5. Sentencias de control (bucles for)

Estructuras de Control en Python: `for`, `continue`, `break`, `pass` y `try-except`

Estructura	Propósito Principal	Sintaxis Básica	Uso Común
Bucle `for`	Iterar sobre una **secuencia** (lista, tupla, cadena, rango) para ejecutar un bloque de código por cada elemento1.	`for variable_control in iterable:`	Procesar listas de datos, aplicar el mismo cálculo a múltiples elementos2.
`continue`	Saltar la iteración **actual** del bucle y pasar a la siguiente.	`if condicion: continue`	Ignorar datos defectuosos o nulos en un proceso sin detenerlo3.
`break`	Detener la ejecución del bucle **completamente**.	`if condicion: break`	Buscar un elemento específico y parar tan pronto como se encuentra4.
`pass`	Actuar como un **marcador de posición** que no hace nada.	`if condicion: pass`	Definir temporalmente estructuras vacías o incompletas sin causar error de sintaxis5.
`try-except`	**Manejar errores** (excepciones) en el código para evitar que el programa se detenga abruptamente.	`try: codigo except Error:`	Procesar datos "sucios" (con fallos) sin interrumpir el análisis6.

---

1. El Bucle `for`

El bucle `for` permite la **iteración**, ejecutando un bloque de código para cada subelemento dentro de un elemento iterable.

Ejemplos Básicos de `for`:

Iterable	Código	Resultado (Output)
Cadena de texto	`for letra in "Ana": print(letra.upper())`	A, N, A
Lista	`for numero in [1, 2, 3]: print(numero * 20)`	20, 40, 60
Tupla	`tupla_nombres = ("Paco", "Ana")\nfor nombre in tupla_nombres: print(len(nombre))`	4, 3
Función `range()`	`for i in range(2, 10, 2): print(i)`	2, 4, 6, 8

Iteración en Diccionarios:

Por defecto, iterar sobre un diccionario recorre sus **claves**. Para obtener los valores o ambos, se usan métodos especiales7.

Objetivo	Código
Solo claves	`for clave in diccionario_menu: print(clave)`
Claves y Valores	
Solo Valores	`for valor in diccionario_menu.values(): print(valor)`

Ejemplo con Condicionales (Análisis de Datos):

Python

```
productos = {"mesa": 40, "sillón": 20, "silla": 60, "cama": 200}

for articulo in productos:
    # Chequea si el valor de la clave (el precio) es >= 30
    if productos[articulo] >= 30:
        print(f"El articulo {articulo} cuesta {productos[articulo]}")
# Output: El articulo mesa cuesta 40, El articulo silla cuesta 60, El
articulo cama cuesta 200
```

---

2. List Comprehensions

Permiten la **creación de listas de forma rápida y elegante** 9 a partir de cualquier iterable, utilizando una sintaxis más compacta y eficiente que un bucle `for` tradicional.

Sintaxis y Ejemplos:

Tipo de `List Comprehension`	Sintaxis Concisa	Equivalente Clásico (`for`)
Básico (Solo `for`)	`[expresion for elemento in iterable]`	`for... append(expresion)`
Con Filtro (`if`)	`[expresion for elemento in iterable if condicion]`	`for... if condicion: append(expresion)`

Ejemplos Prácticos:

Objetivo	Bucle `for` Clásico	`List Comprehension`
Multiplicar por 10	`numeros = []\nfor i in range(1, 4): numeros.append(i * 10)`	`numeros = [i * 10 for i in range(1, 4)]`
Filtrar pares	`pares = []\nfor i in range(1, 11): if i % 2 == 0: pares.append(i)`	`pares = [i for i in range(1, 11) if i % 2 == 0] "]`
`if` ... `else` (Transformación)	`for... if: append("A") else: append("B")`	`["A" if condicion else "B" for elemento in iterable]`

---

3. Control de Errores: `try-except`

Esta estructura es clave para la robustez del código, ya que **gestiona las excepciones** (errores) para que el programa pueda continuar su ejecución a pesar de un fallo en una porción de código.

Ejemplos de `try-except` :

Tipo de Error	Código con `try-except`	Explicación
Conversión de Tipo	`try: print(int("hola")) except: print("¡Error!")`	Intenta convertir `"hola"` a entero. Al fallar, salta al `except` e imprime "¡Error!" en lugar de cortar el programa.
Manejo de Lógica en Bucle	`mascotas = ["perro", 20]\nfor m in mascotas: try: print(m.upper()) except: print("Elemento extraño")`	Recorre la lista. Para `"perro"` funciona. Para `20` da `AttributeError` , salta al `except` ,

Tipo de Error	Código con `try-except`	Explicación
		y el bucle **continúa** con los elementos restantes.
Manejo Específico	`entradas = ["10", "hola"]\nfor e in entradas: try: int(e) except ValueError: print(f"{e} no es un número válido.")`	Usa `except ValueError` para manejar específicamente el error de conversión de tipo, permitiendo que otros posibles errores no relacionados pasen sin ser capturados por este bloque.

Bucle `for`: Iteración y Automatización

El bucle `for` es una herramienta esencial para la **iteración masiva**, permitiendo repetir una porción de código tantas veces como elementos contenga una secuencia. Es decir, toma un **elemento iterable** (lista, tupla, cadena, rango de números, etc.) y ejecuta una acción por cada subelemento que contiene.

Tipo de Iterable	Ejemplo	Explicación
Lista	`for alimento in lista_compra: print(alimento)`	Recorre cada *string* ("lechuga", "tomate", etc.) dentro de la listafor comida in lista_compra: if comida == "tomate": continue print(f"Tienes que comprar {comida}")"].
Cadena de texto	`for letra in "ADALAB": print(letra)`	Recorre cada **carácter** (`A`, `D`, `A`, `L`, `A`, `B`) de la cadena.
Función `range()`	`for i in range(1, 11): print(i)`	Genera una secuencia de números del 1 al 10 (el número final es exclusivo).

Iteración en Diccionarios

Cuando se itera sobre un diccionario con un `for` básico, por defecto se recorren las **claves**.

- **Para acceder a claves y valores simultáneamente** se utiliza el método `.items()`: Python

```
diccionario_menu = {"plato": "sopa", "postre": "tiramisú"}
for clave, valor in diccionario_menu.items():
    print(f"De {clave} tenemos {valor}")
# Output: De plato tenemos sopa, De postre tenemos tiramisú
```

- **Para aplicar lógica** (e.g., filtrar por precio): Python

```
productos = {"zapatos": 20, "camisa": 32}
for articulo in productos:
    if productos[articulo] >= 30: # Accedemos al valor (precio) usando la
clave
        print(f"El articulo {articulo} cuesta {productos[articulo]}")
# Output: El articulo camisa cuesta 32
```

Control de Flujo de Bucle: `break`, `continue`, y `pass`

Estas palabras clave modifican el flujo normal de un bucle, permitiéndote tomar decisiones sobre cuándo detenerlo o saltar iteraciones.

Comando	Acción	Ejemplo Práctico
`break`	**Detiene el bucle por completo** y salta a la primera línea después de él.	Se utiliza al **buscar un elemento** y detenerse una vez encontrado para ahorrar procesamiento.
`continue`	**Omite el resto del código** en la iteración actual y pasa a la siguiente.	Útil para **limpieza de datos**; ignora valores defectuosos o nulos sin detener el análisis completo.
`pass`	No hace absolutamente **nada**. Es un marcador de posición sintáctico.	Se utiliza durante el desarrollo para estructuras que se completarán más tarde.

Ejemplo de `continue`:

Python

```
edades = [10, 30, 11, 5, 17, 43]
mayores_edad = []
for edad in edades:
    if edad > 18:
        continue # Ignora 30 y 43 y va a la siguiente edad
    mayores_edad.append(edad)
print(mayores_edad)
# Output: [10, 11, 5, 17]
```

Ejemplo de `break` con Catálogo de Películas (Búsqueda):

Python

```
catalogo = [{"nombre": "Avatar", "genero": "accion"}, {"nombre": "Barbie",
"genero": "familiar"}]
pelicula_alquilar = input("Dime qué película quieres alquilar") # Supongamos
que el usuario escribe 'Barbie'
for pelicula in catalogo:
    if pelicula["nombre"].lower() == pelicula_alquilar.lower():
        print(f"Toma la película {pelicula['nombre']}")
        break # ¡Se encontró! No es necesario seguir revisando el catálogo
else: # Este 'else' se ejecuta SÓLO si el bucle termina sin un 'break'
    print("No tenemos esa peli")
# Output: Toma la película Barbie
```

---

3. List Comprehensions

La forma más **elegante y eficiente** de crear listas nuevas. Su sintaxis concisa reemplaza las 3-4 líneas de código de un bucle `for` tradicional por una sola línea.

Transformación	**Sintaxis `List Comprehension`**
Mapeo (Aplicar Expresión)	`[palabra.replace("a", "e") for palabra in ["casa", "asa"]]`
Filtro (`if`)	`[num for num in range(1, 21) if num % 2 == 0]`
Doble `for` (Aplanar Lista)	`[num for sublista in listas for num in sublista]`

Ejemplo `if` (Filtro):

Python

```
# Bucle for clásico:
lista_nueva = []
for num in range(1, 21):
    if num % 2 == 0:
        lista_nueva.append(num)

# List Comprehension:
pares = [num for num in range(1, 21) if num % 2 == 0]
# pares = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```for num in range(1,21): if num % 2 == 0:
lista_nueva.append(num)print(lista_nueva)", "pares = [i for i in range(1,
11) if i % 2 == 0]"]
```
```

***

4. Control de Errores: `try-except`

El bloque `try-except` es un pilar de la **programación defensiva**, usado para manejar errores sin que el programa se detenga (corte).

* El código propenso a errores va en el bloque **`try`**.
* El código de manejo del error va en el bloque **`except`**.

Ejemplo de Prevención de Fallo en Bucle:
```python
mascotas = ["perro", "gato", 20]

for mascota in mascotas:
    try:
        # Intenta hacer .upper() (falla para 20)
        print(f"¡QUÉ PRECIOSIDAD DE {mascota.upper()}!")
    except:
        # Si falla (para 20, ya que no tiene método .upper), ejecuta esto y continúa el bucle
        print("Elemento extraño (Error de tipo ignorado)")
# Output: ¡QUÉ PRECIOSIDAD DE PERRO!, ¡QUÉ PRECIOSIDAD DE GATO!, Elemento extraño (Error de tipo ignorado)
```
```for mascota in mascotas: print(f"¡QUÉ PRECIOSIDAD DE {mascota.upper()}!")", "for mascota in mascotas: try: print(f"¡QUÉ PRECIOSIDAD DE {mascota.upper()}!") except: print("Elemento extraño")"]

**Ejemplo de Validación de Entrada (Manejo de Error Específico):**
```python
try:
    numero = int(input("Escribe un número: "))
    print(f"El doble de {numero} es {numero * 2}")
except ValueError: # Solo maneja errores cuando el valor no es convertible a int
    print("Tienes que introducir un NÚMERO válido.")
except IndexError: # Si el error es de índice
    print("Error de índice.")
# El programa no se detiene si el usuario escribe "abc"
```
```