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

Enclosing list comprehension in square bracket creates new list based on condition provided in it whereas enclosing list comprehensions in parenthesis produces generator expression. It is memory-efficient compared to creating a list, as it doesn't store all the elements in memory at once.

2) What is the relationship between generators and iterators?

Iterators in general are a broader concept. They include any object that follows the iterator protocol (__iter__() and __next__() methods), which includes not just generator functions but also custom classes that implement these methods. All generators are iterators, but not all iterators are generators. Generators are a specific type of iterator that is defined using generator functions or expressions.

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

Generator function in is identified by the presence of the yield keyword.

4) What is the purpose of a yield statement?

The yield statement in serves a crucial role in defining generator functions and enabling lazy evaluation. It allows a function to generate a series of values on-the-fly and temporarily pause its execution state, resuming from where it left off when the generator is iterated again. The primary purpose of the yield statement is to enable the creation of memory-efficient iterators and generators.

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

Similarities:

1. Both map() and list comprehensions allow you to apply a function to each element of an iterable and create a new iterable with the transformed values.
2. Both concepts are rooted in functional programming principles, where operations are applied to data without mutating the original data.

Differences:

1. map() requires a function and one or more iterables. It applies the function to the corresponding elements of the iterables. List Comprehensions: Uses a concise syntax to define the transformation along with the loop structure.
2. map() can use built-in functions, lambda functions, or any callable as the transformation function. List Comprehensions can include any expression that computes the transformed value for each element.
3. map() always returns a map object, which is an iterator. To get a list, you need to convert it explicitly using list(map(...)). List Comprehensions always generate a list directly.