5.1 Cómo se define una función en Python, explicar la forma de manejo de los argumentos: En Python, puedes definir una función utilizando la palabra clave def, seguida del nombre de la función y paréntesis que pueden contener los argumentos de la función. Aquí tienes un ejemplo de cómo definir una función en Python:

def saludar(nombre):

print("Hola,", nombre)

saludar("Juan") # Llamada a la función

n este ejemplo, hemos definido una función llamada saludar que acepta un argumento llamado nombre. Dentro de la función, imprimimos un saludo utilizando el nombre proporcionado. Luego, llamamos a la función pasando el valor "Juan" como argumento.

5.2 Definir cómo Python maneja el pasaje por valor y por referencia: En Python, el pasaje de argumentos a una función se realiza por asignación de referencia. Esto significa que cuando se pasa un objeto a una función, se pasa la referencia al objeto y cualquier modificación realizada dentro de la función afectará al objeto original. Sin embargo, el comportamiento puede variar según el tipo de objeto.

Pasaje por valor: En Python, los tipos de datos inmutables como enteros, cadenas y tuplas se pasan por valor. Esto significa que una copia del valor se pasa a la función y cualquier modificación dentro de la función no afectará al objeto original fuera de la función.

Pasaje por referencia: En Python, los tipos de datos mutables como listas, diccionarios y conjuntos se pasan por referencia. Esto significa que se pasa la referencia al objeto y cualquier modificación realizada dentro de la función afectará al objeto original fuera de la función.

5.3 Definir los argumentos "especiales" en Python: \*Args y \*\*Kwargs en Python: En Python, existen dos argumentos especiales que se utilizan para manejar un número variable de argumentos en una función:

\*Args: Se utiliza para pasar un número variable de argumentos posicionales a una función. Los argumentos se empaquetan en una tupla dentro de la función. Puedes utilizar cualquier nombre en lugar de "args", pero se recomienda seguir esta convención. Aquí tienes un ejemplo:

def imprimir\_argumentos(\*args):

for argumento in args:

print(argumento)

imprimir\_argumentos(1, 2, 3) # Llamada a la función con varios argumentos

\*\*Kwargs: Se utiliza para pasar un número variable de argumentos de palabra clave a una función. Los argumentos se empaquetan en un diccionario dentro de la función. Al igual que con \*args, puedes utilizar cualquier nombre en lugar de "kwargs". Aquí tienes un ejemplo:

def imprimir\_valores(\*\*kwargs):

for clave, valor in kwargs.items():

print(clave, ":", valor)

imprimir\_valores(nombre="Juan", edad=30) # Llamada a la función con argumentos de palabra clave

5.4 Dar ejemplos de \*Args y \*\*Kwargs en Python: Aquí tienes algunos ejemplos adicionales de cómo utilizar \*args y \*\*kwargs

def sumar\_numeros(\*args):

resultado = 0

for numero in args:

resultado += numero

return resultado

print(sumar\_numeros(1, 2, 3)) # Im

5.5 Qué es recursividad, qué tipos de problemas soluciona, dar ejemplos: La recursividad es un concepto en programación en el cual una función se llama a sí misma durante su ejecución. Es decir, una función se puede definir en términos de sí misma. La recursividad se utiliza para resolver problemas que pueden dividirse en subproblemas más pequeños y similares al problema original.

La recursividad es útil para resolver problemas que siguen una estructura de "divide y vencerás", donde se resuelve un caso base directamente y los casos más grandes se dividen en subproblemas más pequeños. Algunos ejemplos de problemas que se pueden resolver utilizando recursividad son:

Cálculo de factorial: El factorial de un número entero positivo n se define como la multiplicación de todos los números enteros desde 1 hasta n. Se puede calcular recursivamente utilizando la relación n! = n \* (n-1)!.

Cálculo de la serie de Fibonacci: La serie de Fibonacci es una secuencia numérica en la que cada número es la suma de los dos números anteriores. Se puede calcular recursivamente definiendo la relación fib(n) = fib(n-1) + fib(n-2) con casos base fib(0) = 0 y fib(1) = 1.

Búsqueda en árboles o estructuras de datos recursivas: Algunas estructuras de datos como los árboles se pueden recorrer de manera recursiva, visitando los nodos o elementos de manera ordenada.

La recursividad puede ser una solución elegante para ciertos problemas, pero es importante tener cuidado de definir correctamente los casos base y asegurarse de que la recursividad termine en algún punto para evitar bucles infinitos.

5.6 Qué es una función Lambda, qué permite o facilita, y la diferencia con otros lenguajes: Una función lambda en Python es una función anónima que se define en una sola línea de código sin necesidad de utilizar la palabra clave def. Son funciones pequeñas y simples que se utilizan cuando se necesita una función temporal y no es necesario definirla formalmente.

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

Algunas características de las funciones lambda son:

Son funciones anónimas: No se les asigna un nombre, por lo que no se pueden llamar directamente después de ser definidas. Se utilizan principalmente como argumentos de otras funciones o en expresiones donde se requiere una función breve.

Son funciones de una sola expresión: La expresión después de los dos puntos (:) es el cuerpo de la función y es el valor de retorno.

Permiten la creación rápida de funciones: En lugar de definir una función completa utilizando def, se pueden crear funciones lambda de forma concisa y directa.

La principal diferencia entre las funciones lambda en Python y las funciones en otros lenguajes de programación es la simplicidad y la capacidad de expresar funciones cortas en una sola línea. Las funciones lambda son útiles en situaciones donde se necesita una función rápida y sencilla sin la necesidad de definir una función completa con un nombre.