Este checkpoint será un poco distinto, vas a poder utilizar documentación que busques por la web, porque el objetivo es que tienes que generar una documentación sobre las preguntas que te envío a continuación, tienes que prepararlas para que las personas que no tienen conocimientos puedan aprender, por lo que intenta ser lo más detallado posible, poner ejemplos, etc... Eres la persona responsable de crear la documentación que utilizarán los nuevos compañeros de tu equipo...

¿Qué es un condicional?

Un condicional es una estructura de control de flujo en el código. Esto nos permite ejecutar bloques concretos del código según cumplan esas condiciones, esto significa que si la condición es cierta, ejecutaría las líneas de código del bloque que sigue a esa condición.

Ejemplo en seudocódigo:

```
#Si cumple la condición
si  eres_mayor_de_edad:

        # ejecuta código 1
        Puedes entrar.

#Si no la cumple
si no:

        # ejecuta código 2
        No puedes entrar
```

Para definir estas condiciones se utilizan las palabras reservadas: *if, elif y else*. Tanto *if* como *elif* irían seguidas de una condición a cumplir, mientras que else se utiliza para definir una opción por defecto para los casos que no cumplan la condición previa, por lo que no necesitan condición alguna.

```
if edad > 30:
        print("Puedes entrar.")
elif edad > 17:
        print("Puedes entrar con descuento.")
else:
        print("No puedes entrar.")
```

Siguiendo este ejemplo, si una persona es mayor de 30 años, el programa imprimirá "Puedes entrar", pero no le indicará ninguna de las otras dos opciones.

Mientras si es una persona entre 18 y 30 años, no cumplirá la primera condición (if), pero sí la segunda (elif), por lo que le indicará "Puedes entrar con descuento".

Finalmente, si no cumple ninguna de las dos condiciones, entenderá que es un menor de edad y le indicará “No puedes entrar”.

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

Los bucles que disponemos en Python son el bucle **for** y el bucle **while**.
Son otra estructura de control de flujo que nos permite repetir secuencias de código.

En el caso del bucle **for**, nos resulta de utilidad para iterar o recorrer otras estructuras como listas, tuplas o diccionarios. Sus palabras reservadas serían *for*, *in*.

El seudocódigo sería:

```
#Sintaxis del bucle for
Si hay más elementos en la estructura:
    #Código
    Haz esto
```

Como ejemplo, si tenemos una lista de ingredientes en una receta y queremos imprimirlos todos, sería:

```
lista_ingredientes = [‘Azúcar’, ‘Agua’, ‘Limón’]
for ingrediente in lista_ingredientes:
    print(ingrediente)
```

Cada vez que el bucle usara la variable “ingrediente”, apuntaría al siguiente elemento de “ingredientes”. Así estaría imprimiendo cada ingrediente de la lista hasta imprimirlos todos.

Azúcar
Agua
Limón

El bucle **while,** tiene la utilidad de ejecutar el bloque de código mientras se cumpla una condición. Lo cual es útil para las condiciones que no sepamos cuando van a cambiar o para su uso como centinela si se genera una alerta.

```
#Sintaxis del bucle while
Mientras sea cierto:
    #Código
    Haz esto
```

Si por ejemplo queremos generar una impresión de números hasta n:

```
num = 0
```

```
    while num < n:
        print(num)
        num += 1
```

Cuando la variable “num” alcance el valor de “n”, saldrá del bucle **while** y dejará de imprimir la variable que hemos ido aumentado en cada iteración.

El flujo de ambos bucles puede ser alterado con el uso de *continue* y *break*. Si en el código interno establecemos una confición que se cumpla, y en su código a ejecutar usamos la palabra clave *continue*, pasaría directamente a la siguiente iteración del bucle. Mientras si usamos *break*, terminaría el bucle sin finalizarlo.

```
    lista_ingredientes = [‘Azúcar’, ‘Agua’, ‘Limón’]
    diabetico = True
    for ingrediente in lista_ingredientes:
        if diabetico == True and ingrediente == ‘Azúcar’:
            continue
        print(ingrediente)
```

Imprimirá:
Agua
Limón

```
    lista_ingredientes = [‘Azúcar’, ‘Agua’, ‘Limón’]
    diabetico = True
    for ingrediente in lista_ingredientes:
        if diabetico == True and ingrediente == ‘Azúcar’:
            break
        print(ingrediente)
```

No imprimirá nada, porque quién querrá sólo agua y limón...

¿Qué es una comprensión de listas en python?

La comprensión de listas nos permite crear una lista nueva tomando como base otra lista y aplicándole a cada elemento una expresión modificadora. Esta forma de crearlas es más eficiente y legible que si tratásemos estas operaciones en bucles que iterasen la lista y añadiesen cada elemento ala nueva, porque resolvemos en una línea lo que de otra forma serían tres o más.

Seudocódigo:

```
    mi_lista_uno = [elem_1, elem_2,...]
    mi_lista = [expresión for elemento in mi_lista_uno]
```

Expresión sería la modificación que se le aplicaría a cada elemento la lista a iterar.

Ejemplo de forma manual:

```
num_list = [1,2,3,4,5] #Lista base
cubed_nums = []     #Lista destino

#Recorre la lista base
for num in num_list:
    #Añade el cubo de cada elemento base a la lista destino
    cubed_nums.append(num ** 3)
```

Dará como resultado una lista cubed_nums == [1, 8, 27, 64, 125]

Si usamos la compresión de listas:

```
num_list = [1,2,3,4,5]
cubed_nums = [num ** 3 for num in num_list]
```

Lo resolvemos en una línea para obtener una lista con el cubo de cada elemento de la primera lista.

¿Qué es un argumento en Python?

En Python tenemos funciones y métodos que ejecutan un bloque de código cuando son llamadas. Estas funciones pueden necesitar una serie de datos que tenemos que proporcionarles para operar o para modificarlos y generar una respuesta. Estos datos los facilitamos al llamarles mediante los argumentos.

El argumento será usado como una variable en la definición de la función y que adoptará el valor que le asignemos en la llamada.

Ejemplo:

```
#Definición de la función
def mi_función(argumento_1, argumento_2):
    return argumento_1 * argumento_2

#Llamada a la función
resultado = mi_función (5, 2)
```

Estos argumentos pueden tener también un valor por defecto al definir la función y que usará si no se ha proporcionado ese argumento.

Ejemplo:

```
#Definición de la función
def mi_función_saludo(nombre, saludo = “Hi,”):
```

```
            print(saludo, nombre)

    #Llamada a la función
    resultado = mi_función_saludo ("Peter")
```

Imprime: "Hi, Peter"

¿Qué es una función de Python Lambda?

Es una forma de crear funciones anónimas (no tienen nombre y generalmente pequeñas), que pueden tener varios argumentos y una expresión a cumplir. Útiles para situaciones breves como argumento de una llamada a otra función.

Pseudocódigo:

```
    lambda  argumentos : expresión
```

Ejemplo:

```
    suma= lambda x,y : x + y
```

Si la usamos en la llamada de una función o método podría ser:

```
    lado = 5
    area_cuadrado = lambda x: x * x
    print(area_cuadrado(lado))
```

Lo que imprimirá el área de un cuadrado al llamar a print.

¿Qué es un paquete pip?

Es un módulo de Python que se guarda en el repositorio oficial PyPI y que se gestiona a través de sus sistema pip (Pip Instalador de Paquetes).

Esto nos ofrece una librería enorme de paquetes con sofware encapsulado para usos de todo tipo, siendo necesario usar cada paquete para su porpósito concreto. Puede haber orientados a cálculos matemáticos, como a la obtención de datos online para Machine Learning. La librería es inmensa y te ahorra muchos esfuerzos a la hora de escribir código, dado que muchos desarrolladores ya lo han hecho antes al crear dicho paquete para que puedas usarlo.

La forma de instalarlos es usando la siguiente línea de código en la consola de comandos:

```
    pip install nombre_del_paquete
```

O en caso de querer desinstalarlo:

```
pip uninstall nombre_del_paquete
```