Lección 12: Bucles

Índice

Anterior

Definición

Los bucles se utilizan para gestionar tareas repetitivas o tareas en las que se iteran los elemenos de una lista, tupla, diccionario o cualquier otro elemento iterable. En python se pueden utilizar dos tipos de bucles:

- Bucles *While*: Mientras se cumpla una determinada condición, ejecuta un bloque de código.
- Bucles *For*: Para todos (o ciertos) elementos de un objeto iterables, ejecuta un bloque de operaciones.

Bucles *While*

- Mientras se cumpla una determinada condición, ejecuta un bloque de código.

El flujo de ejecución sería el siguiente: 1. Evalúa la condición 2. Si la condición es falsa, sale de bucle (no lo ejecuta más) 3. Si la condición es verdadera, ejecuta el bloque de código (incluyendo actualizar las variables de la condición) y vuelve al paso 1.

```
# sintaxis en pseudocódigo
while condicion:
    haz esto
    haz esto también
    haz esto también
    update condicion
```

Nota: Si la condición no se actualiza puede producirse un bucle infinito.

```
# Bucles While

a = 0
while a < 5:
    print(f'El valor de a es {a}')
    a = a + 1   # En los bucles While hay que actualizar la condición para no caer en un bu
print(f'Fin del bucle While\n\n')

a = 0
while a < 10:
    print(f'El valor de a es {a}')
    a+= 1   # Otra forma de actualizar la variable del bucle
```

Modificando el comportamiento de los bucles con *break* y *continue*

En ocasiones el programador no sabe anticipadamente cuántas veces tiene que repetirse la iteración. En ese caso, puede utilizarse de forma controlada un bucle infinito que será detenido cuando se cumpla otra condición utilizando *break*.

```
# pseudocódigo
while condicion:
    haz esto
    if otra_condicion:
        break
```

```
# Modificando el comportamiento de los bucles con break y continúe

while True:
    cmd = input(">")
    if cmd == "exit":
        break
    print(cmd)
```

A veces te encuentras en una iteración de un bucle y deseas finalizar la iteración actual y pasar de inmediato a la siguiente iteración. En ese caso, puedes usar la instrucción *continue* para saltar a la siguiente iteración sin terminar el cuerpo del bucle para la iteración actual.

```
 # pseudocódigo
while condicion:
    haz esto
    if otra_condicion:
        continue
    haz esto otro   # Esta parte no se ejecuta en las iteraciones en las que otra_condicion
```

```
# Modificando el comportamiento de los bucles con break y continúe

while True:
    cmd = input(">")
    if cmd.startswith("#"):
        continue
    if cmd == "exit":
        break
    print(cmd)
```

Bucles *For*

- Se utiliza para iterar sobre una secuencia de elementos de un iterable (lista, tupla, diccionario y otros). Conceptualmente es similar a otros lenguajes de programación, pero la sintaxis y forma de utilizarlo es ligeramente distinta.

```
for elemento in iterable:
    haz esto
    y esto
    y esto otro
    y se puede acceder al elemento o modificarlo (si el objeto es mutable)
```

```
# Bucles For

# Bucles for en una lista
amigos = ["Pedro", "Arturo", "María José", "José Luis", "Enrique"]

for amigo in amigos:
    print(f"Feliz Día de la Hipanidad, {amigo}")


# Bucles for con enumerate
for i,amigo in enumerate(amigos):
    print(f'{i}: {amigo}')


# Bucles for en una tupla
tpl = (1,4,5,7,8,10)

for n in tpl:
    print(f"{n}")

# Bucles for en una cadena de texto
cadena = "Python!"

for letra in cadena:
    print(letra)

# La forma menos pythonica de hacerlo:
for i in range(len(cadena)):
    print(cadena[i])
```

```
# Bucles for en diccionarios

persona = {
    'nombre':'Manuel',
    'apellido':'Ejemplar',
    'edad':26,
    'país':'España',
    'casado':True,
    'conocimientos':['JavaScript', 'React', 'Node', 'MongoDB', 'Python'],
    'direccion':{
        'calle':'De la protección de datos',
```

```
        'cp':'28000'
    }
    }

for clave in persona:
    print(clave)

for valor in persona.values():
    print(valor)

# De forma mas eficiente se puede acceder a claves y valores en un diccionario

for clave,valor in persona.items():
    print(f'La clave {clave} tiene el valor {valor}')
```

Break y *Continue* en bucles *For*

- *break* y *continue* se pueden utilizar también en bucles *for*. Se utilizan de forma similar a cómo se utilizan en bucles *while*

```
# pseudocódigo
for elemento in iterable:
    haz esto
    if condicion:
        break
```

```
# pseudocódigo
for elemento in iterable:
    haz esto
    if condicion:
        continue
    haz también esto si no se cumple la condición
```

La función *range* y bubles *For*

La función *range()* se utiliza para crear listas de números. La función *range(inicio, fin, incremento)* toma tres parámetros: inicio, fin e incremento. Por defecto empieza en 0 y el incremento es 1. La secuencia range necesita al menos 1 argumento (end).

La función *range* se puede utilizar para construir bucles *for* similares a los que se utilizan en otros lenguajes.

```
# La función range y bucles for

numeros = range(10)
print(numeros)
print(type(numeros))
```

```python
for i in range(10):
    print(i)
```

Bucles y condicionales anidadas

Al igual que los condicionales, los bucles *for* se pueden anidar.

```python
# Bucles anidados

for clave,valor in persona.items():
    print(f'{clave.capitalize()}:')
    if clave == "conocimientos":
        for c in valor:
            print(f'\t* {c}')
    elif clave == "direccion":
        print(f"\t Calle: {valor['calle']}, C.P.: {valor['cp']}")
    else:
        print(f'\t{valor}')
```

For Else

- Se utiliza para ejecutar códido después de la última interación. No se ejecuta tras un *break*.

```python
# For Else

for i in range(10):
    print(f'Iteración: {i}')
else:
    print(f'Ejecución finalizada en iteración: {i}')


for i in range(10):
    print(f'Iteración: {i}')
    if i == 5:
        break
else:
    print(f'Ejecución finalizada en iteración: {i}')
```

Pass

En python cuando se requiere una sentencia (después del punto y coma), pero no queremos ejecutar ningún código allí, podemos escribir la palabra *pass* para evitar errores de sintaxis. También podemos utilizarlo como un marcador de posición, para futuras declaraciones.

```python
for amigo in amigos:
    pass
```

Siguiente