Borja Gonzalvo González Checkpoint 5

¿Qué es un condicional?
En el lenguaje de programación Python, un condicional es, una pequeña estructura de datos que podemos hacer para que el propio programa sea capaz de tomar decisiones, según los parámetros que le hayamos marcado.
Podríamos explicarlo como una instrucción en la que el programa tiene que evaluar si hace unas tareas, si se cumplen determinadas condiciones marcadas o si hace otras,si se no se cumplen.
Ejemplo en la vida real:
Si llueve voy al trabajo en coche, sino llueve voy en bicicleta.

En Python funciona por medio de la respuesta Verdadera o Falsa (True o False) a la condición de si se cumple(if) o sino (else) se cumple lo que le hemos marcado.
Ejemplo: si llueve es igual a verdadero la respuesta sera ir en coche , si es falso la respuesta sera ir en bicicleta.

Ejemplo sencillo en Python:

```
clima = "llueve"
if clima == "llueve":
    print("Voy al trabajo en coche.")
else:
    print("Voy al trabajo en bicicleta.")
```

El programa comprueba que clima es igual a llueve, devuelve la respuesta True y imprime la salida voy al trabajo en coche.

Partes de un condicional

- if(si): Es la palabra clave que inicia el condicional. Después de if`, ponemos la condición que puede ser verdadera o falsa en su respuesta.
- Condición: Es lo que consultamos o verificamos , como clima es igual a llueve en nuestro ejemplo,esta condición se evalúa como verdadera o falsa.
- Cuerpo del if: Son las acciones que se realizan si la respuesta es verdadera. En nuestro ejemplo, sería : print(Voy al trabajo en coche).
- else (si no): Es opcional. Se usa para dar una opción a que pasa si la condición es falsa,en nuestro ejemplo, sería: print(Voy al trabajo en bicicleta).

Las condiciones a consultar pueden ser cualquier instrucción que se su respuesta pueda dar verdadera o falsa, no solo si es igual a alguna pauta marcada o no, también podemos consultar si es mayor o menor por ejemplo.

Ejemplo:
Supongamos que tienes que decidir si tu hijo puede jugar a videojuegos según la hora del día:

```
hora_del_dia = 15
if hora_del_dia < 12:
    print("Es de mañana, puede jugar.")
elif 12 <= hora_del_dia < 18:
    print("Es de tarde. Puede jugar.")
else:
    print("Es de noche. No puede jugar.")
```

En este ejemplo, he añadido otra palabra clave : elif` (abreviatura de "else if"). Esto nos da opción a verificar múltiples condiciones.
Si hora del día es menor que 12, se imprimirá "Es de mañana. Puede jugar."
Si hora del día está entre 12 y 18 (exclusivo), se imprimirá "Es de tarde. Puede jugar."
Si ninguna de las condiciones es verdadera, se imprimirá "Es de noche. No puede jugar."

En resumen un condicional nos permite tener un control del programa, basándonos en decisiones , es muy importante para poder generar programas que nos den respuestas a diferentes situaciones y entradas.

Existen también los condicionales anidados y los operadores lógicos:
Los condicionales anidados son un condicional dentro de otro condicional. Se utilizan para comprobar distintas variables, de forma parecida al elif. Pero con los condicionales anidados nos permite valorar cada requisito independientemente del resto, y nos da mayor precisión en consultas complicadas.

Para formar las condiciones anidadas, añado a la explicación los operadores and y or:

- Or (o): nos permite dar una alternativa, puede cumplirse la condición del if o la del or. Ejemplo de si llueve que hemos detallado antes añadiéndole el operador or para decir que si nieva también vamos en coche: if clima == "llueve"or clima == "nieve" :
- And (y): nos da la opción de hacer que se cumpla las dos condiciones descritas.
  Ejemplo: if clima == “llueve” and dia_laboral == “True”.
- Operadores de comparación (`==`, `!=`, `<`, `>`, `<=`, `>=`)

En Python los bloques de código tienen que ir bien indentados(generalmente 4 espacios), es muy crucial que los bloques dentro de if , elif y else estén perfectamente indentados, para que el programa lo reconozca como parte del condicional.

Ejemplo con múltiples condiciones:

```
edad = 25
tiene_licencia = True

if edad >= 18 and tiene_licencia:
   print("Puedes conducir")
elif edad >= 18 and not tiene_licencia:
   print("Necesitas una licencia para conducir")
else:
   print("No puedes conducir")
```

En este ejemplo:

- Si la persona tiene 18 años o más y también tiene licencia, se imprimirá "Puedes conducir".
- Si la persona tiene 18 años o más pero no tiene licencia, se imprimirá "Necesitas una licencia para conducir".
- Si ninguna de las condiciones anteriores se cumple, se imprimirá "No puedes conducir".

=> ¿Cuáles son los diferentes tipos de bucles en Python? ¿Por qué son útiles?
Un bucle es la manera que tenemos de repetir instrucciones una y otra vez. En vez de escribir la misma instrucción varias veces,creamos un bucle para que repita la instrucción.
Existen dos tipos de bucle en Python:

- Bucle for: usamos este bucle cuando sabemos cuantas veces queremos repetir algo, la usamos en para a iterar sobre una secuencia (como una lista, una tupla, un diccionario, un conjunto o una cadena de caracteres) o cualquier objeto iterable.
- Bucle while: usamos este bucle cuando no sabemos cuantas veces vamos a repetir algo, y su repetición dependerá de una condición que puede cambiar.

Ejemplos de bucle for:
tenemos una lista de ciudades y queremos nombrar a cada ciudad:

```
ciudades = ['barcelona', 'madrid', 'sevilla']
for ciudad in ciudades:
   print(ciudad)
```

En este ejemplo su funcionamiento es:

- Tenemos una lista de ciudades con 3 elementos
- El bucle for coge cada elemento de lista uno por uno.
- La palabra ciudad es una especie de variable que en cada repetición del bucle, contiene uno de los elementos de la lista.. Tienes una lista llamada `frutas` con tres elementos: ['barcelona', 'madrid', 'sevilla'].
- La instrucción print(ciudad) nos imprime cada nombre de la ciudad en cada pasada del bucle.

Ejemplo de bucle while:
Necesitamos contar del numero 10 al 100:

```
contador = 10
while contador < 101:
   print(contador)
   contador += 1
```

En este ejemplo,su funcionamiento es:

- Tenemos la variable contador que empieza en el 10.
- Nuestro bucle while va a seguir repitiéndose mientras el contador sea menor que 101.
- La instrucción print(contador) nos imprime el numero en cada pasada del bucle.
- La instrucción contador += 1, hace aumentar en cada pasada en +1 la variable contador.
- En el momento que la variable contador llegue a 101, la condición ya no se cumple y el bucle se detiene.

Para poder tener un mayor control en los bucles, muchas veces necesitamos usar las instrucciones:

- break: utilizamos break para salir inmediatamente del bucle, se siga o no cumpliendo la condición. detallada.
- continue: utilizamos continue para saltarnos el resto de código en una vuelta de la iteración en activo y poder pasar a la siguiente vuelta del bucle.

Ejemplo de uso de break:
Queremos encontrar en una lista el numero 3:

```
numeros = [1, 2, 3, 4, 5]
for numero in numeros:
   if numero == 3:
      break
   print(numero)
```

Mediante break conseguimos que el programa imprima 1, 2 pero al encontrar el 3, el break hará que salga del bucle.

Ejemplo de uso de continue:
Queremos imprimir solo los numero impares de una lista:

```
for num in range(1, 16):
    if num % 2 == 0:
        continue
    print(num)
```

Mediante el continue hacemos que cada vez que encuentro que se da la condición , vuelva a reiniciar la vuelta del bucle, y así logramos no imprimir los numeros pares.

En resumen el bucle for es perfecto sobre secuencias y poder iterar en ellas y el while es ideal para cuando esa iteración va a depender de una condición,y con break y continue ardimos control y flexibilidad sobre la repetición.

Los bucles son una herramienta muy útil en programación, nos permite ejecutar de manera repetitiva bloques de código, de forma eficiente y flexible. Son esenciales para manejar tareas repetitivas, procesar grandes conjuntos de datos y escribir código limpio y que podamos hacerle mantenimiento.
Nos permiten procesar grandes cantidades de datos(listas,tuplas,diccionarios...)al permitirnos iterar sobre estas colecciones.

=>¿Qué es una lista de comprensión en Python?
Una lista de compresión es la manera mas rápida y fácil de crear listas en Python, en vez de utilizar el bucle for para agregar elementos de uno en uno , podemos crear una lista en una sola linea de código.
Ejemplos:

- Necesitamos crear una lista con los numeros al cuadrado desde el 0 al 4:

```
cuadrados = [i**2 for i in range(5)]
print(cuadrados)
```

Se genera la lista: `[0, 1, 4, 9, 16]` porque está elevando cada número al cuadrado.

- Necesitamos crear una lista con los numeros pared del 0 al 9:

```
pares = [i for i in range(10) if i % 2 == 0]
print(pares)
```

Se genera la lista `[0, 2, 4, 6, 8]` porque solo incluye los números que son pares

Las lista de comprensión pueden incluir los bucles for y la condiciones if,
Su forma de escribirse básica es :
[nuevo_elemento for elemento in secuencia if condición]

- `nuevo_elemento` es lo que quieres poner en la lista.
- `elemento` es cada valor de la secuencia que estás recorriendo.
- `secuencia` es la colección de datos que estás iterando (como una lista o un rango).
- `condición` es opcional y filtra los elementos que no cumplan con ella

Las ventajas principales de usar listas de comprensión serian:

- Código mas conciso, sencillo de leer y escribir
- Mejor rendimiento, ya que su uso es mas rápido que el uso de bucles for.

=>¿Qué es un argumento en Python?
Un argumento en Python es la información adicional que le pasamos a una función para que pueda realizar la tarea asignada. Son esenciales para que las funciones sean útiles y puedan hacer diferentes tareas dependiendo de los datos que le pasemos.
Los argumentos permiten a las funciones ser flexibles y reutilizables, haciendo posible que una función realice una amplia variedad de tareas con diferentes entradas.
Ejemplo en el que mediante un saludo nos presentamos:

```
def saludo(nombre, edad):
   return f"Me llamo {nombre} y tengo {edad} años."

print(presentar(edad=42, nombre="Borja"))
```

En este ejemplo:

- def: Es una palabra clave en Python que se usa para definir una función.
- saludo: Es el nombre de la función.
- nombre y edad: son los argumentos.

Ejemplo 2 en que mediante dos argumentos sumamos dos numeros:

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

- sumar: Es el nombre de la función.
- a y b: Son los argumentos,son dos variables que la función usará para hacer la suma.

Para usar esta función y darle dos números como argumentos, lo hacemos de la siguiente manera:

```
suma = sumar(4, 7)
print(resultado)
```

La salida será 7

Python trabaja con diferentes tipos de argumentos,los podemos clasificar en las siguientes categorías:

- Argumentos Posicionales:
  Los pasamos en el orden en que están definidos. El valor del primer argumento posicional sera el primer parámetro de la función , el segundo sera el segundo y así el resto.
  Ejemplo:

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

  resultado = sumar(4, 4)
  print(resultado)  # La salida sera 8
```

- Argumentos con Nombre :
  Los pasamos especificando el nombre del parámetro, sin importar el orden.
  Ejemplo:

```
def saludo(nombre, mensaje):
   print(f"{mensaje}, {nombre}")

saludo(mensaje="Hola", nombre="Borja")  # Salida: Hola, Borja
```

- Argumentos por Defecto:
  Tienen unos valores predeterminados que se usan si no se proporcionan otros valores.
  Ejemplo:

```
def saludo(nombre, mensaje="Hola"):
   print(f"{mensaje}, {nombre}")

 saludo("Cristina")  # Salida: Hola, Cristina
 saludo("Borja", "Buenos días")  # Salida: Buenos días, Borja
```

- Argumentos Variables:
  Nos permiten pasar un número variable de argumentos posicionales (*args) o con nombre (**kwargs).
  Ejemplo:
  def sumar_todo(*args):
  return sum(args)
  print(sumar_todo(1, 6, 2, 5)) # Salida: 14

  def informacion(**kwargs):
  for clave, valor en kwargs.items():
  print(f"{clave}: {valor}")
  informacion(nombre="Borja", edad=42, ciudad="Barcelona")
  # Salida: nombre: Borja edad: 42 ciudad: Barcelona

¿Qué es una función Lambda en Python?
En Python, una función lambda es una pequeña función anónima, es decir, una función que no está declarada con el nombre def tradicional. En vez de usar def`, las funciones lambda utilizan la palabra clave `lambda`. Las funciones lambda pueden tener cualquier número de argumentos pero sólo pueden contener una única expresión.
Las funciones lambda en Python son herramientas poderosas para definir funciones pequeñas y anónimas de manera concisa. Son especialmente útiles cuando se necesitan funciones cortas y temporales, como en el caso de las funciones de orden superior y otras operaciones simples. Aunque tienen sus limitaciones, las funciones lambda pueden mejorar la legibilidad y concisión del código cuando se usan adecuadamente.

La sintaxis básica de una función lambda es:
lambda argumentos: expresión

Características de las Funciones Lambda:

- Anónimas: Las funciones lambda no tienen un nombre explícito, por lo que son útiles para definir funciones cortas que se utilizan temporalmente.
- Resumidas: Se definen en una sola línea de código.
- Limitadas: Sólo pueden contener una expresión, no pueden incluir declaraciones múltiples o complejas.

Ejemplo
```
suma = lambda a, b: a + b
print(suma(4, 4))  # Salida: 8
```
En este ejemplo, `lambda a, b: a + b` crea una función que suma dos números. Esta función se asigna a la variable `suma`, que luego se puede llamar como cualquier otra función.

Los usos mas típicos de las funcionas Lambda son:

- Funciones de Orden Superior:
  Las funciones lambda son frecuentemente utilizadas con funciones de orden superior como map()Aplica una función a todos los elementos de un iterable., filter()Filtra elementos de un iterable según una función que devuelve `True` o `False`. y sorted()Ordena elementos de un iterable, opcionalmente usando una función clave.
- En funciones Temporales:
  Las funciones lambda se usan a menudo para crear pequeñas funciones que sólo se necesitan durante un corto período de tiempo y no requieren un nombre.

Las funciones Lamba tienen ciertas ventajas y limitaciones:

- Ventajas:Sintaxis Concisa: Ideal para funciones pequeñas y simples que se usan temporalmente. Legibilidad: Cuando se usan correctamente, pueden hacer que el código sea más fácil de leer al eliminar la necesidad de definir funciones completas para operaciones simples.
- Limitaciones:Sólo permiten una expresión.
- Complejidad: Para funciones más complejas o con lógica extensa, es mejor usar una función definida con def para mantener la claridad y la legibilidad del código.

=>¿Qué es un paquete pip?
Pip, (pip installs packages)es una herramienta que usamos en Python para instalar y gestionar paquetes, estos paquetes son una colección de código de Python que otros programadores escribieron para realizar trabajos específicos.
Utilizar estos paquetes nos permite ahorrar mucho trabajo y tiempo, ya que no tenemos que escribir el código desde cero.

Instalación y uso de paquetes PIP:

- Instalar Pip:en las versiones recientes d Python, ya esta instalado, para confirmar si esta o no instalado, ejecutamos : pip –version . Si no está instalado, podemos instalarlo desde la documentación oficial de Pip(https://pip.pypa.io/en/stable/installation/).
- Instalar un paquete: utilizamos el comando pip install seguido del nombre del paquete.
  Ejemplo para instalar el paquete requests: pip install requests
- Actualizar un paquete:si necesitamos actualizar un paquete a la última versión, usa el comando:
  pip install --upgrade seguido del nombre del paquete, para actualizar el paquete requests: pip install --upgrade requests
- Desinstalar un paquete:para poder desinstalar un paquete, usa el comando pip uninstall seguido del nombre del paquete, para el mismo ejemplo de paquete requests: pip uninstall requests
- Listar paquetes ya instalados:para ver que paquetes tenemos instalados usamos el comando: pip list

Ventajas de usar Pip

- Instalación sencilla:instalar y gestionar paquetes es muy sencillo y rápido.
- Amplia variedad de paquetes: tenemos una gran cantidad de paquetes y bibliotecas disponibles en PyPI.
- Manejo de dependencias: Pip gestiona automáticamente las dependencias de los paquetes, instalando cualquier otro paquete necesario para que el paquete principal funcione correctamente.
- Potabilidad: podemos crear un archivo `requirements.txt` con una lista de todas las dependencias de tu proyecto, lo que nos facilita la replicación del entorno en diferentes máquinas. Este archivo puede generarse con: pip freeze > requirements.txt y podemos usarlo para instalar todas las dependencias en otro entorno con: pip install -r requirements.txt