**ANSWERS**

**Q1. What is the difference between enclosing a list comprehension in square brackets and parentheses?**

**Ans.** Difference between enclosing a list of comprehension in square brackets and parenthesis is as follows:

* When we use square brackets ([ ]), we create a list comprehension. It generates a new list by applying an expression to each item in the iterable.
* When we use parentheses (( )), we create a generator expression. It generates an iterator that produces values on-the-fly as you iterate through it. This is memory-efficient compared to creating a full list in memory.

**Q2. What is the relationship between generators and iterators?**

**Ans.** Relationship between generators and iterators is as follows:

* An iterator is an object that implements the methods \_\_iter\_\_() and \_\_next\_\_() to provide a sequence of values, one at a time.
* A generator is a specific type of iterator that is defined using a special syntax involving functions and the yield keyword. Generators allow you to create iterators in a more concise and readable way, without needing to manually implement the iterator protocol.

**Q3. What are the signs that a function is a generator function?**

**Ans.** Signs that a function is a generator function are as follows:

* It contains one or more yield statements.
* It might have a regular return statement, but it's not required.
* When called, it doesn't execute the function body immediately but returns a generator object instead.

**Q4. What is the purpose of a yield statement?**

**Ans.** Purpose of a yield statement is as follows:

* The yield statement is used in generator functions to yield a value to the caller, while maintaining the function's state so it can continue execution later.
* When the generator function is called, it doesn't run the entire function body at once; instead, it returns a generator object. The function is only executed when the generator's \_\_next\_\_() method is called.
* Each time the yield statement is encountered, the function's state is saved, and the yielded value is returned to the caller. The state can be restored when the function continues executing after the yield statement.

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

**Ans.** It is as follows:

**Relationship:**

* + **Map:** The map() function applies a given function to each item of an iterable (or multiple iterables) and returns an iterator that produces the results.
  + **List Comprehension:** A list comprehension creates a new list by applying an expression to each item in an iterable and collecting the results in a single list.

**Comparison:**

* + Both map and list comprehensions are used to transform data based on a given operation.
  + map tends to be more concise and memory-efficient for simple operations on large datasets.
  + List comprehensions are more versatile as they can include conditionals and nested operations, making them more powerful for complex transformations.

**Contrast:**

* + List comprehensions are more expressive and allow for complex conditions and nested operations.
  + map is particularly useful when you want to apply a single function to multiple iterables in parallel.
  + List comprehensions create a new list immediately, while map returns an iterator, which can be memory-efficient for large datasets.