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

Answer =

Enclosing a list comprehension in square brackets [...] creates a list, while enclosing it in parentheses (...) creates a generator expression. A list comprehension creates a new list in memory and holds the entire result set in memory, which can use a lot of memory for large inputs. On the other hand, a generator expression generates the values one-by-one on the fly and doesn't store the result set in memory, which is more memory-efficient.

2) What is the relationship between generators and iterators?

Answer =

Iterators of a certain kind are generators. An object that implements both the \_\_iter\_\_ and \_\_next\_\_ methods, which return iterator objects and the subsequent item in the series, is known as an iterator.

It may build a generator by using a generator function or a generator expression. A generator is a sort of iterator. A generator function is created similarly to a regular function, but rather of using return to return a value, it utilises yield to produce the next element in the sequence each time it is called. Similar to a list comprehension, a generator expression returns a generator object rather than a list and is contained in parentheses as opposed to square brackets.

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

Answer =

If a function contains at least one yield statement (it may contain other yield or return statements), it becomes a generator function. Both yield and return will return some value from a function.

4) What is the purpose of a yield statement?

Answer =

A yield statement looks much like a return statement, except that instead of stopping execution of the function and returning, yield instead provides a value to the code looping over the generator and pauses execution of the generator function

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

Answer =

**Map function**

Suppose we have a function and we want to compute this function for different values in a single line of code . This is where map() function plays its role. map() function returns a map object(which is an iterator) of the results after applying the given function to each item of a given iterable (list, tuple etc.)

**List Comprehension**

List Comprehension is a substitute for the lambda function, map(), filter() and reduce(). It follows the form of the mathematical set-builder notation. It provide a concise way to create lists.

Map VS List Comprehension

|  |  |
| --- | --- |
| Map Comprehension | List Comprehension |
| Map comprehension is less concise and easier to read | List comprehension is more concise and easier to read |
| In map, we have no such facility. | List comprehension allows filtering. |
| map only returns a map object and does not return any list. | List comprehension are used when a list of results is required |
| Map is slow compare to list | List comprehension is faster than map when we need to evaluate expressions that are too long or complicated to express |
| Map is faster in case of calling an already defined function | In this case list comprehension is slow |