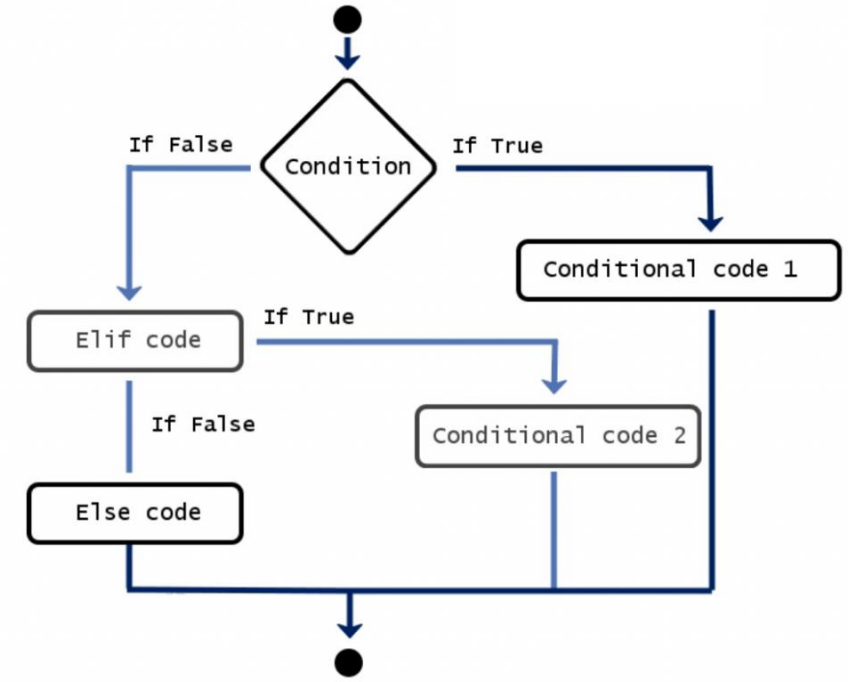
Jul. 2022

Unidad 2: Sentencias de control de flujo



Introducción

Continuamos adquiriendo conocimientos del lenguaje para poder empezar a incorporar estructuras más complejas en nuestros programas

- Cómo Python maneja las sentencias de control
- Cómo podemos cambiar el flujo de ejecución
- Cómo podemos ejecutar código repetitivamente

Objetivos de aprendizaje

Ser capaz de:

- Describir las diferentes sentencias de control
- Controlar el flujo de ejecución del programa
- Utilizar estructuras de bucle
- Implementar bifurcaciones dentro de estructuras de bucle

Flujo del programa

Describe la forma en que se ejecutan las sentencias en el código incluyendo la prioridad de los diferentes elementos

- De arriba hacia abajo
- Cada línea tiene que esperar
  hasta que todas las líneas que vienen antes de ella hayan completado su ejecución

Sentencia de control

Es una estructura en el código
que cambia condicionalmente el flujo del programa
ejecutando diferentes partes del código
también puede utilizarse para ejecutar repetidamente algún código

Analogía: Como un policía de tráfico en un cruce
que sólo deja pasar el tráfico si la salida está libre.

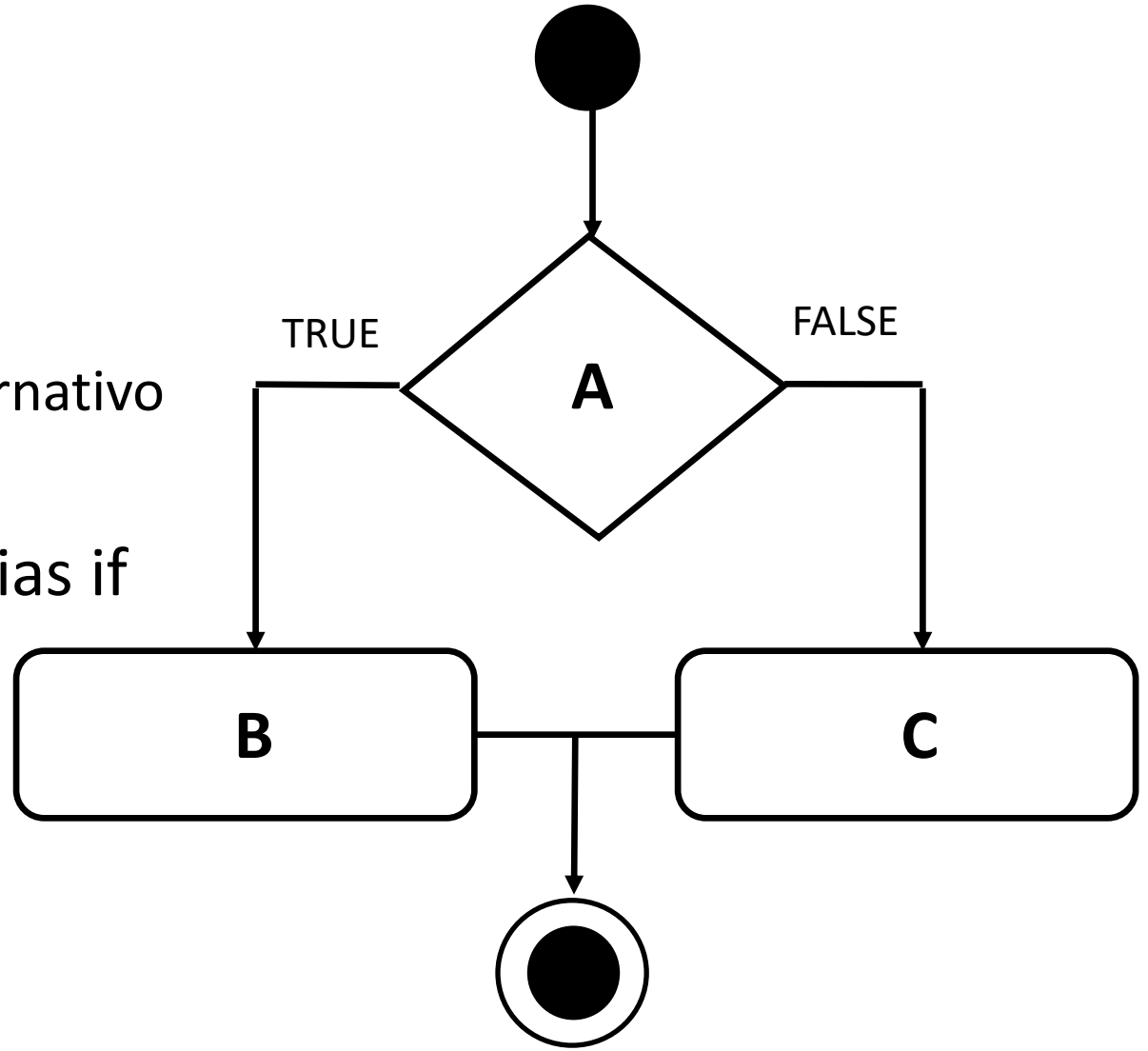
Las dos principales sentencias de control en Python son

- if
- while

La sentencia **if**

- Si una condición es verdadera
  - Ejecutar un bloque de código
  - caso contrario ejecuta un bloque alternativo
- La cláusula **else** es opcional
- Se puede encadenar varias sentencias if
- La sintaxis básica:

```
if condición:
    #Ejecuta es Verdadera (T)
else:
    # Ejecuta es Falsa (F)
```



E1: Uso de la sentencia if

1. Declarar una variable llamada año_de_lanzamiento y le asignamos el valor “1991”
2. Declarar otra variable llamada respuesta y asígnale la respuesta a la pregunta: “¿Cuándo se lanzó Python por primera vez?”
3. Utilizar if para comprobar si la respuesta introducida es correcta
4. Utilizar elif para comprobar si la respuesta es mayor o menor que la correcta y decirle al usuario que la respuesta es demasiado alta / baja”
5. Imprimir el mensaje de salida (¡Adiós!')

Actividad 1: Trabajando con la sentencia if

Modificamos el ejercicio anterior como sigue:

1. La pregunta al usuario debe ser `"Escribe TRUE o FALSE: ¿Python se lanzo en 1991?`
2. Si el usuario ingresa TRUE, imprime “Correcto”
3. Si el usuario ingresa FALSE, imprimir “Incorrecto”
4. Cualquier otra entrada del usuario debe imprimir el mensaje `'Por favor responda TRUE o FALSE'`

- La salida debería ser la siguiente:

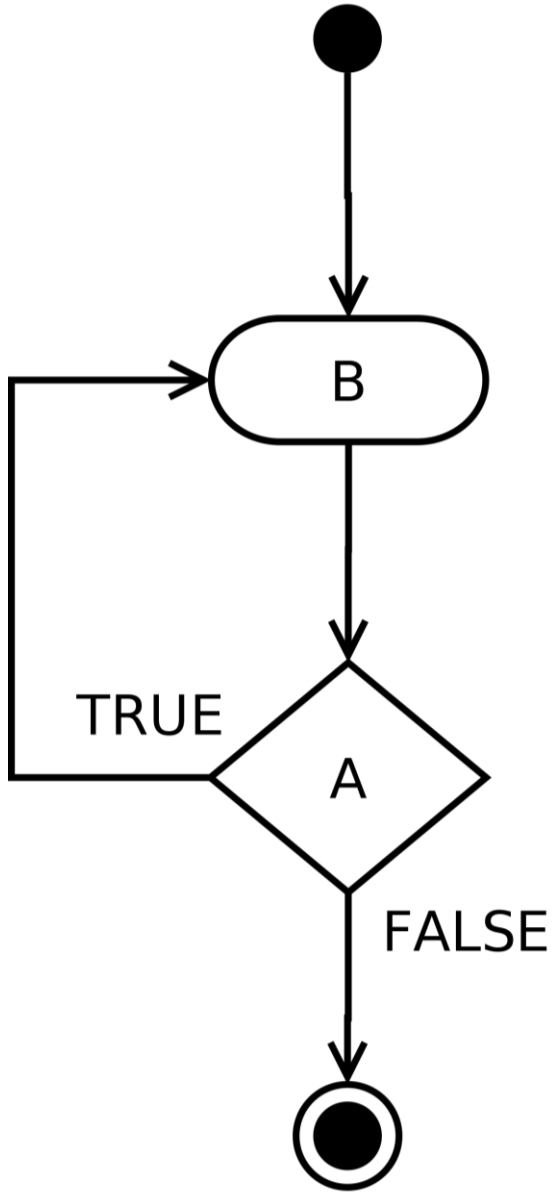
```
Escribe TRUE o FALSE: ¿Python se lanzo en 1991?: SI
Por favor responda TRUE o FALSE
Adios!

Escribe TRUE o FALSE: ¿Python se lanzo en 1991?: FALSE
Incorrecto
Adios!
```

La sentencia **while**

- Permite ejecutar un bloque de código repetidamente mientras una condición permanezca verdadera

```
while condición:
        # mientras la condición sea verdadera
else:
        # una vez que la condición ya no es verdadera
        # se ejecuta este código una sola vez
```

- Analogía: Un agente de tráfico dejando pasar el tráfico mientras la salida esté despejada y en cuanto ya no lo esta impediendo que los conductores pasen



E2: Uso de la sentencia while

1. Declarar una variable llamada contador con valor “1”
2. Declarar la variable final con un valor de 10
3. Escribir un while con la condición de que el contador sea menor o igual que el valor de la variable final
4. Para cada valor del contador, imprimirlo e incrementarlo en 1
5. Cuando la condición ya no es verdadera, imprimir

“Has llegado al final”

- La salida debería ser la siguiente:

```
1
2
3
4
5
6
7
8
9
10
Ha llegado al final
```

E3: Uso de while para mantener un programa en ejecución

Reescribir el programa que escribimos antes y añadiremos una sentencia while:

1. Declarar año_de_lanzamiento en 1991 (respuesta correcta)
2. Establecer la condición correcto en False.
3. Mientras la respuesta proporcionada no sea correcta mantener el programa en marcha
4. Imprimir la pregunta en la terminal y esperar la respuesta del usuario
5. Utiliza una sentencia **if** para comprobar que la respuesta proporcionada es correcta
    Si la respuesta es correcta
        Imprimir un mensaje de éxito
        Establecer el valor de correcto a True.
    Si la respuesta es incorrecta, animar al usuario a volver a intentarlo
6. Imprimir un mensaje de salida

Actividad 2: Trabajar con la sentencia while

Crear un programa de autenticación de contraseñas utilizando un bucle while

1. Definir una variable contraseña con el valor “random”
2. Definir un validador booleano con el valor False
3. Inicie el bucle while y pida la entrada del usuario
4. Valide la contraseña
    Si la contraseña es correcta
        Imprimir un mensaje de bienvenida al usuario
        Establecer el valor del validador a True
    Si la respuesta es incorrecta, animar al usuario a volver a intentarlo
5. Ejecuta el script

- La salida debería ser:

```
por favor introduzca su contraseña: ramdom
Contraseña incorrecta, intentelo nuevamente
por favor introduzca su contraseña: random
Bienvenido usuario!
```

while versus if

- Una sentencia if da la oportunidad de bifurcar la ejecución del código basándose en una condición
- El código en el bloque if se ejecuta una sola vez
- Una sentencia while puede ejecutar un bloque de código varias veces

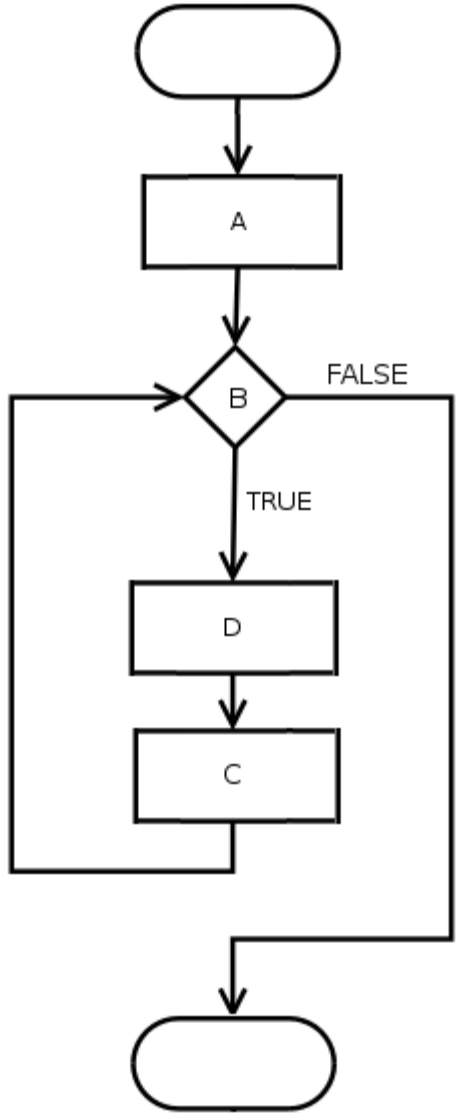
Bucles

- Son una forma de ejecutar un bloque de código varias veces
- Se utilizan para iterar sobre cualquier objeto
  - Cadenas, Listas, Diccionarios, Archivos
- Un iterable como una colección de cosas homogéneas que se han agrupado para formar un gran colectivo
- Los individuos del colectivo tienen las mismas propiedades,

  y cuando se combinan, forman algo nuevo

El bucle **for**

- Permite recorrer y ejecutar un bloque de código para cada elemento de un iterable
- for VS while:
  - **for** el bloque de código repetido se ejecuta un número predeterminado de veces
  - **while** el código se ejecuta un número arbitrario de veces siempre que se cumpla una condición



E4: Uso del bucle for

1. Declare una variable llamada números e inicializarla con una lista de enteros del 1 al 10
2. Recorre la lista que acabamos de crear
   Crear una variable llamada **num** en el bucle for
3. Dentro del bucle
   Calcula y asignar el cuadrado del numero
   Asignamos el calculo a una variable **cuadrado**
4. Imprimimos una cadena que nos dice que "**num** al cuadrado es"

- La salida será algo como:

```
1 el cuadrado es 1
2 el cuadrado es 4
3 el cuadrado es 9
4 el cuadrado es 16
5 el cuadrado es 25
6 el cuadrado es 36
7 el cuadrado es 49
8 el cuadrado es 64
9 el cuadrado es 81
10 el cuadrado es 100
```

La función Range

- Es una función incorporada que genera una lista de números
- Se utiliza principalmente para iterar sobre ella utilizando un bucle for sin perder de vista el índice.
- Sintaxis: `range([inicio], stop, [paso])`

  inicio: Es el número inicial de la secuencia (0 si no se especifica)

  stop: Generar números hasta este número pero sin incluirlo

  paso: Es la diferencia entre cada número de la secuencia
- Para ver los números convertir el objeto range a un objeto lista

Actividad 3: El bucle for y la función range

- Usando un bucle for y una función de rango,
  encontrar los números pares entre 2 y 100
  y luego encuentre su suma

- La salida será:

```
2550
```

1. Definir una variable para la suma.
2. Definir un bucle **for** con un rango par para los números entre 2 y 101.
3. Añadir cada número del bucle a la suma
4. Fuera del bucle, imprime la suma total

Anidamiento de bucles

- La práctica de colocar bucles dentro de otros bucles
- Cuando se necesita acceder a datos dentro de una estructura de datos compleja
- A veces es necesario usar uno o más bucles dentro de otro bucle para llegar a ese nivel de granularidad

E5: Anidamiento de bucles

- Utilizaremos bucles anidados para los cuadrados de los números:

1. Crear una variable llamada grupos, que es una lista que contiene otras tres listas de tres enteros cada una

```
[[1,2,3],[4,5,6],[7,8,9]]
```

2. Hacer un bucle sobre los grupos
3. Por cada grupo, hacemos un bucle sobre él

para llegar a los enteros a los que llamamos num

4. Calcular el cuadrado de cada número
5. Imprimir el cuadrado del número

- La salida será algo como:

```
1 su cuadrado es:  1
2 su cuadrado es:  4
3 su cuadrado es:  9
4 su cuadrado es:  16
5 su cuadrado es:  25
6 su cuadrado es:  36
7 su cuadrado es:  49
8 su cuadrado es:  64
9 su cuadrado es:  81
```

Actividad 4: Bucles anidados

- Suma los números pares e impares entre 1 y 11 e imprima el cálculo

1. Escribir un **for** con la función de rango para los números pares
2. Escribir otro **for** con la función de rango para los impares
3. Utilizar una variable **suma** para calcular la suma de pares e impares
4. Imprimir **suma**

- La salida será algo como

```
2 + 1 = 3
2 + 3 = 5
2 + 5 = 7
2 + 7 = 9
2 + 9 = 11
4 + 1 = 5
4 + 3 = 7
....
8 + 7 = 15
8 + 9 = 17
10 + 1 = 11
10 + 3 = 13
10 + 5 = 15
10 + 7 = 17
10 + 9 = 19
```

Resumen

- hemos aprendido
  - cómo fluyen los programas
  - a controlar y bifurcar el flujo de un programa (if y while).
  - algunas aplicaciones prácticas de las dos sentencias de control
  - cómo difieren en su implementación y sintaxis
  - las estructuras de bucle
  - Hemos visto la estructura de un bucle for y hemos visto ejemplos prácticos
  - la función range y su utilidad
  - cómo y cuándo anidar bucles