¿Qué es un condicional?

Un condicional en programación, específicamente en Python, es una estructura que permite tomar decisiones basadas en ciertas condiciones. Básicamente, te permite ejecutar un bloque de código si una condición es verdadera y otro bloque si esa condición es falsa. Esto es esencial para controlar el flujo de un programa y hacer que este responda de manera adecuada a diferentes situaciones.

En Python, los condicionales se implementan principalmente con la estructura if, que se puede complementar con elif (abreviatura de "else if") y else. Aquí hay una explicación de cada uno:

1. if: Se utiliza para verificar si una condición es verdadera y ejecutar un bloque de código en consecuencia. Si la condición es verdadera, el bloque de código indentado debajo del if se ejecuta; de lo contrario, se ignora.

2. elif: Se utiliza para verificar condiciones adicionales después de la primera condición if. Si la condición if es falsa, se evalúa la condición elif. Si la condición elif es verdadera, se ejecuta su bloque de código.

3. else: Se utiliza para ejecutar un bloque de código cuando ninguna de las condiciones anteriores es verdadera.

Ejemplo:

```
age = 55
if age < 25:
  print(f"I'm sorry, {age} is under 25 years old")
elif age > 100: #else if
  print(f"I'm sorry, {age} is over 100 years old")
else:
  print(f"You're good to go, {age} fits in the range to rent a car")
```

En el ejemplo, si la edad es inferior a 25, se cumple el primer condicional. En caso de ser la edad superior a 100 se cumple el segundo condicional y en los demás cases se cumple el tercero.

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

En Python, existen dos tipos principales de bucles: el bucle for y el bucle while.

- ❖ Bucle for:

El bucle for se utiliza para iterar sobre una secuencia (como una lista, tupla, cadena de caracteres, diccionario, etc.) y ejecutar un bloque de código para cada elemento de esa secuencia. Aquí hay un ejemplo:

```
for num in range(1, 6):
    print(num)
```

En este caso, el bucle for recorre el rango de números de 1 hasta 6 y ejecuta el bloque de código dentro del bucle para cada elemento de la lista. La salida sería:

- 1
- 2
- 3
- 4
- 5

- ❖ Bucle while:

El bucle while se utiliza para repetir un bloque de código mientras una condición sea verdadera. Aquí tienes un ejemplo:

```
def guessing_game():
    while True:
        print('What is your guess?')
        guess = input() # Para conseguir el valor a traves de la consola
        if guess == '42':
            print('You correctly guessed it!')
            return False
        else:
            print(f"No, {guess} isn't the answer, please try again\n")
guessing_game()
```

Este bucle se repetirá mientras la condición sea verdadera. La condición se evalúa antes de cada iteración, y el bucle se ejecutará mientras esa condición no cambie a “false” por haber acertado el número.

Utilidad de los bucles:

1. Automatización de tareas repetitivas: Los bucles permiten ejecutar un bloque de código varias veces, lo que es esencial para realizar tareas repetitivas de manera eficiente.
2. Procesamiento de colecciones de datos: Los bucles for son particularmente útiles para recorrer elementos en listas, tuplas, diccionarios, etc.
3. Control de flujo basado en condiciones: Los bucles while permiten ejecutar un bloque de código mientras se cumple una condición específica, proporcionando un control de flujo más dinámico.

¿Qué es una comprensión de listas en python?

Una comprensión de listas en Python es una forma concisa y poderosa de crear listas. Proporciona una sintaxis más compacta para la creación de listas basadas en expresiones y condiciones. Esta característica permite reducir la cantidad de código necesario para construir listas, haciéndolo más legible y eficiente.

Ejemplo:

Sean estas lineas de programación:

```
num_list = range(1, 11)
cubed_nums = []
for num in num_list:
  cubed_nums.append(num ** 3)
print(cubed_nums) # [1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

Con la función “for” recorremos el rango de 1 a 11 generando una lista `cubed_nums` que contiene esos valores al cubo: `[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]`

Utilizando comprensión de listas en Python podemos hacer lo mismo de una forma más concisa:

```
cubed_nums = [num ** 3 for num in range(1,11)] # Esto hace lo mismo que 4 lineas del otro programa
print(cubed_nums) # [1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

La sintaxis general de una comprensión de listas es la siguiente:

nueva_lista = [expresion for elemento in iterable if condicion]

- expresion: La expresión que define los elementos de la nueva lista.
- elemento: La variable que toma cada valor del iterable (como una lista, tupla, rango, etc.).
- iterable: La secuencia de elementos sobre la cual se iterará.
- condicion (opcional): Una condición que filtra los elementos que cumplen con cierta condición.

Ejemplo con un condicional “if” (opciona):

Sean estas lineas de programación:

```
num_list = range(1, 11)
even_numbers = []
for num in num_list:
  if num % 2 == 0: # Si el número es par
    even_numbers.append(num)
```

Como en el caso anterior, podemos reemplazar todo lo anterior con la compresión de listas siguiente:

```
even_numbers = [num for num in range(1,11) if num % 2 == 0]
print(even_numbers) # [2, 4, 6, 8, 10]
```

En este segundo ejemplo recorremos un rango de valores de 1 hasta 11 y si cumple el condicional de ser par el valor es añadido a la lista `even_numbers` .

¿Qué es un argumento en Python?

En Python, un argumento se refiere a un valor que se pasa a una función cuando esta es llamada. Los argumentos son esenciales para permitir que las funciones realicen operaciones específicas utilizando datos proporcionados desde fuera de la función. Hay dos tipos principales de argumentos: los argumentos posicionales y los argumentos de palabra clave.

- ❖ Argumentos posicionales:

Los argumentos posicionales se pasan a una función en el orden en que están definidos en la firma de la función. La posición del argumento determina a qué parámetro de la función se asigna.

Ejemplo:

```
def suma(a, b):
    resultado = a + b
    return resultado
# Llamamos a la función con argumentos posicionales
resultado_suma = suma(3, 5)
print(resultado_suma)
```

En este ejemplo, 3 se asigna al parámetro a y 5 al parámetro b. La función realiza la suma y devuelve 8.

- ❖ Argumentos de palabra clave:

Los argumentos de palabra clave se pasan a una función utilizando el nombre del parámetro al que se quiere asignar un valor. Esto permite desordenar los argumentos y proporciona claridad en la llamada de la función.

Ejemplo:

```
def saludo(nombre, mensaje):
    return f"Hola {nombre}, {mensaje}"
# Llamada a la función con argumentos de palabra clave
saludo_personalizado = saludo(mensaje="¿Cómo estás?", nombre="Juan")
print(saludo_personalizado)
```

En este ejemplo, los argumentos se pasan utilizando los nombres de los parámetros, lo que facilita entender qué valor se asigna a qué parámetro.

Finalmente mencionar que puedes asignar valores predeterminados a los parámetros de una función, lo que significa que si el usuario no proporciona un valor para ese parámetro, se utilizará el valor predeterminado.

```
def greeting(name = 'Guest'): #si se llama a la función name=” “, nos da valor 'Guest'
  print(f'Hi {name}!')
```

¿Qué es una función de Python Lambda?

Una función lambda en Python es una función anónima y de una sola línea que se crea utilizando la palabra clave lambda. A diferencia de las funciones definidas con la palabra clave def, las funciones lambda son más concisas y se utilizan generalmente para operaciones simples o situaciones en las que solo se necesita una función temporal.

Las funciones lambda son útiles cuando necesitas definir una función rápida y simple sin tener que utilizar la estructura completa de una función definida con def. Sin embargo, es importante destacar que las funciones lambda deben utilizarse con moderación y para casos en los que la simplicidad y concisión sean prioritarias, ya que pueden volverse menos legibles cuando la lógica se vuelve más compleja.

La sintaxis general de una función lambda es la siguiente:

lambda argumentos: expresion

1. lambda: La palabra clave que indica que estás creando una función lambda.
2. argumentos: Los parámetros que la función lambda aceptará.
3. expresion: La operación que la función lambda realizará y devolverá.

Ejemplo 1: generamos un saludo almacenando valores en una variable con lambda.

```
full_name = lambda first, last: f'{first} {last}'  #con lambda almacenamos valores en full_name
def greeting(name):
  print(f'Hi there {name}')
greeting(full_name('Kristine', 'Hudgens'))
```

Ejemplo 2: sumamos 2 números com lambda.

```
sumar = lambda x, y: x + y
resultado = sumar(3, 5)
print(resultado)
```

¿Qué es un paquete pip?

Un paquete pip en Python se refiere a un conjunto de módulos o bibliotecas de código que se pueden instalar y utilizar para agregar funcionalidades específicas a tus programas. El término "pip" proviene de "Pip Installs Packages" (Pip Instala Paquetes), que es un sistema de gestión de paquetes para Python. Pip se utiliza para instalar, actualizar y gestionar paquetes de terceros que no forman parte de la biblioteca estándar de Python.

Supongamos que estás trabajando en un proyecto de Python que requiere realizar solicitudes HTTP. En lugar de escribir todo el código desde cero para manejar estas solicitudes, puedes aprovechar un paquete existente como requests.

Primeramente se instala en Python con la siguiente instrucción en el terminal de la consola:

pip install requests

A continuación vemos un ejemplo:

```
import requests
from bs4 import BeautifulSoup
rawtext = requests.get('http://www.dailysmarty.com/topics/python') # cogemos todo el contenido “bruto”
soup = BeautifulSoup(rawtext.text, 'html.parser') #Modificamos el contenido ordenandolo
links = soup.find_all('a') #cogemos todos los enlaces
for link in links: # Recorremos todos los enlaces
        print(link['href']) #imprime los enlaces con todo el texto que lo sobra
```

En este ejemplo, requests es un paquete que proporciona funciones y métodos predefinidos para realizar solicitudes HTTP de manera sencilla. Al instalarlo con pip, puedes agregar rápidamente esta funcionalidad a tu proyecto sin tener que implementar toda la lógica desde cero.

Los paquetes pip son esenciales para la eficiencia y la reutilización de código en el desarrollo de software en Python, ya que permiten aprovechar el trabajo de la comunidad y agregar fácilmente nuevas capacidades a tus proyectos.