CHECKPOINT 5

Índice

¿Qué es un condicional?

Un condicional (*if*) es para ejecutar (o no) unas lineas de código, dependiendo de una condición. Si la respuesta es verdadera, se añade una o varias tareas. Se puede añadir unas líneas de código también para el caso de que la condición sea falsa (utilizando la palabra *else*). Hay una tercera opción, que es añadir una condición cuando la anterior es negativa (*elif*). Esta opción se puede utilizar más de una vez: si la condición principal no se cumple, se evaluará la condición de la primera *elif*. Si es falsa, se evaluará la segunda, etc.

Por ejemplo:

```
if edad < 18:
        print("No eres mayor de edad")
elif edad >=18 and edad <70:
        print("Puedes participar en la actividad")
else:
        print("Demasiado mayor para la actividad")
```

En este caso, la condición es si la persona tiene menos de 18 años. Si la condición es verdadera, se ejecuta la tarea *print("No eres mayor de edad").*

En caso de que tenga más de 18 años, elif propone otra condición: tener entre 18 y 70 años. Por último, se utiliza *else* para ejecutar *print("Demasiado mayor para la actividad")* para los que no cumplen las dos condiciones anteriores.

A la hora de escribir la condición, se puede utilizar más de una condición haciendo uso de *and* (hay que cumplir las dos condiciones) o la palabra *or* (una o la otra).

Por ejemplo:

```
if edad < 18 and viene_solo == True:
        print("No eres mayor de edad y sin adulto no puedes entrar")
```

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

Hay dos tipos de bucles en Python: *for* y *while*. Las dos opciones son parecidas y sirven para aprovechar un trozo de código al repetir mismas acciones una y otra vez.
For es útil si conocemos cuantas veces hay que ejecutar (sobre una lista, tupla o diccionario conocido) y se utiliza más que *while*. En cambio, *while* se utiliza cuando queremos que se ejecute un código las veces que haga falta, mientras se cumple una condición. Hay que tener cuidado que no entre en un bucle infinito (si la condición se cumple siempre, no saldrá del bucle).

En caso del bucle *for*, el elemento para iterar puede ser un string, una lista, rangos,…

Por ejemplo:

```
lista1 = [“casa”,”garaje”,”coche”]
for objeto in lista1:
        print(objeto)

lista2 = range(1,11)
for num in lista2:
        print(num)
```

En caso del bucle *while,* en este ejemplo se escribirá “hola” hasta que el numero sea 0 y entonces se terminará:

```
numero = 15
while numero > 0:
        print(“hola”)
        numero -=1
```

Para controlar los bucles, existen palabras reservadas: *break* (sale del bucle) y *continue* (salta el código que falta en la iteración actual y pasa a la siguiente iteración).

¿Qué es una lista por comprensión en Python?

Lista por comprensión es otra manera de hacer un bucle. Es más compacta pero más difícil de entender.

Hay que poner entre corchetes lo que se ejecutaría. Luego *for* y la lista para iterar. Por último, como opcional, se puede añadir una condición con *if*.

Ejemplo 1: el resultado sería una lista con los números de 1 a 10, todos al cuadrado.

*numeros_cuadrados = [num**2 for num in range (1,11)]*

Ejemplo 2: Aquí sería la misma lista, eliminando los números impares.

*numeros_cuadrados_pares = [num**2 for num in range (1,11) if num%2 ==0]*

¿Qué es un argumento en Python?

Un argumento es una variable que se utiliza en una función como valor de entrada. La función recibe el valor y trabaja con dicho valor. A la hora de crear la función, se pueden utilizar cero, uno o varios argumentos que se definen entre paréntesis justo después del nombre de la función.

En el momento de utilizar la función, le decimos qué valor asignamos a ese o esos argumentos.

Por ejemplo:

```
def cuantos_cumples (num):
    print (f"Felicidades, hoy cumples {num} años!")
```

En este ejemplo, *cuantos_cumples* es el nombre de la función y *num* es el único argumento que tiene esta función.

Se puede poner el número:

```
cuantos_cumples(4)
```

También a través de una variable:

```
num_cumple = 5
cuantos_cumples(num = num_cumple)
```

Hay diferentes tipos de argumentos:

1. Argumentos posicionales. Son los más comunes. Hay que tener en cuenta el orden, por ejemplo si definimos

   ```
   def mi_funcion (nombre, apellido):
       print(f'Mi nombre es {nombre} {apellido}')
   ```

   Después hay que utilizar el nombre y apellido en el orden correcto:

   ```
   mi_funcion("Penelope", "Cruz")
   ```

   Otra opción sería poner el nombre del argumento:

   ```
   mi_funcion(apellido = "Cruz",  nombre ="Penelope")
   ```

2. Argumentos por defecto: a la hora de definir la función, podemos asignarle un valor que será utilizado en caso de que al llamar a la función no se le asigne ningún valor. Siguiendo con el ejemplo anterior:

   ```
   def mi_funcion (nombre, apellido="Cruz"):
       print(f'Mi nombre es {nombre} {apellido}')
   ```

mi_funcion (“Mar”) → Resultado “Mi nombre es Mar Cruz
mi_funcion (“Mar”,”Sanchez”) → Resultado: “Mi nombre es Mar Sanchez

3. Argumentos arbitrarios (*args): puedes incluir diferente longitud de variables y en la función especificar cómo se utilizarán. Se puede utilizar * y el nombre que elijas, pero es mejor utilizar **args*.
   El ejemplo que nos ofrece el tutorial de la página oficial es:

   ```
   def concat (*args, sep ="/"):
       return sep.join(args)
   concat (“earth”,”mars”,”venus”) → Resultado: “earth/mars/venus”
   ```

4. Argumentos **kwargs permite pasar argumentos de longitud variable y se utilizará en la función como par clave y el valor asignado. En este caso también se puede utilizar ** y el nombre que elijas, pero es mejor utilizar ****args*.

La página oficial de python tiene toda la información bien explicada y con buenos ejemplos:

https://docs.python.org/3/tutorial/controlflow.html#defining-functions

¿Qué es una función Lambda en Python?

La función lambda (también llamada función anónima) es una manera compacta y rápida de hacer funciones. Hay que poner la o las variables, dos puntos y el código que queremos para la función (sólo admite una expresión).

Por ejemplo:

*num_cuadrados = lambda num : print(num**2)*

num_cuadrados(2) → En la pantalla aparecerá el resultado: *4*

Tiene el inconveniente que no es tan fácil entender el código si el/la programadora es principiante.

Un buen tutorial sobre la función lambda es: https://realpython.com/python-lambda/

¿Qué es un paquete pip?

A la hora de programar, no es necesario inventar la rueda. Existen librerías en python con funciones que nos pueden servir y así tendremos nuestro código listo más rápido, más corto, más fácil para entender y seguramente con menos errores.

Pip es un gestor de paquetes para buscar, instalar y/o administrar las librerías que están en Pipy (Python Package Index). En la página web https://pypi.org/ podemos encontrar los paquetes disponibles, toda la información de cómo instalar y los pasos a seguir para publicar nuestras librerías.

Cuando queramos instalar una librería, lo buscamos en Pipy (por ejemplo *numpy*) y en la línea de comandos escribimos:

pip install numpy

y para utilizar en python:

```
import numpy as np
num_range = np.range(16)
```

Observaciones generales

A la hora de programar (condicionales, bucles, funciones, …) se cometen errores. Esta información se ha obtenido en las páginas web:

https://www.linkedin.com/pulse/10-errores-comunes-en-python-que-pasan-la-primera-semana-aldo-lares/

Y otras tres páginas web interesantes:

1.- La información en la página oficial:

https://docs.python.org/3/library/exceptions.html

2.- Un foro para poder resolver problemas (seguramente otras personas ya han tenido la misma duda y está resuelta):

https://stackoverflow.com/

3.- Tutoriales

https://www.w3schools.com/

Estos son los más comunes para poder evitarlos y poder entender la explicación del error que aparece en la pantalla.

SyntaxError: Cuando hay un error de sintaxis. Por ejemplo, en caso de un condicional, hay que poner dos puntos después de la condición.

Mal:

```
if x > 100
   print("x es mayor que 100")
```

Bien:

```
if x > 100:
   print("x es mayor que 100")
```

IndentationError: Hay que tener cuidado con la indentación (tabulacion y espacios)

Mal:

```
if edad < 18:
print(“No eres mayor de edad”)
```

Bien:

```
if edad < 18:
    print("No eres mayor de edad")
```

NameError: cuando la variable no está definida

TypeError: cuando se intenta mezclar diferentes tipos de datos