Respuestas a las preguntas teóricas del 5 punto de control, por Daniel Duque

1. **¿Qué es un condicional en Python?**

Un **condicional** es una estructura de control que nos permite tomar decisiones en nuestros programas. Se utiliza para ejecutar diferentes bloques de código según si se cumple o no una condición específica. En Python, los condicionales se crean utilizando las siguientes palabras clave:

- if: Se utiliza para evaluar una condición y ejecutar un bloque de código si esa condición es verdadera.
- elif (opcional): Permite evaluar múltiples condiciones en orden. Si la primera condición no se cumple, se verifica la siguiente.
- else: Se ejecuta si ninguna de las condiciones anteriores es verdadera.

Ejemplo 1: Verificación de edad

Digamos que queremos verificar si una persona es mayor de edad. Utilizaremos un condicional para lograrlo:

```
edad = 18

if edad >= 18:

    print("Eres mayor de edad")

else:

    print("Eres menor de edad")
```

En este caso, si la variable edad es mayor o igual a 18, se imprimirá “Eres mayor de edad”; de lo contrario, se imprimirá “Eres menor de edad”.

Ejemplo 2: Evaluación de notas

Tenemos un sistema de calificaciones y queremos clasificar a los estudiantes en base a sus notas:

Python

```
nota = 75

if nota >= 90:
```

```
        print("Excelente")
elif nota >= 70:
        print("Aprobado")
else:
        print("Reprobado")
```

En este ejemplo, si la nota es mayor o igual a 90, se imprime “Excelente”; si está entre 70 y 89, se imprime “Aprobado”; de lo contrario, se imprime “Reprobado”.

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

En Python, tenemos dos tipos principales de bucles:

- **Bucle** for: Se utiliza para iterar sobre una secuencia (como una lista, tupla o rango). Es útil cuando sabemos cuántas veces queremos repetir una acción.

  Ejemplo:

  Python

  ```
  for i in range(5):
          print(i)
  ```

- **Bucle** while: Se ejecuta mientras se cumpla una condición. Es útil cuando no sabemos cuántas veces se repetirá una acción.

  Ejemplo:

  Python

  ```
  contador = 0
  while contador < 5:
          print(contador)
          contador += 1
  ```

Los bucles son fundamentales para automatizar tareas y procesar datos de manera eficiente.

3. ¿Qué es una lista por comprensión en Python?

Una **lista por comprensión** es una forma concisa de crear listas a partir de otros elementos iterables (como listas, tuplas o rangos). Se utiliza para aplicar una expresión a cada elemento y, opcionalmente, filtrar los elementos que cumplan cierta condición. Veamos un ejemplo más detallado:

```
numeros = [1, 2, 3, 4, 5]

duplicados = [n * 2 for n in numeros if n % 2 == 0]

print(duplicados)  # Resultado: [4, 8]
```

En este ejemplo, creamos una nueva lista duplicados que contiene el doble de los números pares de la lista original numeros.

4. ¿Qué es un argumento en Python?

Un **argumento** en Python es un valor que se pasa a una función cuando se la llama. Los argumentos pueden ser posicionales o de palabras clave (keyword arguments). Veamos más detalles:

- **Argumentos posicionales**: Se pasan por orden en que aparecen en la definición de la función. Por ejemplo:

  ```
  def saludar(nombre, edad):
      print(f"Hola, {nombre}. Tienes {edad} años.")

  saludar("Ana", 25)
  ```

- **Argumentos de palabras clave**: Se especifican por su nombre al llamar a la función. Por ejemplo:

```
saludar(nombre="Carlos", edad=30)
```

Los argumentos nos permiten personalizar la funcionalidad de las funciones y reutilizar código.

5. ¿Qué es una función Lambda en Python?

- Una función lambda es una **función anónima** que se define en una sola línea y puede contener **una sola expresión**.
- A diferencia de las funciones regulares definidas con def, las funciones lambda no tienen un nombre específico.
- Se utilizan para crear funciones pequeñas y rápidas en el momento, sin necesidad de asignarles un nombre.
- Son especialmente útiles en situaciones como **ordenamiento, filtrado y mapeo**.

Sintaxis de una función lambda:

- La sintaxis básica de una función lambda es: lambda argumentos: expresión.
- Los argumentos se separan por comas y la expresión es el resultado que se devuelve.
- No puedes darle un nombre a una función lambda, ya que son **anónimas** por definición.

Ejemplo de función lambda:

```
# Definición de una función lambda que duplica su argumento
doble = lambda x: x * 2

# Uso de la función lambda
resultado = doble(3)
print(resultado)  # Salida: 6
```

Ventajas de las funciones lambda:

- **Concisión**: Las funciones lambda son ideales cuando necesitas una función sencilla y rápida.

- **Menos líneas de código**: Puedes crear una función lambda en el mismo lugar donde la necesitas, evitando nombrar una función que solo usarás una vez.
- **Funciones de orden superior**: Se pueden usar como argumentos en funciones como map o filter.

Ejemplo práctico con map**:**

```
mi_lista = [1, 2, 3, 4, 5, 6]
lista_nueva = list(map(lambda x: x * 2, mi_lista))
print(lista_nueva)  # Salida: [2, 4, 6, 8, 10, 12]
```

Errores comunes al trabajar con expresiones lambda:

- **Sobreuso**: Si abusamos de las funciones lambda, nuestro código puede volverse más complejo y difícil de mantener.
- **Legibilidad**: Python enfatiza la legibilidad, por lo que en ocasiones es preferible usar funciones nombradas para mayor claridad.

En resumen, las funciones lambda son una herramienta valiosa para simplificar y agilizar el código en situaciones específicas

6. ¿Qué es un paquete pip?

- **pip** es un **sistema de gestión de paquetes** utilizado para instalar y administrar paquetes de software escritos en Python.
- Muchos paquetes se encuentran disponibles en el **Python** Para desinstalar un paquete: pip uninstall nombre-paquete.
- También puedes gestionar listas de paquetes y sus versiones mediante un archivo de requisitos: pip install -r requisitos.txt.

En resumen, **pip** es una herramienta esencial para extender las funcionalidades de Python y reutilizar código de manera eficiente.