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

ANS :- Enclosing a list comprehension in square brackets (`[]`) creates and returns a new list containing the generated elements. On the other hand, enclosing it in parentheses (`()`) creates and returns a generator object, which dynamically generates the elements as needed. Lists are immediately computed and stored in memory, while generators produce values on-the-fly, making them memory-efficient for large datasets. The choice between square brackets and parentheses depends on whether you need an immediate list or a generator for lazy evaluation.

2) What is the relationship between generators and iterators?

ANS :- Generators and iterators are like special containers in Python that allow you to get values one at a time instead of all at once.

A generator is a way to create an iterator using a special kind of function or expression. It lets you generate values on-the-fly as you need them, without needing to store all the values in memory at once. It's like a factory that produces values whenever you ask for them.

An iterator is an object that holds a sequence of values and gives you access to those values one at a time. It has a special method called `\_\_next\_\_()` that provides the next value in the sequence when you call it. You can keep getting values from the iterator until there are no more values left.

So, generators are a specific type of iterator that allows you to create sequences of values dynamically, while iterators are the objects that actually give you those values one by one.

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

ANS :- There are a few signs that indicate a function is a generator function: 1. The function uses the yield keyword instead of return to return values. yield allows the function to pause and resume its execution, generating a sequence of values. 2. Generator functions typically contain one or more yield statements within their body. These statements are used to yield values to the caller and temporarily suspend the function's execution. 3. When a generator function is called, it returns a generator object rather than immediately executing the function's body. The generator object is an iterator that can be iterated over to retrieve values one at a time. 4. Generator functions often have a loop or some other mechanism for generating values dynamically. They can generate an infinite sequence of values or generate values based on some condition or input.

4) What is the purpose of a yield statement?

ANS :- The purpose of a `yield` statement is to create a generator function that can generate a sequence of values lazily, one at a time. Instead of returning a single value and ending the function, `yield` allows the function to temporarily pause its execution, yield a value to the caller, and then resume from where it left off when called again. It is a way to produce a series of values over time without generating them all at once, which can be memory-efficient and allows for efficient processing of large or infinite sequences.

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

ANS :- Both `map` calls and list comprehensions are used to transform and process elements in an iterable, such as a list. However, there are some differences between them:

1. Syntax: List comprehensions have a more concise and readable syntax compared to `map` calls. List comprehensions use square brackets `[ ]` to enclose the expression and iteration, while `map` calls require the `map` function to be explicitly called.

2. Output: List comprehensions always return a new list as the result, containing the transformed elements. On the other hand, `map` calls return a map object by default, which is an iterable. To obtain the transformed elements as a list, the map object needs to be explicitly converted using the `list()` function.

3. Flexibility: List comprehensions offer more flexibility and support additional features. They can include conditions using the `if` clause, allowing for conditional filtering of elements. `map` calls, by themselves, cannot include conditional logic. However, `map` can be combined with `filter` to achieve similar functionality.

4. Readability: List comprehensions tend to be more readable and intuitive for simple transformations, especially when the logic involves a single expression. They provide a more declarative and concise way of expressing the transformation. `map` calls can be used for more complex transformations that require custom functions, but they may introduce more syntax and reduce readability.