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

A1.In Python, enclosing a list comprehension in square brackets [] creates a list, while enclosing it in parentheses () creates a generator object.

A list comprehension is a way to create a new list by applying an operation to each item in an existing iterable object. Using square brackets around the comprehension creates a new list containing the results of the operation for each item.

On the other hand, a generator expression is a way to lazily evaluate the items in a sequence. It only generates values as needed, and can be more memory-efficient than creating a full list. Using parentheses around the comprehension creates a generator object

2) What is the relationship between generators and iterators?

A2. Generators and iterators are closely related concepts in Python. In fact, generators are a type of iterator.

An iterator is an object that can be iterated (looped) upon, meaning that we can traverse through all the values. It represents a stream of data and allows us to process that data one item at a time, rather than loading it all into memory at once. In Python, an iterator is an object that implements the iterator protocol, which requires it to have a **\_\_next\_\_()** method that returns the next item in the sequence and raises a **StopIteration** exception when there are no more items.

A generator is a special type of iterator that is defined using a function rather than a class. A generator function is a function that uses the **yield** keyword to return a generator object. When we call a generator function, it returns a generator object, which we can use to iterate over the values it generates. The main advantage of using a generator function over an iterator class is that it is simpler and more concise.

In summary, a generator is a specific type of iterator that is implemented using a function with the **yield** keyword.

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

A3.   
There are two main signs that a function is a generator function:

1. The function contains a **yield** statement: The **yield** statement is what makes a function a generator function, as it allows the function to produce a series of values that can be iterated over one at a time. When a function with a **yield** statement is called, it returns a generator object, rather than executing the function's code immediately.
2. The function uses the **yield** statement in a loop: Most generator functions will use the **yield** statement within a loop, so that they can produce a sequence of values. In this case, the **yield** statement will usually be combined with some kind of logic that generates the values to be yielded.

4) What is the purpose of a yield statement?

A4. The yield statement is used in the body of a function like a return statement, but instead of terminating the function and returning a value, it temporarily suspends the function's execution and sends a value back to the caller.

When a generator function encounters a **yield** statement, it produces a generator object, which is a special type of iterator that can be used to iterate over a series of values produced by the generator function. Each time the **next()** method is called on the generator object, the function resumes execution from where it left off and continues until it encounters the next **yield** statement, at which point it sends the next value back to the caller.

The purpose of the **yield** statement is to allow a function to produce a series of values lazily, one at a time, instead of generating all the values at once and returning them as a list or other sequence object. This can be more memory-efficient, especially when dealing with large data sets, because it allows the caller to consume the values as needed, without having to store them all in memory at once.

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

A5. Both map and list comprehensions are used to apply a function to every element of an iterable object (such as a list) and return the result as a new iterable object. The difference between them lies in their syntax and the type of object they return.

**map** is a built-in Python function that takes a function and an iterable object as its arguments, applies the function to each element of the iterable, and returns a map object.

List comprehensions, on the other hand, are a syntactic construct in Python that allow us to create a new list by specifying the elements we want to include and the conditions they must meet.