Tema 4. Control de flujo

October 15, 2020

1 Introducción

- Con las instrucciones vistas hasta ahora solamente podíamos hacer programas secuenciales. Las instrucciones de control de flujo permiten romper el flujo secuencial de un programa.
- Hay tres familias:
  - Condicionales: `if`, `if-else`, `if-elif-else`
  - Bucles: `while`, `for`
  - Ramificación (branching): `break`, `continue` y `return` (tema 6)

2 Condicionales

- Ejecutan un bloque de sentencias si se cumple una condición y opcionalmente otro bloque si la condición no se cumple.

2.1 if

```
if condición:
    instrucción1
    instrucción2
     ...
    instrucciónN
```

- Condición: debe ser un booleano o una expresión booleana (algo que sea verdadero o falso)
- Las instrucciones se ejecutan solo si la condición es verdadera, si no lo es se ignoran. Para ello deben estar indentadas (desplazadas unos espacios a la derecha) respecto al if. La convención es poner 4 espacios, la tecla tabulador en casi todos los IDE coloca los 4 espacios automáticamente.
- A las instrucciones que se ejecutan si la condición es verdadera se les llama cuerpo o bloque del condicional (bloque del if)

[2]:
```
nota = float(input("Introduce tu nota "))
if nota >= 5:
    # Todo lo que está desplazado hacia la derecha se ejecutará solamente
    # si la condición es verdadera
    print("¡Aprobaste!")
    print("¡Enhorabuena!")
    print("En cuanto acabe el confinamiento hacemos una fiesta")
# Esto se ejecuta en cualquier caso porque no pertenece al bloque del if
```

```
print("Fin del programa")
```

```
Introduce tu nota 4.5
Fin del programa
```

2.2 if-else

```
if condición:
     instrucción1
     ...
     instrucciónN
else:
     instrucciónA
     ...
     instrucciónK
```

- Permite ejecutar un bloque si la condición es verdadera y otro si la condición es falsa
- La parte `else` es opcional, si no hay nada que ejecutar si la condición es falsa, se omite

[4]:
```
nota = float(input("Introduce tu nota "))
if nota >= 5:
    # Esto se ejecuta si la condición es verdadera
    print("¡Aprobaste!")
    print("¡Enhorabuena!")
    print("En cuanto acabe el confinamiento hacemos una fiesta")
else:
    # Esto si es falsa
    print("Vaya, mala suerte")
    print("Te toca estudiar para Junio")
# Esto en cualquier caso porque está fuera del if-else
print("Fin del programa")
```

```
Introduce tu nota 4.9
Vaya, mala suerte
Te toca estudiar para Junio
Fin del programa
```

2.3 Comprobar varias condiciones

- Utilizo operadores lógicos: `and`, `or`, `not`
- Si la comparación es muy grande y ocupa más de 79 caracteres, la puedo partir en varias líneas si la pongo entre paréntesis

[ ]:
```
PRECIO = 1.5
dinero = float(input("¿Cuánto dinero tienes? "))
# Todo lo que no sea una cadena vacía es verdadero
cerveza = bool(input("¿Quieres una cerveza? (Presiona Enter si no quieres)"))
# Por convención: cuando tengo un booleano no pongo == True sino que pongo
# solo su nombre. Si quisiera poner cerveza == False, pondría not cerveza
```

```
# Pongo primero lo más fácil de calcular (la comparación lleva algo más de␣
 ↪trabajo)
if cerveza and dinero >= PRECIO:
    print('Genial! Cerveza para ti')
    print('Está fresquita')
else:
    print('Vaya o eres pobre o no quieres cerveza')
    print('Malo en cualquier caso')
```

2.4 Encadenamiento de condicionales (if-elif-else)

- Se usa cuando quiero comparar varias cosas seguidas y solamente cuando todas las anteriores son falsas: comparo con una cosa, si es falso, lo comparo con otra, si esa también es falsa lo comparo con una tercera y así sucesivamente.
- El else es opcional igual que en el caso del if, se ejecuta si ninguno de los anteriores es verdadero
- `elif` es la contracción de `else if`
- En cuanto un `elif` es verdadero IGNORA todos los posteriores (podría haber varios verdaderos, aunque esos casos suelen ser por un fallo del programador)

[6]:
```
# Programa que usa elif para clasificar notas
nota = float(input("Introduce tu nota "))
if nota < 0 or nota > 10:
    print("¡Esa nota no existe!")
elif nota < 5:
    print("Suspenso")
elif nota < 7:
    print("Aprobado")
elif nota < 9:
    print("Notable")
elif nota < 10:
    print("Sobresaliente")
    # Este else se ejecuta si ninguno de los anteriores es verdadero
    # Otra alternativa sería poner elif nota == 10 (es un poco menos eficiente
    # porque requiere una comparación adicional)
else:
    print("Matrícula de Honor")
# Esto se ejecuta siempre porque está fuera del condicional
print("Este programa te ha dicho tu nota")
print("Gracias por usarlo")
```

```
Introduce tu nota 12
¡Esa nota no existe!
Este programa te ha dicho tu nota
Gracias por usarlo
```

[7]:
```python
# Programa que usa if para clasificar notas (mal)
# Cuando tengo varios if seguidos comprueba todos aunque alguno anterior
# haya sido verdadero. A veces me interesa ese comportamiento, pero en
# este caso no
nota = float(input("Introduce tu nota "))
if nota < 0 or nota > 10:
    print("¡Esa nota no existe!")
if nota < 5:
    print("Suspenso")
if nota < 7:
    print("Aprobado")
if nota < 9:
    print("Notable")
if nota < 10:
    print("Sobresaliente")
    # Este else solamente se ejecuta si el último if no es verdadero
    # No está relacionado con ningún if que no sea el último
else:
    print("Matrícula de Honor")
# Esto se ejecuta siempre porque está fuera de cualquier condicional
print("Este programa te ha dicho tu nota")
print("Gracias por usarlo")
```

```
Introduce tu nota 6
Aprobado
Notable
Sobresaliente
Este programa te ha dicho tu nota
Gracias por usarlo
```

[19]:
```python
# Python proporciona la instrucción elif para hacer más
# legible el código, pero lo que se hace con ella se puede
# hacer solo con if-else. Esto es un ejemplo del programa
# anterior hecho con if-else. Es más difícil de leer, aunque
# se ve más claramente que una vez un if se cumple, el resto se ignoran
nota = float(input("Introduce tu nota "))
if nota < 0 or nota > 10:
    print("¡Esa nota no existe!")
else:
    if nota < 5:
        print("Suspenso")
    else:
        if nota < 7:
            print("Aprobado")
        else:
            if nota < 9:
                print("Notable")
```

```
            else:
                if nota < 10:
                    print("Sobresaliente")
                else:
                    print("Matrícula de Honor")
```

```
Introduce tu nota 9.8
Sobresaliente
```

2.5 Definición de variables dentro de condicionales

- ¿Qué ocurre con las variables que declaro dentro del bloque de un `if/elif/else`?
- Tengo que asegurarme de que tienen un valor sea cual sea el valor de la condición si quiero usarlas después del condicional
- Se puede hacer de dos formas:
    - Dándole un valor inicial cualquiera antes del bloque condicional (recomendado y la única forma válida en otros lenguajes de programación)
    - Asegurándome de que en todas las ramas del condicional le doy un valor

[9]:
```
nota = float(input("Introduce tu nota "))
# Forma recomendada, declarar la variable antes con un valor cualquiera
aprobado = False
if nota >= 5:
    aprobado = True
# Segunda forma, asegurarme de que tanto el if como el else le dan un valor
#else:
#    aprobado = False
if aprobado:
    print("Enhorabuena")
else:
    print("Lo siento")
    print("Nos vemos en Junio")
```

```
Introduce tu nota 4
Lo siento
Nos vemos en Junio
```

3 Bucles

- Los bucles valen para repetir instrucciones
- Hay dos tipos de bucles:
    - Bucles que repiten mientras una condición es verdadera (no sabemos cuántas veces se van a repetir): `while`
    - Bucles que se repiten un número determinado de veces: `for`
- Ambos son equivalentes, podríamos tener solamente uno de ellos

3.1 Bucle while

- Bucle que repite mientras una condición sea verdadera

```
while condición:
    sentencia1
    sentencia2
    sentencia3
```

- Primero comprueba la condición: si es verdadera ejecuta `sentencia1`, etc. y vuelve a comprobar la condición. Si es falsa sigue con la siguiente línea después del bloque del `while`
- Un bucle while podría no ejecutarse nunca (si la condición es falsa la primera vez que pasa por él) o puede ejecutarse infinitas veces si la condición nunca se hace falsa
- Un bucle infinito es un error muy típico de principiante: hay que asegurarse de que en algún momento la condición se hace falsa

[17]:
```
import random
numero = random.randrange(1,11)
encontrado = False
# El bucle solo se ejecutará si la condición es verdadera
# not encontrado es equivalente a encontrado == False
while not encontrado:
    intento = int(input("Introduce un nombre entre 1 y 10 "))
    if numero == intento:
        print("¡Lo encontraste! Era el", numero)
        encontrado = True
    # Este else es opcional pero lo ponemos para que la salida del
    # programa sea más completa
    else:
        print("No, no es el", intento ,"sigue probando")
print("Gracias por jugar a este programa")
```

```
Introduce un nombre entre 1 y 10 1
No, no es el 1 sigue probando
Introduce un nombre entre 1 y 10 2
No, no es el 2 sigue probando
Introduce un nombre entre 1 y 10 3
No, no es el 3 sigue probando
Introduce un nombre entre 1 y 10 4
No, no es el 4 sigue probando
Introduce un nombre entre 1 y 10 5
¡Lo encontraste! Era el 5
Gracias por jugar a este programa
```

3.2 Componentes de un bucle

- Variable de control (o variables) es la que está en la condición y la que controla si se va a repetir el bucle o no
- Esta variable se debe inicializar antes del bucle

- Su valor debe cambiar en algún momento dentro del bucle (si su valor no cambia, bucle infinito)
- Me tengo que asegurar de que en algún momento toma un valor que haga la condición del bucle falsa (si no, bucle infinito)

[3]:
```
# Inicializo la variable de control
contador = 0
# Es la variable de control porque es la que se usa en la condición
while contador < 10:
    print(contador, end = ", ")
    # Cambio su valor dentro del bucle, en algún momento hará
    # que la condición sea falsa
    contador = contador + 1
```

```
0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
```

3.3 Bucle for

- Se utiliza cuando sabemos las veces que queremos repetir algo
- Dos tipos:
    - Bucle for 'clásico' (el que tienen la mayor parte de los lenguajes): decimos cuantas veces queremos que se repita
    - Bucle for-each (para cada uno): coge un conjunto o colección de cosas y repite lo mismo para cada uno de los elementos del conjunto. El único que tiene Python (el otro se puede simular fácilmente)

3.3.1 Bucle for-each

```
for elemento in colección:
    instrucción1
    ...
    instrucciónN
```

- `elemento` es un nombre de variable (puede ser cualquier nombre)
- `colección` debe ser un tipo de dato que contenga varios datos dentro. La única colección que hemos visto hasta ahora es string

[29]:
```
mi_letra = input("Introduce una letra ")
cadena = "Esto es una cadena de texto: es una colección de letras"
contador = 0
# letra es un nombre indicativo de lo que va a guardar
# no hace falta declarar letra antes, se inicializa en el bucle
for letra in cadena:
    # en letra se van copiando cada uno de los caracteres de la
    # cadena. Es como si en la primera iteración hiciéramos
    # letra = cadena[0], en la segunda letra = cadena[1] y así
    if letra == mi_letra:
        contador += 1
# Esto ya está fuera del bucle (no está indentado dentro)
```

```
print("La letra", mi_letra, "aparece", contador, "veces")
```

```
Introduce una letra a
La letra a aparece 5 veces
```

- ¿Cómo funciona un bucle for-each?
  - Coge cada elemento de la colección y lo copia en la variable
  - Ejecuta las instrucciones con el valor de la variable (no con la colección original)

[30]:
```
# Programa absurdo que no vale para nada, muestra que trabajamos
# con la copia, no con el original
for letra in cadena:
    # En letra COPIO cada elemento de cadena
    if letra == 'a':
        # Si cambio la variable letra, el elemento original no varía
        # porque las cadenas de texto son inmutables
        letra = 'x'
    # Vemos que la variable letra sí cambia de valor
    print(letra, end = "")
print()
# Pero la cadena no lo hace
print(cadena)

# Para cambiar caracteres de una cadena utilizo el método
# replace(letra_original, nueva_letra, numero_de_cambios)
# que devuelve una cadena con las letras cambiadas
print(cadena.replace('a',"x"))
# Pero tampoco cambia la cadena original
print(cadena)
# Para cambiar la original, tengo que guardar el resultado de
# cambiar las letras en la cadena original (realmente no estoy)
# cambiando la cadena original, sino creando una nueva y borrando
# la anterior
cadena = cadena.replace('a', "x", 2)
print(cadena)
```

```
Esto es unx cxdenx de texto: es unx colección de letrxs
Esto es una cadena de texto: es una colección de letras
Esto es unx cxdenx de texto: es unx colección de letrxs
Esto es una cadena de texto: es una colección de letras
Esto es unx cxdena de texto: es una colección de letras
```

3.3.2 Bucles for clásicos: función range

- Para hacer un bucle for clásico en Python (el que repite un número de veces) utilizo la función `range()`
- `range(número)`: devuelve todos los valores entre 0 y ese número (sin incluirlo)

- `range(a, b)`: devuelve todos los valores entre a y b (b no incluido)
- `range(a, b, c)`: devuelve todos los valores entre a y b (b no incluido) en pasos de c
- Se considera muy mala práctica de programación (y puede dar muchos problemas) cambiar el valor de la variable que hace de contador dentro del bucle

[22]:
```
# n toma sucesivamente los valores 0, 1, ... 9
# nunca debería cambiar el valor de n dentro del bucle
for n in range(10):
    print(n, end = " ")
print()
# n toma sucesivamente 2, 3, 4, ... 14
for n in range(2,15):
    print(n, end = " ")
print()
# n va de 2 a 26 (no incluido) contando de 3 en 3
for n in range(2, 26, 3):
     print(n, end = " ")
# Esto está fuera del bucle for, pero n sigue existiendo y tiene
# como valor el último que tomó en el último bucle
print()
print("El valor final de n es", n)
```

```
0 1 2 3 4 5 6 7 8 9
2 3 4 5 6 7 8 9 10 11 12 13 14
2 5 8 11 14 17 20 23
El valor final de n es 23
```

3.4 Bucles anidados

- Un bucle dentro de otro bucle: generalmente son bucles `for` dentro de otros bucles `for`, pero podríamos tener `for` dentro de `while` y viceversa
- Por cada iteración del bucle exterior, el interior se ejecuta todas las veces. Por lo tanto si el bucle exterior se ejecuta n veces y el interior m veces, en total el interior se ejecutará n * m veces

[31]:
```
# Usaremos estas variables para contar cuántas veces se ejecuta
# el bloque de cada bucle
exterior, interior = 0, 0
# Este es el bucle exterior, se repetirá 4 veces
# ext valdrá 0, 1, 2 y 3
for ext in range(4):
    exterior += 1
    # Este es el bucle interior, cada vez que el programa llegue
    # aquí se repetirá 5 veces
    for inter in range(5):
        interior += 1
print("El bucle exterior se repite:", exterior, "veces")
```

```
print("El bucle interior se repite:", interior, "veces")

# cuando ext vale 0: inter va tomando los valores 0, 1, ..4
# cuando ext vale 1: inter va tomando los valores 0, 1, ..4
# el interior se ejecuta ext * inter veces
```

```
El bucle exterior se repite: 4 veces
El bucle interior se repite: 20 veces
```

4 Instrucciones de ramificación

- `break`, `continue` y `return` (esta la vemos en tema 6)

4.1 break

- Si aparece `break` en un bucle se para el bucle y termina, ignorando si le quedaba algo de esa iteración por ejecutar (solo termina el bucle en el que está, no un bucle exterior si lo hubiera)
- Suele usarse más en los `for` que en los `while`. Este curso está terminantemente prohibido usar `break`: porque yo uso un for para algo que sé cuantas veces se va a repetir y si uso `break` ya no sé cuántas veces se va a repetir, debería usar un while en ese caso. Usar `break` en un ejercicio del examen significa que el ejercicio está mal

[32]:
```
# Ejemplo de funcionamiento de break, cuando se ejecuta break
# se interrumpe el bucle (como print va detrás, ni siquiera se
# imprime el 4)
for n in range(10):
    if n == 4:
        break
    print(n, end= " ")

# Ejemplo de uno de los usos más comunes de break
# Busco algo en una colección y cuando lo encuentro no sigo con
# el resto de elementos de la colección
# En este caso vamos a buscar una letra en una frase y a devolver
# su posición
s = "¿hola, cómo te encuentras hoy?"
mi_letra = input("Introduce la letra que estás buscando ")
# Primera forma, usando un for y break (no se puede usar este año)
contador = 0
for letra in s:
    if letra == mi_letra:
        print(mi_letra, "está en la posición", contador)
        break
    contador = contador + 1

# Segunda forma, como no sabemos cuántas veces vamos a repetir
# porque depende de la posición en la que esté la letra, usamos
```

```
# un bucle while con dos variables de control (podría valer con
# una, pero sería algo menos claro)
encontrada = False
contador = 0
while (contador < len(s) and not encontrada):
    if s[contador] == mi_letra:
        print(mi_letra, "está en la posición", contador)
        encontrada = True
    else:
        contador = contador + 1
```

```
0 1 2 3 Introduce la letra que estás buscando t
t está en la posición 12
t está en la posición 12
```

4.2 continue

- Salta lo que le queda de esa iteración y pasa a la siguiente

[27]:
```
for n in range(10):
    if n == 4:
        continue
    print(n, end= " ")
```

```
0 1 2 3 5 6 7 8 9
```

5 Operador puntos suspensivos

- No existe en Python (en realidad existe para una cosa que se llama elipsis)
- Si veis que necesitais puntos suspensivos es que os hace falta un bucle