4. Sentencias de control

Resumen de Sentencias de Control en Python

Las sentencias de control (o **control de flujo**) definen el orden en que se ejecutan las instrucciones en un programa. Se dividen en tres tipos principales: **estructuras de selección (condiciones)** y **estructuras de iteración (bucles)**.

---

1. Estructuras de Selección (Condicionales)

Permiten que el programa tome **decisiones** y ejecute diferentes bloques de código según se cumpla o no una **condición**. La condición es una **comparación** que siempre resulta en un valor **Booleano** (`True` o `False`).

Operadores de Comparación

Se utilizan para crear las condiciones en las sentencias de control:

Operador	Significado	Ejemplo	Resultado
`>`	Mayor que	`A > B`	`True` si A es mayor que B
`<`	Menor que	`A < B`	`True` si A es menor que B
`==`	Igual a	`A == B`	`True` si A es igual a B
`>=`	Mayor o igual que	`A >= B`	`True` si A es mayor que B o igual a B
`<=`	Menor o igual que	`A <= B`	`True` si A es menor que B o igual a B
`!=`	Distinto a	`A != B`	`True` si A es distinto a B

Export to Sheets

Otros operadores importantes son `in` o `not in` , que se usan para verificar la pertenencia a una secuencia.

La sentencia `if`

Es la estructura de selección más común y básica.

- **Sintaxis:** Python

```
if condicion:
    # Bloque de código a ejecutar si la condición es True
```

- Si la condición se evalúa como `True`, el **código indentado** que le sigue se ejecuta. Si es `False`, ese bloque se omite.
- Dentro del bloque `if`, se pueden realizar operaciones, llamar a métodos de listas, etc..

`if ... else ...`

Amplía el `if` para definir una acción cuando la condición inicial es `False`.

- **Sintaxis:** Python

```
if condicion_1:
    # Código si condicion_1 es True
else:
    # Código si condicion_1 es False (no lleva condición)
```

- El bloque `else` **engloba todas las condiciones que no se cumplen** en el `if`.
- La indentación del `else` debe ser igual a la del `if` al que pertenece.

`if ... elif ... else`

Permite chequear **múltiples condiciones** de forma secuencial.

- **Sintaxis:** Python

```
if condicion_1:
    # Código si se cumple la condicion_1
elif condicion_2:
    # Código si NO se cumple la condicion_1, pero SÍ la condicion_2
elif condicion_3: # Se pueden añadir tantos elif como se quiera
    # ...
else:
    # Código si NO se cumple ninguna de las condiciones anteriores
(opcional)
```

- **El orden es importante**: Las condiciones se evalúan de arriba hacia abajo. En el momento en que una condición es `True`, su bloque de código se ejecuta, y el programa **se detiene** sin evaluar el resto de `elif` ni el `else`.
- El `else` es **opcional**.

Condiciones Múltiples y Anidadas

- **Condiciones Múltiples (`and`, `or`):** Se pueden incluir varias condiciones en una misma línea de `if` o `elif` usando los operadores lógicos.
  - `and`: **TODAS** las condiciones deben cumplirse para que sea `True`.
  - `or`: **ALGUNA** de las condiciones debe cumplirse para que sea `True`.
- **Condicionales Anidados:** Es una sentencia `if` dentro de otra sentencia `if` o `else`. Permiten construir lógicas más complejas, pero un **uso excesivo** puede hacer el código difícil de leer y mantener.

Ejemplo de Condición Múltiple:

Python

```
numero_peras = 11

# Se usan el operador 'and' y la sintaxis abreviada de rango
if numero_peras > 4 and numero_peras < 50:
    print('Tenemos muchas peras, ¿no te apetece comerte una?')
# Equivalente y más legible en Python:
# if 4 < numero_peras < 50:
```

Ejemplo de `if` Anidado:

Python

```
nevera_en_marcha = True
numero_peras = 11

if nevera_en_marcha: # Condición 1: ¿Está encendida?
    if 0 < numero_peras < 4: # Condición 1.1
        print('Añadimos peras a la lista')
    elif 4 < numero_peras < 50: # Condición 1.2
        print('Tenemos muchas peras, ¿no te apetece comerte una?')
else: # Condición 2: Si nevera_en_marcha es False
    print('La nevera está rota.')
```

---

2. Estructuras de Iteración (Bucles)

Permiten al programa **repetir** un bloque de código varias veces.

El bucle `while`

Ejecuta un bloque de código **mientras se cumpla una determinada condición**.

- **Sintaxis:** Python

```
while condición:
    # Bloque de código a ejecutar mientras la condición sea True
```

- El bucle se repite continuamente mientras la condición se evalúe como `True`. Cuando la condición se vuelve `False`, el bucle termina.
- Es importante asegurarse de que la condición **se vuelva `False`** en algún momento, para **evitar un bucle infinito**.
- Se usa típicamente cuando **no se sabe de antemano** cuántas veces se tendrá que ejecutar el bucle.
- Dentro de un bucle `while`, se pueden incluir otras estructuras de control, como `if`, `elif`, y `else`.

Ejemplo de `while`:

Python

```
limite_peras = 2
numero_peras = 11

# Mientras el número de peras sea mayor que el límite, el bucle continúa.
while numero_peras > limite_peras:
    print(f'Comiendo pera. Quedan {numero_peras}')
    # La variable se actualiza en cada iteración, acercándose al límite
    numero_peras -= 1

print(f'¡Fin de bucle! Quedan {numero_peras} peras. Es hora de ir a
comprar.')
```

---

3. Otras Estructuras de Control (Flujo de Funciones)

Estas estructuras permiten controlar el flujo de ejecución **dentro y fuera de funciones** o bucles.

- `break`: Detiene la ejecución del bucle actual y salta al código que sigue inmediatamente después del bucle.

- `continue`: Detiene la ejecución de la iteración actual del bucle y pasa a la siguiente iteración.
- `pass`: Es una operación nula, se usa cuando se requiere una sentencia sintácticamente, pero no se desea ejecutar ninguna acción (por ejemplo, en un `if` o una función aún no implementada).
- `try/except`: Permite al programa manejar errores (excepciones) dentro de una función y continuar ejecutando el código.

EJERCICIOS Y MAS

¡Claro! Aquí tienes un resumen completo de las **Sentencias de Control** estructurado como apuntes para un Jupyter Notebook. 📝🐍

Sentencias de Control

Las sentencias de control permiten definir el **flujo de ejecución** de un programa, determinando qué instrucciones se ejecutan y en qué orden. Se dividen principalmente en **estructuras de selección (condiciones)** y **estructuras de iteración (bucles)**.

---

1. Estructuras de Selección (`if`, `elif`, `else`)

Permiten al programa tomar **decisiones** basándose en si una **condición** es `True` o `False`. Usarás mucho las **comparaciones** y **valores booleanos**.

1.1 Sintaxis y Funcionamiento Básico

Componente	Estructura	Descripción
`if`	`if condicion1:`	La primera condición a evaluar. Es la **línea de condición**.
`elif`	`elif condicion2:`	Se evalúa **solo si** las condiciones `if` y `elif` anteriores fueron `False`. Puede haber varios `elif`.
`else`	`else:`	Se ejecuta si **ninguna** de las condiciones (`if` o `elif`) anteriores se cumplió. **No lleva condición**.

- **Reglas Cruciales:**
  - **Sangría y Dos Puntos (`:`):** La **línea de condición** siempre termina con dos puntos (`:`), y el **bloque de instrucciones** (la **línea de consecuencia**) debe ir **indentado** (con sangría).

- **Ejecución Secuencial:** El programa evalúa las condiciones de arriba abajo y se **detiene** en la primera que es `True`.
- **Uso Recomendado:** Es más eficiente usar la estructura `if...elif...else` que concatenar varios `if` independientes, ya que así se evita que el programa evalúe todos los bloques.

1.2 Ejemplos Básicos

Ejemplo 1: `if/elif/else`

Python

```
a = 30
b = 15
c = 5

if b < a:
    print(b, "es menor que", a)
elif b > a:
    print(f"{a} es menor que {b}")
else:
    print("Son iguales")
```

Resultado: `15 es menor que 30`.

Ejemplo 2: Usando `in` (Pertenencia)

Python

```
# Si estoy en la lista, puedo pasar. Si no, nada
nombre = "Marta Martínez"
lista_de_invitados = ["Marta Martínez", "Rosa Fernández", "Paquita Salas",
"María Pérez"]

if nombre in lista_de_invitados:
    print("Estás en la lista. Puedes pasar")
else:
    print("No puedes pasar. Asegúrate de que hayas escrito tu nombre con las
tildes.")
```

Resultado: `Estás en la lista. Puedes pasar`.

1.3 Sentencias Anidadas

Consiste en colocar una estructura de control (`if/elif/else`) **dentro** de otra. Esto permite chequear condiciones más específicas.

Ejemplo de `if` Anidados (Edad y Autorización)

Python

```
edad = 15
autorizacion = True

if edad >= 18:
    print("Sí puedes pasar")
elif 0 < edad < 18:
    if autorizacion:
        print("Puedes pasar, menos mal que has traído autorización")
    else:
        print("No puedes pasar sin autorización")
else:
    print("Introduce una edad correcta, no puede ser negativa")
```

Resultado: `Puedes pasar, menos mal que has traído autorización`.

1.4 Condiciones Múltiples

Se comprueban varias condiciones en la misma línea utilizando los operadores lógicos `and` y `or`.

- **`and`**: Requiere que **TODAS** las condiciones sean `True`.
- **`or`**: Requiere que **ALGUNA** de las condiciones sea `True`.

Ejemplo 3: Condiciones Múltiples (`and`)

Python

```
edad = 40

# Hay una beca que solo te la dan si tienes entre 30 y 50 años
# Opción 1 (and):
if edad >= 30 and edad <= 50:
    print("Puedes acceder a la beca (opción 1)")
else:
    print("No puedes acceder a la beca (opción 1)")

# Opción 2 (Sintaxis Abreviada de Rango):
if 30 <= edad <= 50:
```

```
    print("Puedes acceder a la beca (opción 2)")
else:
    print("No puedes acceder a la beca (opción 2)")
```

Resultado (ambas opciones): `Puedes acceder a la beca (opción X)`.

---

2. Estructuras de Iteración (Bucles)

Permiten ejecutar un **bloque de código repetidamente**.

2.1 Bucle `while`

El bucle `while` repite un bloque de instrucciones **mientras una condición se mantenga** `True`.

- **Regla Crucial:** Para evitar un **bucle infinito**, se debe incluir una instrucción dentro del bucle que **modifique la variable de control** o cambie la condición a `False` en algún momento.
- **Uso:** Es útil cuando **no sabemos de antemano** cuántas veces se debe repetir el bucle.

Estructura:

Python

```
variable_control = valor_inicial

while condicion:
    # bloque_instrucciones

    modifica_variable_control # ¡Necesario para evitar bucle infinito!
```

Ejemplo 4: Bucle `while` (Contador)

Python

```
tu_edad = 10

while tu_edad < 18:
    print("No puedes montar en moto")
    tu_edad += 1 # Modifica la variable de control
    print(f"Ha pasado un año, ahora tienes {tu_edad} años")
```

```
print("¡Ya puedes montar en moto!")
```

Resultado (últimas líneas): `No puedes montar en moto` -> `Ha pasado un año, ahora tienes 18 años` -> `¡Ya puedes montar en moto!`.

Uso de while True:

Se repite indefinidamente. Siempre debe usarse un break en su interior para detener el bucle cuando se cumpla una condición específica.

Ejemplo 5: Bucle `while True` (Menú)

Python

```
print("Menú Principal. 1. Saludar. 2. Mostrar información. 3. Salir")
while True:
    opcion = 3 # Simulamos la entrada del usuario

    if opcion == 3:
        print("Adiós")
        break # Detiene y sale del bucle

    # Si se selecciona otra opción, el bucle continúa
```

Resultado: `Menú Principal. 1. Saludar. 2. Mostrar información. 3. Salir` -> `Adiós`.

2.2 Control de Bucles

Son sentencias que permiten controlar el flujo de ejecución **dentro** de un bucle (`while` o `for`).

- **`break`**: **Corta el bucle por completo** y sale de él, continuando con el código que sigue al bucle.
  - *Uso común:* Salir de un bucle `while True` o de un bucle `while` antes de que se cumpla su condición.
- **`continue`**: **Se salta la iteración actual** del bucle y pasa directamente a la siguiente repetición.

Ejemplo 6: Uso de `continue`

Python

```
contador = 0
while contador < 5:
    num = -1 # Simulación de un número negativo

    if num < 0:
        print("Número negativo ignorado.")
        continue # El bucle salta a la siguiente iteración, sin ejecutar lo
de abajo

    print(f"Añadido {num} a la lista de números válidos") # Esta línea no se
ejecuta si num < 0
    contador += 1
```

Resultado (ejecución real): El `print` del final y el `contador += 1` no se ejecutarían cuando `num < 0`, saltando a la siguiente vuelta del bucle.

Estructuras de Iteración (Bucles)

El bucle **`while`** es una estructura de control que permite **repetir** un bloque de código **mientras** una condición específica se mantenga **`True`**.

Bucle `while`: Estructura y Funcionamiento

Concepto	Descripción
Objetivo	Repetir un conjunto de código mientras se cumpla la condición.
Condición de Parada	Para que el bucle deje de repetirse, la condición debe **cambiar a `False`**.
Riesgo	**Bucle Infinito**: Si no se incluye una instrucción que modifique la condición, el bucle se ejecutará indefinidamente.

Export to Sheets

Estructura Básica:

Python

```
variable_control = valor_inicial # 1. Variable de control

while condicion:               # 2. Mientras la condición sea True, se
ejecuta el bloque
    bloque_instrucciones
```

```
    modifica_variable_control  # 3. ¡Crucial! Modifica la variable
(contador/acumulador)
                               # para que la condición cambie a False.
```

Ejemplo 1: Contador El bucle repite la acción hasta que la variable de control (`edad`) cambia y hace que la condición (`edad < 18`) sea `False` .

Python

```
edad = 15

while edad < 18:
    print("No puedes montar moto")
    print("ha pasado un año")
    edad += 1  # Esto es un contador: edad = edad + 1

print("Ya puedes montar moto")
```

Control de Bucles: `break` y `continue`

Son declaraciones que permiten controlar el flujo de ejecución **dentro** de un ciclo, decidiendo cuándo parar o cuándo saltar una repetición.

`break`

La sentencia **`break`** **corta el bucle por completo** y sale de él, independientemente de si la condición principal del `while` es `True` .

- **Uso Común:** Para detener un bucle **`while True`** o para salir inmediatamente de un bucle cuando se cumple una condición específica.

Ejemplo 2: Menú con `while True` y `break` El bucle se repite hasta que la opción `3` se selecciona, momento en el que `break` detiene la ejecución.

Python

```
print("1. Saludar 2. Mostrar 3.salir")

while True:
    opcion = int(input("Elige una opción del menú: "))

    if opcion == 1:
```

```
        print("Hola, que tal")
    elif opcion == 2:
        print("Estos son tus datos")
    elif opcion == 3:
        print("Adios")
        break  # Rompe el bucle

    # El 'else' también rompe el bucle si la opción no es válida
    else:
        break
```

`continue`

La sentencia **`continue` se salta la iteración actual** del bucle y regresa al inicio para chequear la condición del `while` y comenzar la siguiente repetición.

Ejemplo 3: Uso de `continue` Si el usuario ingresa un número negativo, `continue` ignora el resto del código dentro del bucle para esa vuelta (es decir, no incrementa el `contador` ni muestra el mensaje de "Número contado"), volviendo directamente al inicio.

Python

```
contador = 0

while contador < 5:
    numero = int(input("Dime un numero positivo: "))

    if numero < 0:
        print("Numero negativo ignorado")
        continue # El bucle salta aquí a la siguiente iteración

    # Este código solo se ejecuta si el número NO es negativo
    contador += 1
    print("Numero contado ", contador)
```