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

**-** The difference between enclosing a list comprehension in square brackets (`[]`) and parentheses (`()`) is:

Square brackets (`[]`) create a list comprehension and produce a list as the result.

Parentheses (`()`) create a generator expression and produce a generator object as the result**.**

1. **What is the relationship between generators and iterators?**

- Generators are a specific type of iterator in Python. Both generators and iterators allow you to iterate over a sequence of values, but there are some key differences:

- Generators are a more specific and convenient way to create iterators.

- Generators can be defined using functions with the `yield` keyword or using generator expressions.

- Generators yield values one at a time, allowing lazy evaluation and efficient memory usage.

- Iterators, on the other hand, are more general and can be implemented using classes with `\_\_iter\_\_` and `\_\_next\_\_` methods.

- All generators are iterators, but not all iterators are generators.

1. **What are the signs that a function is a generator function?**

**-** The signs that a function is a generator function are:

1. It contains the `yield` keyword within its definition.

2. It does not have a `return` statement or has a `return` statement without a value.

The presence of `yield` and the absence of a return value distinguish a generator function from a regular function. When a generator function is called, it returns a generator object that can be iterated over to produce values one at a time using the `yield` statements within the function.

1. **What is the purpose of a yield statement?**

- The purpose of a `yield` statement in a function is to indicate that the function is a generator function and to yield a value to the caller without terminating the function's execution. When the generator function is iterated, it produces values one at a time and temporarily suspends its execution state, allowing it to resume where it left off when the next value is requested. This lazy evaluation enables efficient memory usage and processing of large datasets.

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

**-** The relationship between `map` calls and list comprehensions is that both are used to transform and process iterable data, but they have differences in terms of syntax and usage:

Map Calls:

- `map` is a built-in function in Python that applies a specified function to each item in one or more iterables.

- It returns a map object (an iterator) by default, so you often need to convert it to a list or another iterable type to access the results.

- It requires defining a separate function or using a lambda function to specify the transformation.

- `map` is typically used when you want to apply the same function to all elements of one or more iterables and collect the results.

List Comprehensions:

- List comprehensions are a concise and Pythonic way to create lists by applying an expression to each item in an iterable and optionally filtering the items based on a condition.

- They return a list directly, making the result readily accessible without the need for further conversion.

- List comprehensions allow you to specify the transformation and filtering in a single, readable line of code.

- They are used when you want to create a new list with transformed or filtered values from an existing iterable.