómo funcionan los bucles y por qué son importantes.
- La diferencia entre `while` y `for`.
- Cómo usar `range()` para generar secuencias.
- Las palabras clave `break` y `continue`.
- Cómo anidar bucles.
- Cómo iterar sobre listas.
- Operadores de incremento y decremento.
- Casos de uso reales de bucles.

En el siguiente PDF (Listas), aprenderás a trabajar profundamente con **colecciones de datos** usando listas, operaciones sobre ellas, y métodos útiles que hacen que Python sea poderoso para procesar información.