Bottega DevCamp

Full Stack Developer

Checkpoint 5: Ejercicios teóricos

Fecha: 10 de marzo de 2025

Autor: Domínguez Becerril, Jorge DNI: 72607195J

Índice

1. Cuestiones teóricas

1.1. ¿Qué es un condicional?

Una estructura condicional o de tipo **if** es una estructura que permite condicionar la ejecución de un bloque de código al cumplimiento de una o varias condiciones impuestas por el desarrollador. En Python, esta estructura se define utilizando la palabra clave **if** seguida de una condición y dos puntos (:). El bloque de código que sigue debe estar indentado. En la figura 1 se puede ver un ejemplo del uso de esta estructura.

```
edad = 27

if (edad > 18):
    print('Tiene edad para sacarse el carné de conducir.')
```

```
Tiene edad para sacarse el carné de conducir.
```

Figura 1: Ejemplo de uso de la estructura **if**.

Este programa define la variable **edad** y, si esta variable tiene un valor mayor de 18, el programa imprime una cadena. Como puede ver, se ha asignado a la variable **edad** el valor 27, de manera que se imprime la cadena. Existe una variación de la estructura condicional **if**, la estructura **if - elif - else**, que permite ejecutar distintos bloques de código en función de la condición o condiciones que se cumplan. En la figura 2 se recoge un ejemplo de esta variación.

```
edad = 17

if (edad >= 16) and (edad < 18):
    print('Tiene edad para sacarse el carné de conducir de motos pero no de coches.')
elif (edad >= 18):
    print('Tiene edad para sacarse el carné de conducir de coches y motos.')
else:
    print('No puede conducir ningún vehículo a motor.')
```

```
Tiene edad para sacarse el carné de conducir de motos pero no de coches.
```

Figura 2: Ejemplo de uso de la estructura **if - elif - else**.

Este código define la variable **edad** y, en función de su valor se imprimen distintas cadenas. Si su valor no cumple ninguna de las condiciones se ejecuta el bloque de código contenido en la palabra clave **else**, que imprime la cadena **'No puede conducir ningún vehículo a motor.'**. Si se cumple la condición de que sea mayor o igual a 18 se imprime en pantalla la cadena **'Tiene edad para sacarse el carné de conducir de coches y motos.'** y si se cumplen las condiciones de que sea mayor o igual a 16 y menor a 18 se imprime la cadena **'Tiene edad para sacarse el carné de conducir de motos pero no de coches.'**. En el caso del ejemplo se ha asignado a la variable **edad** el valor 17 y el programa nos dice que solo existe la posibilidad de conducir motos. Cabe destacar que

es posible introducir más de una sentencia **elif** en la estructura condicional. Además, no es necesario utilizar las palabras **if**, **elif** y **else** de forma conjunta. Se puede utilizar la estructura **if - else** o la estructura **if - elif**.

1.2. ¿Cuáles son los diferentes tipos de bucles en Python? ¿Por qué son útiles?

Los bucles son estructuras que permiten iterar un bloque de código. En Python existen dos tipos de bucles:

- Bucle **for**: este tipo de bucle permite iterar un bloque de código un número determinado de veces antes de seguir con el flujo normal del programa.
- Bucle **while**: este tipo de bucle permite iterar un bloque de código mientras se cumpla una o varias condiciones.

En la figura 3 se muestra un ejemplo de bucle **for**. Este código define la lista **frutas** y, utilizando el bucle **for**, imprime en pantalla todos los elementos de la lista. Finalmente se imprime la cadena **'Fin del programa'**. Nótese que el bloque de código dentro del bucle **for** está **indentado**. Además, en este tipo de bucles siempre se conoce el número de veces que se va a iterar el bloque. En este caso, se imprimen todos los elementos de la lista **frutas**, de manera que el bloque de código dentro del bucle se ejecutará 5 veces, que corresponde a la longitud de la lista **len(frutas)**. Una vez que finaliza el bucle el programa continua con su flujo normal. En el ejemplo se imprime la cadena **'Fin del programa'**.

```
frutas = ['manzanas', 'platanos', 'peras', 'mandarinas', 'limones']
for fruta in frutas:
    print(fruta)

print('Fin del programa')
```

```
manzanas
platanos
peras
mandarinas
limones
Fin del programa
```

Figura 3: Ejemplo de bucle **for**.

Cuando no se conoce el número de veces que se quiere iterar un bloque de código entra en juego el bucle **while**. En la figura 4 se muestra un ejemplo de este tipo de bucle. Este código define la variable **frutas**, a la que se asigna una lista con nombres de distintas frutas y, utilizando un bucle **while**, se van eliminando una a una empezando por la última hasta que la longitud de la lista sea 2. Finalmente se imprime en pantalla la lista resultante en cada iteración. Al igual que en el bucle **for**, en este tipo de bucle también se debe indentar el bloque de código a iterar. En cada iteración se comprueba la condición. Si esta es verdad se ejecuta una nueva iteración. En caso contrario, se termina el bucle **while** y se prosigue con el flujo normal del programa.

```
frutas = ['manzanas', 'platanos', 'peras', 'mandarinas', 'limones']

while len(frutas) > 2:
    frutas.pop()
    print(frutas)
```

```
['manzanas', 'platanos', 'peras', 'mandarinas']
['manzanas', 'platanos', 'peras']
['manzanas', 'platanos']
```

Figura 4: Ejemplo de bucle **while**.

Estos dos tipos de bucles se pueden usar de forma indistinta, ya que se puede construir un bucle **for** que sea equivalente a un bucle **while** y viceversa. En la figura 5 se muestra un ejemplo de esta equivalencia. En esta figura se recoge un bloque de código con un bucle **for** que produce el mismo resultado que el bloque de código de la figura 4 con un bucle **while**.

```
frutas = ['manzanas', 'platanos', 'peras', 'mandarinas', 'limones']

for fruta in frutas[::-1]:
    frutas.pop()
    if (len(frutas) < 2):
        break
    print(frutas)
```

```
['manzanas', 'platanos', 'peras', 'mandarinas']
['manzanas', 'platanos', 'peras']
['manzanas', 'platanos']
```

Figura 5: Ejemplo de equivalencia entre un bucle **for** y un bucle **while**.

1.3. ¿Qué es una lista por comprensión en Python?

Una lista por comprensión en Python es una forma de crear listas con una sintaxis más corta, es decir, escribiendo menos líneas de código. Por ejemplo, suponga que desea crear una lista con los números del 1 al 10 elevados al cuadrado. Esto se puede hacer fácilmente utilizando un bucle **for**, tal y como se muestra en la figura 6.

```
numeros_cuadrados = []

for i in range(1, 11):
    numeros_cuadrados.append(i**2)

print(numeros_cuadrados)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Figura 6: Creación de una lista utilizando un bucle **for**.

No obstante, existe una forma más corta de hacer esto mismo, utilizando una lista por comprensión. En la figura 7 se muestra el código para definir la misma lista del ejemplo anterior utilizando una lista por comprensión.

```
numeros_cuadrados = [x**2 for x in range(1, 11)]
print(numeros_cuadrados)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Figura 7: Creación de una lista utilizando una lista por comprensión.

Como puede ver, las listas por comprensión son una forma más elegante y corta (en términos de líneas de código) de definir una lista que utilizando un bucle.

1.4. ¿Qué es un argumento en Python?

Una **función** de Python es un bloque de código que se puede reutilizar y ejecuta una tarea determinada. Estas funciones, generalmente, toman argumentos de entrada. Los **argumentos** son información que entra en la función para producir una salida o resultado. En la figura 8 se muestra un ejemplo de función y los argumentos que toma como entrada.

```
def suma(num_1, num_2):
    return num_1 + num_2

print(suma(10, 5))
```

```
15
```

Figura 8: Función **suma** que toma como argumentos de entrada dos números **num_1** y **num_2**.

Este bloque de código define la función **suma**, que toma como argumentos de entrada dos números cualesquiera **num_1** y **num_2** y devuelve la suma de ambos números. A modo de ejemplo, se ha impreso el resultado de sumar los números 10 y 5, dando el resultado de 15.

Existen dos formas de pasar valores a una función: **por posición** y **por nombre**. Cuando se pasan valores por posición a una función estos se reciben por orden en los argumentos definidos. En la figura 8 los valores 10 y 5 se han pasado por posición, de tal manera que el argumento **num_1** es igual a 10 y el argumento **num_2** es igual a 5. En este caso es indiferente el orden en que se pasan los argumentos porque la suma de dos números es conmutativa y el resultado es el mismo. Sin embargo, considere la función definida en la figura 9.

```
def saludo(nombre, apellido):
    print(f'Buenos días {nombre} {apellido}')

saludo('Jorge', 'Dominguez')

saludo('Dominguez', 'Jorge')
```

```
Buenos días Jorge Dominguez
Buenos días Dominguez Jorge
```

Figura 9: Función **saludo** que toma como argumentos de entrada dos cadenas **nombre** y **apellido**. Llamada por posición.

Esta función toma como argumentos de entrada dos cadenas **nombre** y **apellido** e imprime un saludo en pantalla. En la primera llamada a la función se han pasado los valores **'Jorge'** y **'Dominguez'** a los argumentos **nombre** y **apellido**, respectivamente. El resultado que produce la llamada a la función es el saludo **'Buenos días Jorge**

Dominguez'. En la segunda llamada se han pasado los valores **'Dominguez'** y **'Jorge'** a los argumentos **nombre** y **apellido**, respectivamente. El resultado que produce la llamada a la función es el saludo **'Buenos días Dominguez Jorge'**. Como puede comprobar, el resultado obtenido en ambas llamadas a la función es diferente, ya que el orden en que se pasan los valores a los argumentos es fundamental en la definición **por posición**.

Si no recuerda el orden en que debe pasar los valores a la función pero recuerda los nombres de los argumentos resulta útil pasar valores a la función **por nombre**. En este caso, en la llamada a la función se especifica que valor se le asigna a cada argumento utilizando su nombre. En la figura 10 se puede ver un ejemplo.

```
def saludo(nombre, apellido):
    print(f'Buenos días {nombre} {apellido}')

saludo(nombre = 'Jorge', apellido = 'Dominguez')

saludo(apellido = 'Dominguez', nombre = 'Jorge')
```

```
Buenos días Jorge Dominguez
Buenos días Jorge Dominguez
```

Figura 10: Función **saludo** que toma como argumentos de entrada dos cadenas **nombre** y **apellido**. Llamada por nombre.

La función utilizada en este ejemplo es la misma que la del ejemplo anterior. Sin embargo, nótese la diferencia en las llamadas a la función. En ambas llamadas se define que valor toma cada argumento utilizando su nombre. De esta manera, el resultado que imprime la función es igual en ambas llamadas independientemente del orden en que se pasen los valores a los argumentos.

1.5. ¿Qué es una función Lambda en Python?

Una función Lambda es una función anónima que generalmente se define en una única línea, es pequeña y de carácter temporal. Para definir una función lambda se utiliza la palabra clave **lambda** seguida de los argumentos que toma la función, dos puntos (:) y el bloque de código. En la figura 11 se muestra un ejemplo.

```
suma = lambda num_1, num_2: num_1 + num_2
print(suma(10, 5))
```

```
15
```

Figura 11: Función lambda que computa la suma de dos números.

En este bloque de código se ha definido una función lambda que toma dos números **num_1** y **num_2** como argumentos y computa su suma. Esta función lambda se ha asignado a la variable **suma**. Posteriormente se llama a esta función lambda pasando los números 10 y 5 como argumentos y se imprime en pantalla el resultado, 15.

Como ya se ha mencionado, las funciones lambda son útiles para definir funciones pequeñas y de carácter temporal. En ningún caso se aconseja su uso para definir funciones complejas, ya que afectan sobremanera a la legibilidad del código.

1.6. ¿Qué es un paquete pip?

El **pip** o Python Package Installer es un sistema de gestión de paquetes que se utiliza para instalar, actualizar y administrar bibliotecas y módulos de Python. Este gestor permite descargar librerías creadas por otros desarrolladores desde la página `https://pypi.org/` y utilizarlas en tus programas de Python. Para instalar un paquete se debe utilizar el comando **pip install** seguido del nombre del paquete que se desee instalar.