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. It constructs the entire list in memory and returns it as a whole.

- Enclosing a list comprehension in parentheses `( )` creates a generator object. It generates elements on-the-fly as they are needed, rather than creating the whole list in memory.

2) What is the relationship between generators and iterators?

- Generators are a type of iterator. They are created using a special kind of function called a generator function. Generator functions use the `yield` statement instead of `return` to generate a sequence of values. When a generator function is called, it returns a generator object.

- Iterators, on the other hand, are objects that implement the iterator protocol, which consists of the `__iter__()` and `__next__()` methods. They allow iteration over a sequence of elements. Generators implement the iterator protocol and can be used as iterators.

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

- A generator function in Python is identified by the presence of the `yield` keyword. When a function contains a `yield` statement, it becomes a generator function. The `yield` statement is used to produce a value in the sequence generated by the function.

4) What is the purpose of a yield statement?

- The `yield` statement in Python is used in generator functions to produce a value in the sequence generated by the function. It allows the generator function to "yield" a value each time it is called and resume from where it left off when called again. The `yield` statement pauses the execution of the function and saves its internal state.

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 to apply a transformation to a sequence of elements and generate a new sequence.

- `map` is a built-in function in Python that takes a function and an iterable as input and applies the function to each element of the iterable, returning an iterator that yields the transformed values.

- List comprehensions, on the other hand, are a concise way to create lists based on existing sequences. They provide a compact syntax for applying an expression or transformation to each element of an iterable and generating a new list.

- While both `map` and list comprehensions can achieve similar results, list comprehensions are often considered more readable and expressive for simple transformations. `map` can be useful when working with complex or multiple arguments functions or when dealing with functions that are not easily expressed in a simple expression format.