Preguntas teóricas M3C5

DOCUMENTACIÓN SOBRE CONCEPTOS DE PYTHON

- ¿QUÉ ES UN CONDICIONAL?

Un condicional es una estructura de control que permite ejecutar un bloque de código basado en una condición de tipo booleano; es decir, evalúa si es verdadera o falsa dicha condición. La sintaxis que se utiliza para evaluar la condición es **if**.

Hay otra sintaxis, **else**, que se utiliza en caso contrario a **if**; es decir, si la primera condición no se cumple, el primer bloque de código no procedería a ejecutarse, pero sí el segundo, el que se introduce en **else**.

Por otra parte, se podrían anidar condiciones y ahí existe también la sintaxis **elif** en Python, en la que si la condición de **if** no es cierta, pasa a consultar otra condición y si el resultado que arroja es correcto, ejecutará el bloque de código, sino, pasaría al último bloque que sería el mencionado anteriormente: **else**.

Un ejemplo claro es:

```
x = 5
if x > 0:
  print('El número es positivo.')
elif x < 0:
  print('El número es negativo.')
else:
  print('El número es 0.')
```

El ejemplo anterior, traducido a un caso normal en la vida real, lo que está dando es un número asignado a la variable “x”, que en este caso es 5.

Lo que se le pediría sería: “Si el número es mayor que 0: imprime que el número es positivo; sino si el número es menor que 0: imprime que el número es negativo; y si no cumple ninguna de las dos anteriores, quiere decir que el número es igual a 0”.

Este tipo de estructura de control funciona muy bien para controlar, por ejemplo, el acceso a sistemas o páginas web en las que existen bases de datos y hay usuarios de tipo administrador o usuarios sin permisos; en las que deben introducir un nombre de usuario y una contraseña y a la que internamente se le debe consultar si el nombre de usuario introducido pertenece a un usuario administrador o no, para poder permitirle el acceso en caso de serlo. También se utiliza mucho para validar la edad de un usuario en cuanto a ciertas redes sociales o webs que no son infantiles y deberían ser controladas por adultos. Por ejemplo, Youtube, pide la comprobación de edad, en caso de no tener la edad mínima que exigen, no permiten el acceso a dicha plataforma, pero sí a la plataforma Youtube Kids ya que el contenido es más específico para el público infantil.

- ## ¿CUÁLES SON LOS DIFERENTES TIPOS DE BUCLES EN PYTHON? ¿POR QUÉ SON ÚTILES?

Los bucles (loops) son estructuras que permiten ejecutar un bloque de código de forma repetitiva hasta que se cumpla una condición o se termine un iterable; ya sea con una variable inicializada y con un fin definido o que el iterable sea una lista (o tupla, diccionario, etc.) y el bucle ejecute el código hasta que termine la longitud de la lista.

Los dos tipos de bucles que hay en Python son:

- **for**: se utiliza para iterar sobre una secuencia o cuando se conoce el final del iterable. Normalmente utilizado en listas, tuplas, diccionarios… Y habitualmente se sabe el número de veces que va a recorrer el bloque de código.
- **while**: se utiliza para ejecutar un bloque de código mientras se cumpla la condición que se de, por lo que normalmente no se conoce cuántas veces se va a iterar.

Ejemplo **for**:

```
frutas = ['manzana', 'pera', 'naranja', 'fresa', 'melocotón']
for fruta in frutas:
  print(fruta)
```

En este ejemplo recorre la lista “frutas” con la variable “fruta” imprimiendo cada vez una de la lista. El resultado que aparecerá por consola es el siguiente:

```
manzana
pera
naranja
fresa
melocotón
```

Ejemplo **while**:

```
numero = 0
while numero < 10:
  print(numero)
  numero += 1
```

En este ejemplo, se declara la variable “numero” con un valor inicial de 0 y se condiciona a que se cumpla el bloque de código dentro del **while** mientras número sea menor que 10; por lo que imprimirá por consola el número y después lo aumentará en 1 para continuar el bucle hasta que se cumpla la condición dada. El resultado que aparecerá por consola es:

Solo alcanzará el número 9 ya que en la condición se le especificó que fuese menor de 10, por lo que no cuenta con ese número. El bucle termina en cuanto comprueba que el número es igual o mayor que 10 y ya no cumple con la condición que se le dio.

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

Los bucles se utilizan mucho, al igual que los condicionales, para la validación de entrada de usuario, para verificar, por ejemplo, si un nombre está en una lista de usuarios registrados; también se utiliza para automatizar tareas tales como operaciones matemáticas o incluso para comprobar que la persona que está tratando de realizar alguna acción en alguna página web no es un bot y es un humano, como muchas páginas ya lo están implementando y piden, por ejemplo, introducir un número o carácter concreto. Y mientras no se introduzca el valor que se solicita, se seguirá pidiendo por pantalla hasta que sea correcto o se cumpla.

- ## ¿QUÉ ES UNA LISTA POR COMPRENSIÓN EN PYTHON?

Una lista por comprensión es una manera de crear listas en Python utilizando una sola línea de código en conjunto de bucles y condicionales. De esta manera, la sintaxis queda más legible y concisa.

Ejemplo de una lista de comprensión:

```
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
numeros_pares = [numero for numero in números if numero % 2 == 0]
print(numeros_pares)
```

En el ejemplo anterior tenemos una variable llamada “numeros” en la que se define una lista del 1 al 10. Después creamos una lista por comprensión, llamada “numeros_pares” y en ella definimos, en la misma línea de código, lo que queremos almacenar: el elemento “numero” definiría cada número que va a recorrer el bucle **for** dentro de la lista “números”; después tenemos la condición **if** que nos dice que si el módulo del número que recorre en ese momento el bucle, es igual a cero, quiere decir que es par, por lo que lo tiene que almacenar en la variable. Si no cumple esa condición, continúa con la iteración y no hace nada. Esta lista de comprensión se equivale con el siguiente bloque de código:

```
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
numeros_pares = []
for numero in numeros:
  if numero % 2 == 0:
    numeros_pares.append[numero]
print(numeros_pares)
```

Equivale a la misma salida `[2, 4, 6, 8, 10]` en ambos casos, solo que con la lista de comprensión se ejecuta en una línea y es más legible y más sencillo a la hora de guardar los elementos en la lista.

- ## ¿QUÉ ES UN ARGUMENTO EN PYTHON?

Es un valor que se pasa a una función cuando se la llama y se utilizan para que la función pueda realizar las operaciones y ejecutar el código que tienen. Se pueden pasar uno o varios valores, diferentes tipos de datos: numéricos, cadenas, listas, etc.

Ejemplo de argumento en una función:

```
def saludar(nombre):
  print(f'Hola {nombre}')

saludar('Arantza')
```

El argumento declarado en la función es “nombre” y cuando se le llama a dicha función es “Arantza”. Sin ese argumento, la función fallaría ya que requiere de él para cumplir su tarea.

Este ejemplo pasaría por consola lo siguiente:

```
Hola Arantza
```

El “hola “ sería el bloque de código de la función al que se le añade el argumento solicitado y que le hemos pasado “Arantza”, por lo que la concatenación de la cadena del saludo y el argumento, es lo que daría el resultado anterior en consola.

Hay varios tipos de argumentos que se le pueden pasar a las funciones:

- Argumentos **posicionales**: que se pasan a la función en el mismo orden en el que se definen a la hora de crear dicha función.

  ```
  def suma(a, b):
    return a + b

  resultado = suma(3, 5)
  ```

  En el ejemplo anterior, el número 3 equivale a la declaración del argumento “a” y el número 5 al argumento “b”, en caso de establecerlos al revés, se podría generar un resultado diferente. En este caso al ser una suma no habría problema, pero en caso de ser una resta, por ejemplo, el resultado cambiaría, por lo que es importante pasarle correctamente el valor en la misma posición que esté declarado en la función, ya que podría haber errores.
- Argumentos de **palabra clave**: se pasan a la función utilizando el nombre que se utilizó a la hora de definirlos en la función, de esta manera, se puede cambiar el orden de los argumentos o incluso omitir algunos de ellos.

  ```
  def resta(a, b):
  ```

```
    return a - b

  resultado = resta(b = 3, a = 5)
```

Siguiendo la base del ejemplo de los argumentos posicionales y habiendo hablado de las restas, si esta función se hubiera hecho en el anterior ejemplo, el resultado hubiera sido "-2", de esta manera, al darle el nombre definido, se puede cambiar el orden y la operación se hace correctamente.

- Argumentos por **defecto**: tienen un valor ya predeterminado a la hora de definirlos en la función, por lo que si, al llamar a la función, no se les pasa ningún valor, se utilizará el que se definió al crear la función. En caso contrario, si se le pasa cualquier otro valor, se sobrescribirá y el resultado será en base a este último valor.

```
  def saludar(nombre = 'Arantza'):
    return f'¡Hola {nombre}!'

  print(saludar())           # Esto imprimirá '¡Hola Arantza!'
  print(saludar('Debora'))   # Esto imprimirá '¡Hola Debora!'
```

- ## ¿QUÉ ES UNA FUNCIÓN LAMBDA EN PYTHON?

Es una función pequeña que puede tener varios argumentos, pero solo una expresión. Se utiliza para definir funciones en una sola línea para una operación específica y sin necesidad de definirla con la palabra clave **def** como en el resto de funciones.

Ejemplo de función lambda:

```
cuadrado = lambda num : num ** 2
resultado = cuadrado(4)
print(resultado)
```

Esto equivale a la siguiente función:

```
def cuadrado(numero):
  resultado = numero ** 2
  return resultado

print(cuadrado(4))
```

Ambas devuelven el mismo resultado, que sería "16", solo que con la función lambda el código es más corto y se define la función en una sola línea, pero tienen la misma funcionalidad. Normalmente para este tipo de funciones pequeñas, se recomienda la función lambda ya que es más legible, el resto de funciones definidas con **def** suelen incluir más líneas de código y funcionalidades más completas.

- ## ¿QUÉ ES UN PAQUETE PIP?

PIP es el sistema de gestión de paquetes de Python y se utiliza para instalar o administrar paquetes de software. Un paquete PIP es un conjunto de archivos que tiene código, recursos y metadatos para ampliar las capacidades de Python a través de bibliotecas, herramientas, frameworks, etc. Con el siguiente código se instalan los paquetes que se puedan necesitar de terceros para poder implementarlos en nuestros programas Python:

```
pip install nombre_paquete
```

Por ejemplo, un paquete concreto:

```
pip install numpy
```

Esto instalaría el paquete **numpy** que es bastante completo y potente para trabajar en nuestros códigos y programas Python.