Q1. What is the difference between enclosing a list comprehension in square brackets and
Parentheses?
Square Brackets ([]): When you use square brackets to enclose a list comprehension, it creates a new list. The list comprehension evaluates the expression and generates a list of elements based on the specified conditions.
Parentheses (()): When you use parentheses to enclose a list comprehension, it creates a generator object. A generator is an iterator that produces values on-the-fly, and it does not store all the values in memory at once like a list does. Instead, it generates each value when needed, which can be more memory-efficient for large datasets.

Q2. What is the relationship between generators and iterators?
All generators are iterators, but not all iterators are generators. Generators are a specific type of iterator that is implemented using a special syntax (generator functions with yield statements). The generator protocol allows you to create iterators in a more concise and readable way compared to defining a separate iterator class.
The primary benefit of using generators is that they provide a convenient and memory-efficient way to generate large sequences of data on-the-fly, without having to store all the elements in memory at once. They can be used in situations where you need to work with large datasets, infinite sequences, or data streams.
In summary, the relationship between generators and iterators is that generators are a specific type of iterator, and they provide a more elegant and efficient way to implement iterators in Python.

Q3. What are the signs that a function is a generator function?
Presence of yield keyword: The most evident sign of a generator function is the use of the yield keyword inside the function body. If a function contains at least one yield statement, it is a generator function.

Use of function call doesn't execute the function: When you call a generator function, it does not execute the function body immediately. Instead, it returns a generator object, which acts as an iterator. You need to iterate over this generator object to execute the function and retrieve values from the yield statements.

Q4. What is the purpose of a yield statement?
Here's how the yield statement works and its purpose:
Produces a value and pauses execution: When a yield statement is encountered in a generator function, it immediately produces the value specified after the yield keyword and suspends the function's execution. The current state of the function, including the values of local variables and the program counter, is saved.

Returns a generator object: When you call a generator function, it does not execute the function body right away. Instead, it returns a generator object, which is an iterator that can be used to traverse the sequence of values generated by the function.

Resumes execution upon iteration: When you iterate over the generator object (using a for loop or the next() function), the generator function resumes execution from where it was paused, just after the yield statement. The function continues executing until it encounters the next yield statement or reaches the end of the function body.

**Q5. What is the relationship between map calls and list comprehensions? Make a comparison and
contrast between the two.**

Feature	Map Calls	List Comprehensions
Syntax	`map(function, iterable)`	`new_list = [expression for item in iterable if condition]`
Semantics	**Applies a function to each element in a sequence**	**Creates a new list from a sequence**
Memory Usage	**Creates a list in memory**	**Does not create a list in memory (unless explicitly called)**
Speed	**Slower**	**Faster**
Usability	**Less user-friendly**	**More user-friendly**