Course Checkpoint 5

¿Qué es un condicional?

Si hace frío en invierno, cierro la puerta, y enciendo la estufa de leña.
Si la temperatura sube de 28ºC en verano, pongo en marcha el aire acondicionado.
Ambas sentencias son condicionales, se utilizan muchísimo en los lenguajes de programación, y vienen a decir que, si se cumple una expresión/condición que solicitamos en el programa, reaccionaremos ejecutando una serie de comandos, u otros.

En python tenemos 3 tipos de condicionales:

if: es el principal condicional. Se utiliza como si fuera un comparador, si el resultado de la comparación es verdadero, se ejecutará una serie de sentencias que escribiremos a continuación.

elif: este condicional siempre irá debajo de **if**. Si el resultado de **if** fuera falso, podríamos añadir otro/otros condicionales **elif** para ver si alguno de ellos se cumple, y en ese caso ejecutar los comandos que fueren.

else: el coche escoba. Si ninguno de los condicionales que se plantean por encima de else se cumplen, entonces llegaremos a éste y ejecutaremos el código que corresponda.

Vamos con un ejemplo en el que recogeremos todos los condicionales con su estructura de codificación.
Tenemos un termómetro en casa, el cual nos da una lectura de temperatura. Según la temperatura varíe, nos dará un mensaje de alerta
para que interactuemos con el sistema de calefacción/refrigeración. La temperatura de confort se sitúa entre 18ºC y 24ºC, y en todo momento
el sistema nos informará de la misma con indicaciones para mantener la casa templada.

```
temperatura = 23

if temperatura >= 24:
        print("Enciende el aire acondicionado, Manuel!")

        #Si la temperatura es mayor o igual a 24ºC, ejecutaremos el código debajo de
        #if(indicaríamos a Manuel que ha de encender el aire acondicionado para tener un poco
        #fresquito en la casa) y finalizamos el condicional. Si la condición de temperatura no se
        #cumple/False, saltaremos al siguiente condicional (elif)

elif temperatura <= 18:
        print("Enciende la estufa, Manuel!")

        #Si la temperatura es menor o igual a 18ºC, ejecutaremos el código debajo de elif y
        #finalizamos el condicional. Anunciaríamos a Manuel la necesidad de encender la estufa
        #para calentar la casa.
else:
        print("Relájate Manuel, temperatura Ok")

        #A else llegamos porque ninguna de las condiciones previas se han dado por
        #cumplidas/True. Mandaríamos un mensaje de tranquilidad, pues en este caso D. Manuel
```

#no tendría que hacer sino relajarse, y daremos por finalizado el condicional.

¿Cuáles son los diferentes tipos de bucles en Python?

Decir que un bucle es algo que se repite durante un tiempo, o mientras se de una condición.
Dicho esto, tenemos 2 tipos diferentes:

For: Se utiliza para hacer repeticiones en un rango o de un número de iteraciones conocidos.
En el siguiente ejemplo vamos a crear una lista con los dias de la semana.
Seguidamente, el programa irá seleccionando paulatinamente todos los elementos de la lista, mostrándolos en pantalla.
El bucle for terminará su labor cuando haya pasado por todo el contenido de Semana.

```
semana = ["lunes", "martes", "miercoles", "jueves", "viernes","sabado", "domingo"]

for dia in semana: #Bucle en el que irá pasando por cada elemento de la lista llamada semana.
    print(dia)
```

While: Se harán repeticiones hasta que llegue alguna condición que haga detenerse el bucle.
En el siguiente ejemplo vamos a presentar en pantalla los números del 1 al 100.
Para ello crearemos una variable llamada conteo, cuyo valor iremos incrementando de a 1 a medida que vayamos imprimiéndola.

```
conteo = 0

while conteo <=100:  #la clave esta en este punto. Mientras el valor de conteo no alcance el valor
de                   #100, irá imprimiendo el número, y seguidamente sumándole 1, hasta llegar
                     #al valor 100, que entonces se dará la condición de parar.
        print(conteo)
        conteo += 1
```

¿Por qué son útiles?

Son muy útiles porque ahorran muchísimo trabajo a la hora de programar. En lugar de estar repitiendo iteraciones continuamente hasta que consigamos
el resultado que queremos, estos comandos, trabajarán en "modo automático", y van a ir haciendo el trabajo por nosotros además, en escasas lineas de código.

¿Qué es una lista por comprensión en Python?

Trata de ser una secuencia de comandos, que pueda leerse/interpretarse con lenguaje natural, de tal forma que se puede producir código facilmente, dando la sensación de que escribiendo directamente lo que queremos hacer, el programa al ejecutarse, lo hace... Magia!

La estructura de la misma es la siguiente: **[expresión for elemento in iteración]**
En este ejemplo vamos a generar una lista con el resultado de multiplicar los números del 1 al 6 por 2.

```
doble_de_numeros = [x * 2 for x in range(1, 6)]
```

La expresión nos dice: multiplica un número por 2 para todos los números que estén en el rango del

1 al 6 (y guardalos en la lista doble_de_numeros.

También podríamos encontrarnos con este otro tipo de estructura: [expresión for elemento in iteración if condición].
En este caso estaríamos añadiendo un condicional, que nos ayudará a precisar mejor los resultados de nuestro código.

```
numeros_pares = [x for x in range(1, 11) if x % 2 == 0]
```

La expresión quedaría así: Pasa todos los valores de x en el rango de 1 hasta el 11, siempre que el resultado de su división por 2 sea igual a 0.
Con esto lo que conseguimos es tener una lista de nombre numeros_pares en la cual tendremos almacenados todos los números pares del rango 1 hasta el 11.

¿Qué es un argumento en Python?

Argumento es uno o varios valores que se pasan a una función con los cuales ésta realice lo que esté escrito en el código.
En la función siguiente alimentamos la funcióncon 2 argumentos numéricos.
Seguidamente estos se suman y son asignados a la variable Suma, presentando después el resultado.
Al llamar a la función es cuando adjuntamos los argumentos.

```
def numeros_suma(a, b):
    suma = a + b
    print(suma)

numeros_suma(7, 23)
```

Podemos tener también en una función argumentos de diferentes tipos.
En el ejemplo adjunto generamos una tupla de argumentos variopintos.
La clave es la definición de los argumentos. Utilizamos un asterisco, para indicar que CUALQUIER cantidad de argumentos pueden ser pasados a la función, y a args -que es como una convención para nombrar a los argumentos en python- enviaremos la tupla generada.

```
def argumentos_posicionales(*args):
    print(args)

argumentos_posicionales(1, 7, "cocina", {"telediario", "ventana"})
```

Por último nos referiremos a **kwargs. La diferencia sustancial es que éste albergará un tipo de argumentos variables de palabra clave en un diccionario.
Lo utilizaremos cuando trabajemos con asignaciones propias de los diccionarios, donde la estructura suele ser palabra clave = valor.
En el siguiente ejemplo aclaramos el significado.

```
def argumentos_palabra_clave(**kwargs):
    for clave, valor in kwargs.items():
        print(f"{clave}: {valor}")

argumentos_palabra_clave(nombre= "miguel", edad = 20, ciudad = "Madrid", profesión = "trompetista")
```

A la función le añadimos una serie de argumentos por pares, que nos va a generar un diccionario, en el cual vamos a poder luego hacer modificaciones, consultas, listados, etc de forma mucho más eficaz e intuitiva.

¿Qué es una función Lambda en Python?

Es un recurso para crear funciones simples, que en momentos puntuales se utilizarán para resolver un pequeño problema, y que por su fugacidad y requerirse sólo para realizar una pequeña tarea no se computan como funciones ordinarias.

Su estructura es: lambda argumentos : expresión

Un gran ejemplo que he encontrado en ChatGPT es el que acompaño:

```
estudiantes = [
    ("Juan", 85),
    ("María", 90),
    ("Pedro", 75),
    ("Ana", 80)
]

# Ordenar la lista de estudiantes por calificación
estudiantes_ordenados = sorted(estudiantes, key=lambda x: x[1], reverse=True)

# Imprimir la lista de estudiantes ordenada
for estudiante in estudiantes_ordenados:
    print(estudiante)
```

Como va a ser una función que se va a utilizar exclusivamente para facilitar el ordenado, y no se va a volver a usar, utilizamos la función lambda.
Si fuera una función a la que tuvieramos que acudir repetidamente, de forma recursiva en diferentes puntos del programa, usaríamos ya una convencional.

¿Qué es un paquete pip?

Una herramienta que nos permite instalar/desinstalar paquetes de librerias a través de una página web de nombre Python Package Index (PyPI).
Allá los programadores suben sus librerias, y nosotros a través del comando pip install [nombre_del_paquete], podemos instalarlo en nuestro equipo para así
poder usar la librería en cuestión.
Como gestor de paquetes, también nos permite actualizar las librerias, listar las que tenemos instaladas, o incluso instalarlas desde otros repositorios,
o las que hayamos podido crear nosotros en nuestro pc.