Programación

Tema 3. Ramificación e iteración

Javier González Villa

David Lázaro Urrutia

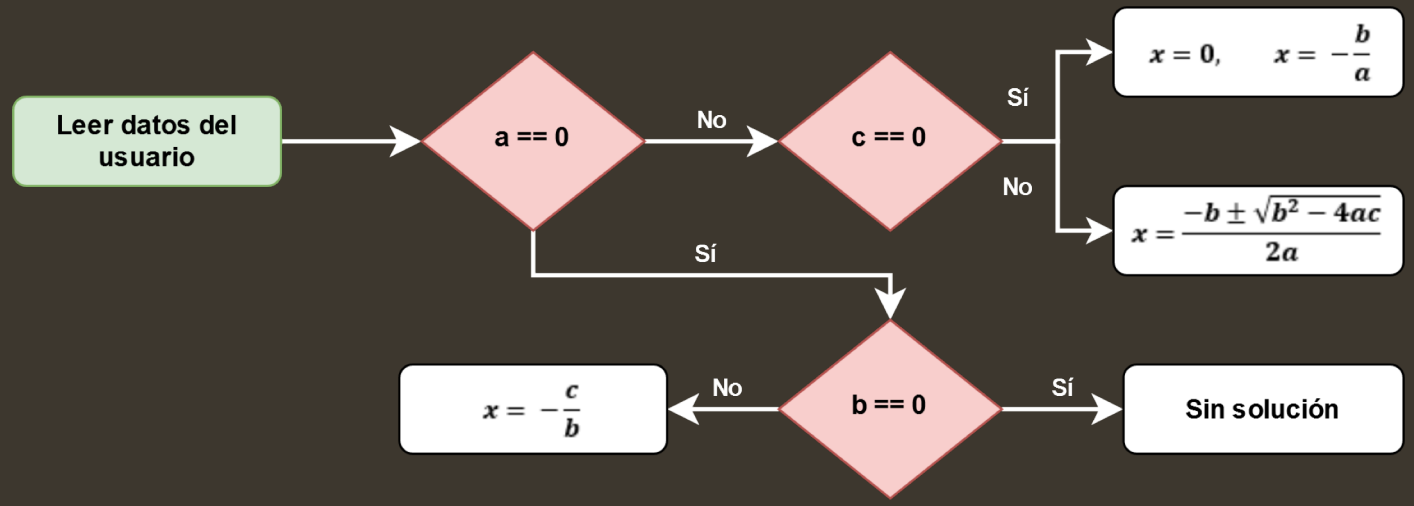
Departamento de Matemática Aplicada
y Ciencias de la Computación

Contenidos

1. Ramificación
    1. Operaciones de control de flujo
    2. Estructuras de control de flujo
2. Iteración
    1. Estructuras de control de flujo
3. Instrucciones adicionales de control

1. Ramificación

- Permite implementar decisiones en nuestros algoritmos, a través de estructuras de control de flujo condicionales.
- Se basa en tres grupos básicos de operaciones (comparación, lógica y sangrado) para establecer los caminos de código a ejecutar.



1.1. Operaciones de control de flujo

Comparación: *Conjunto de operaciones que permiten comparar el valor de dos variables retronando un valor booleano como respuesta (True/False).*

- Permite comparar el valor de variables de tipo entero (int), decimal (float) o cadenas (string).
- En el caso numérico, simplemente realiza la operación matemática de comparación.
- En el caso de las cadenas compara el orden alfabético de la sucesión de caracteres, a través de sus códigos ASCII.

Operador	Ejemplo	Resultado
>, >=	3 > 2	True
<, <=	3 < 2	False
==	‘hola’ == ‘hola’	True
!=	3.0 != 3	False

1.1. Operaciones de control de flujo

Lógica: *Conjunto de conectores lógicos y relaciones para la implementación de afirmaciones propias de la lógica proposicional.*

- Permite operar con variables de tipo booleano para obtener afirmaciones lógicas.
- Los operadores clásicos son los siguientes:
  - not B: niega el valor del booleano indicado (de True a False o de False a True).
  - B1 and B2: retorna True solo en el caso de que ambos valores booleanos sean True (si B1 y B2 entonces…).
  - B1 or B2: retorna False solo en el caso de que ambos valores booleanos sean False (si B1 o B2 entonces…).

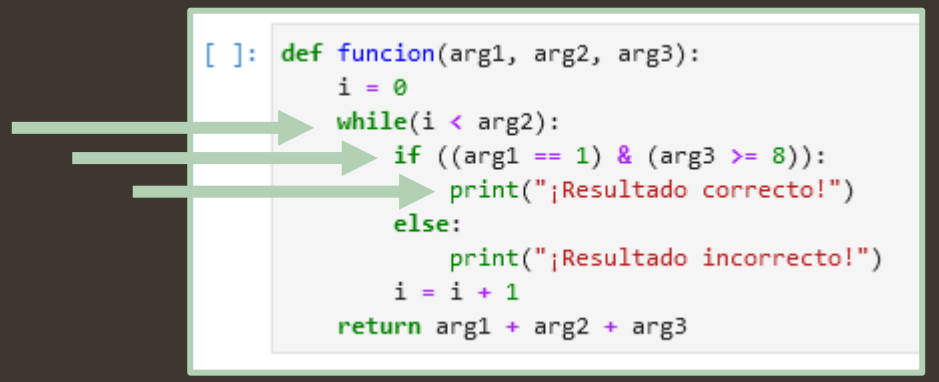
B1	B2	B1 and B2	B1 or B2
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

1.1. Operaciones de control de flujo

Sangrado: *Sangrar (mover ligeramente a la derecha) las líneas de código para indicar pertenencia a una clase, función, método u otro bloque de código.*

- Se puede utilizar con cualquier tipo de instrucción y solo consiste en tabular una vez hacia la derecha indicando pertenencia.
- Las instrucciones definirán un bloque de código y tendrán una jerarquía de ejecución donde siempre la instrucción con menor nivel de sangrado primará.

Niveles de Sangrado
Cada TAB define un nuevo nivel.

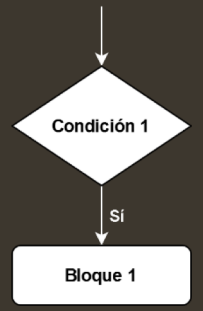


1.2. Estructuras de control de flujo

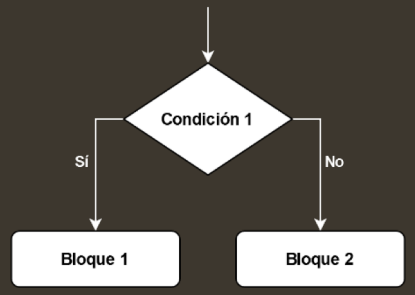
Condicional: *permite decidir por cuál alternativa seguirá el flujo del programa dependiendo del resultado de la evaluación de una condición.*

- Estructura:

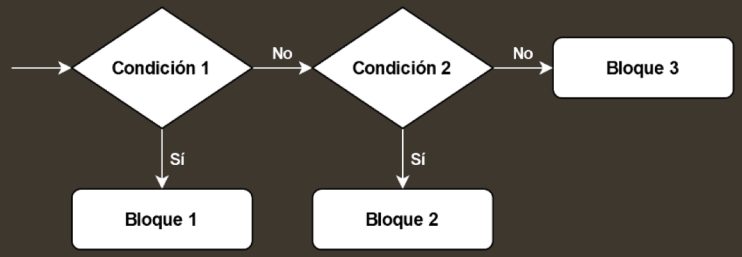
```
if condición 1:
    bloque 1
```




```
if condición 1:
    bloque 1
else:
    bloque 2
```




```
if condición 1:
    bloque 1
elif condición 2:
    bloque 2
else:
    bloque 3
```

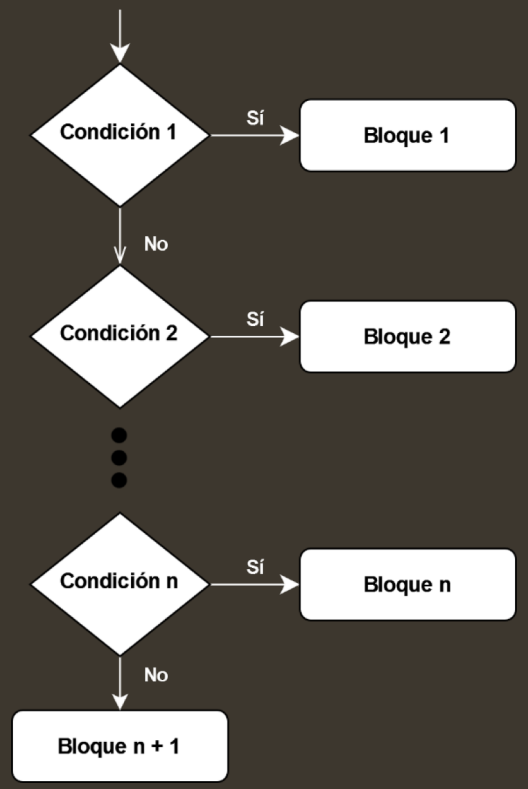


1.2. Estructuras de control de flujo

Condicional: *permite decidir por cuál alternativa seguirá el flujo del programa dependiendo del resultado de la evaluación de una condición.*

- Estructura:

```
if condición 1:
    bloque 1
elif condición 2:
    bloque 2
...
elif condición n:
    bloque n
else:
    bloque n + 1
```




```
[10]: condicion1 = False
      condicion2 = True
      if condicion1:
          print("Bloque 1")
      elif condicion2:
          print("Bloque 2")
      elif (condicion1 | condicion2):
          print("Bloque 3")
      else:
          print("Bloque 4")

      Bloque 2
```

```
[11]: condicion1 = False
      condicion2 = False
      if condicion1:
          print("Bloque 1")
      elif condicion2:
          print("Bloque 2")
      elif (condicion1 | condicion2):
          print("Bloque 3")
      else:
          print("Bloque 4")

      Bloque 4
```

1.2. Estructuras de control de flujo

Condicional: *permite decidir por cuál alternativa seguirá el flujo del programa dependiendo del resultado de la evaluación de una condición.*

- Estructura:

```
match variable:
    case condición 1:
        bloque 1
    ...
    case condición n:
        bloque n
    case _:
        bloque n + 1
```

```
[1]: calificacion = 7
     match calificacion:
         case 1 | 2 | 3 | 4:
             print("Suspenso")
         case 5 | 6:
             print("Aprobado")
         case 7 | 8:
             print("Notable")
         case 9 | 10:
             print("Sobresaliente")
         case _:
             print("No valida")

     Notable
```

```
[2]: calificacion = 7.3
     match calificacion:
         case 1 | 2 | 3 | 4:
             print("Suspenso")
         case 5 | 6:
             print("Aprobado")
         case 7 | 8:
             print("Notable")
         case 9 | 10:
             print("Sobresaliente")
         case _:
             print("No valida")

     No valida
```

1.2. Estructuras de control de flujo

Anidación: *inclusión de una estructura de control dentro de otra estableciendo un orden jerárquico.*

- Ejemplo:

```
if condición 1:
    bloque 1
    if condición 2:
        bloque 2
        …
        if condición n:
            bloque n
else:
    bloque n + 1
```

- Permite automatizar y optimizar procesos de decisión basados en condiciones.
- Ayuda a plasmar de forma más ordenada procesos de decisión complejos.
- Organiza el código de forma precisa plasmando niveles de evaluación de condiciones.
- Hay que mantener un balance entre anidación y secuencias condicionales y lógicas extensas para favorecer la legibilidad del código.

1.2. Estructuras de control de flujo

Buenas prácticas:

- Escribir condiciones claras y concisas, optimizando la lógica booleana y los operadores condicionales.
- Evitar anidamientos excesivos o redundancias.
- Emplear nombres descriptivos para las variables.
- Comentar las estructuras condicionales complejas.

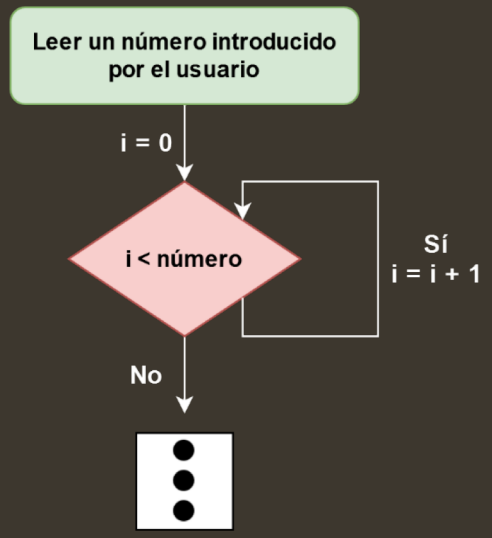
Ejemplos (errores):

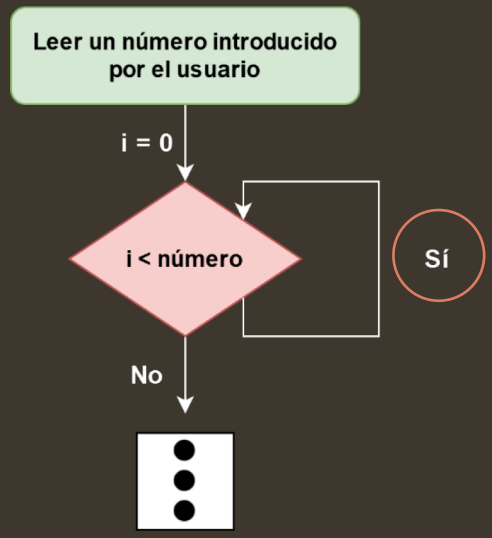
```
[9]: calificacion = 3
     if calificacion >= 5:
         aprobado = True
     elif calificacion < 5:
         aprobado = False
     print(aprobado)

     False

[10]: calificacion = 7
      aprobado = calificacion >= 5
      print(aprobado)

      True
```

```
[12]: var1 = 1.89
      var2 = 80
      if var2/var1**2 < 18.5:
          print("1")
      if (var2/var1**2 > 18.5) and (var2/var1**2 < 24.9):
          print("2")
      if (var2/var1**2 > 24.9):
          print("3")

      2

[13]: altura = 1.89 # m
      peso = 80 # Kg
      imc = peso / altura**2
      if imc < 18.5:
          print("Peso insuficiente")
      elif imc < 24.9:
          print("Peso normal")
      else:
          print("Sobrepeso")

      Peso normal
```

2. Iteración

- Permite implementar bucles en nuestros algoritmos, a través de estructuras de control de flujo iterativas y condicionales.
- Se basa en los mismos tres grupos básicos de operaciones (comparación, lógica y sangrado) para establecer los caminos de código a ejecutar o la permanencia en un bucle.





2.1. Estructuras de control de flujo

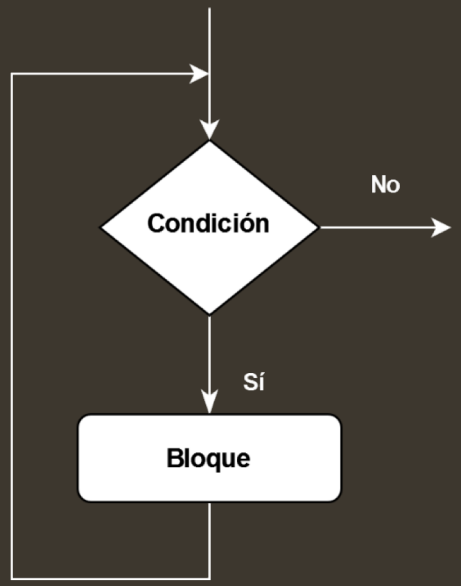
Bucle: *Conjunto de instrucciones cuya ejecución se repite hasta que una determinada condición de salida se vea satisfecha.*

- Estructura (while):

  while condición:

  bloque

- Se debe tener especial cuidado en que las variables utilizadas como condición tengan un criterio de parada ya que sino puede estar ejecutando infinitamente.




```
[12]: i = 0
      while i<3:
          print('Iteración')
          i = i + 1

      Iteración
      Iteración
      Iteración
```

2.1. Estructuras de control de flujo

Bucle: *Conjunto de instrucciones cuya ejecución se repite hasta que una determinada condición de salida se vea satisfecha.*

- Estructura (for):

  for variable **in** iterable:

  bloque

- Iterables: listas, tuplas, cadenas y diccionarios (solo claves).
- La variable que se itera puede utilizarse dentro del bloque de código y hay que tener en cuenta que cambia de valor en cada iteración.

```
[17]: for variable in [1,2,3]:
          print(variable)

      print(variable)
```

```
1
2
3
3
```

```
[18]: for variable in range(1,4):
          print(variable)
```

```
1
2
3
```

2.1. Estructuras de control de flujo

Anidación: *inclusión de una estructura de control dentro de otra estableciendo un orden jerárquico.*

- Ejemplo:

```
for fila in matriz:
    for elemento in fila:
        bloque

while condicion1:
    while condicion2:
        bloque
```

- Permite manejar estructuras de datos multidimensionales como matrices.
- Facilita la resolución de problemas con varios niveles de iteración como combinaciones, permutaciones o comparación cruzada.
- Mejora la legibilidad y optimiza el código.
- La anidación excesiva puede aumentar significativamente el coste computacional.

2.1. Estructuras de control de flujo

Bucle: *Conjunto de instrucciones cuya ejecución se repite hasta que una determinada condición de salida se vea satisfecha.*

for	while
Número de iteraciones predefinidas en el código.	El número de iteraciones no está limitado.
No puede implementar cualquier tipo de algoritmo.	Puede implementar cualquier tipo de algoritmo incluido los realizados con bucles for.
Se utiliza generalmente para recorrer elementos finitos.	Se utiliza para establecer criterios de parada condicionales más complejos.
Puede dar errores de utilización de datos iterados.	Puede dar errores de criterio de parada.

2.1. Estructuras de control de flujo

Bucle: *Conjunto de instrucciones cuya ejecución se repite hasta que una determinada condición de salida se vea satisfecha.*

- Repetición de tareas o cálculos recurrentes.
- Recorrido de estructuras de datos para búsquedas, ordenación, filtrado, etc.
- Interacción con el usuario.

```
[31]: for largo in range(3):
          ancho = 0.3
          alto = 0.5
          volumen = largo * ancho * alto
          print("Volumen de viga de",largo,"m:", volumen, "m³")
```

```
Volumen de viga de 0 m: 0.0 m³
Volumen de viga de 1 m: 0.15 m³
Volumen de viga de 2 m: 0.3 m³
```

```
[2]: while True:
         operacion = int(input("Indica operación (0 - Salir, 1- Continuar):"))
         if operacion == 0:
             break
```

```
Indica operación (0 - Salir, 1- Continuar): 1
Indica operación (0 - Salir, 1- Continuar): 0
```

```
[32]: cargas = [5, 12, 7, 18]
      for carga in cargas:
          if carga > 10:
              print("Carga mayor a 10 toneladas:", carga)
```

```
Carga mayor a 10 toneladas: 12
Carga mayor a 10 toneladas: 18
```

3. Instrucciones adicionales de control

Comandos de control de flujo:

- range(inicio, parada, paso): genera una lista iterable con los valores indicados.
- continue: salta a la siguiente iteración sin ejecutar el resto del bloque.
- break: para el bucle y no permite realizar el resto de iteraciones.

https://docs.python.org/es/3.13/tutorial/datastructures.html#looping-techniques

```
[23]: lista = range(10)
      print(len(lista))
      print(lista[0])
      print(lista[9])

      10
      0
      9

[26]: for variable in range(9):
          if(variable == 3):
              continue
          print(variable)
          if(variable == 5):
              break

      0
      1
      2
      4
      5
```