1) . What is the difference between enclosing a list comprehension in square brackets and parentheses?

Enclosing a list comprehension in square brackets [] creates a list object, while enclosing it in parentheses () creates a generator object. A list comprehension eagerly evaluates and generates all the items in memory at once, while a generator comprehension lazily evaluates and generates items on the fly as they are needed. Generators are more memory-efficient for large datasets or infinite sequences as they generate items one at a time, whereas lists store all the items in memory.

2) What is the relationship between generators and iterators?

generators are a specific type of iterator that are created using generator functions. They allow you to define iterators in a more concise and efficient way compared to manually implementing the iterator protocol.

3) What are the signs that a function is a generator function?

There are a few signs that indicate a function is a generator function:

1. The presence of the yield keyword: Generator functions use the yield keyword to define the values to be generated. If you see yield statements within a function, it is a strong indication that it is a generator function.
2. Use of the return statement: Generator functions may contain return statements, but they are used to indicate the end of the generator and stop iteration. Unlike regular functions, where return terminates the function and returns a value, in generator functions, return raises a StopIteration exception to signal the end of the iteration.

4) What is the purpose of a yield statement?

The yield statement in Python is used in the context of generator functions to define points at which the function can "yield" a value.

The purpose of the yield statement is to create a generator object and define a sequence of values that can be iterated over. When a generator function encounters a yield statement, it temporarily suspends its execution and returns the yielded value to the caller. The generator function's state is saved, allowing it to resume execution later from where it left off.

5) What is the relationship between map calls and list comprehensions? Make a comparison and contrast between the two.

Both map calls and list comprehensions are used in Python for transforming and manipulating iterables, but they have some differences in syntax and usage:

1. Syntax:

Map call: The map function takes a function and one or more iterables as arguments and returns a map object. The syntax is map(function, iterable).

List comprehension: A list comprehension is an expression that generates a new list by iterating over an iterable and applying an expression or condition. The syntax is [expression for item in iterable if condition].

1. Output:

Map call: The map function returns a map object, which is an iterator that generates values lazily. To obtain the results as a list, you need to convert the map object to a list using the list() function.

List comprehension: A list comprehension directly generates a new list as its output.