ASSIGNMENT 25

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

ANSWER 1 :

Two important differences are enumerated below:

1. Enclosing a list comprehension in parentheses `()` will form a **generator** and so the memory consumption is much lower in comparison to list comprehension with square brackets especially dealing with big lists.
2. Generators not only consume significantly less memory but also will take much less time because they don't need to pre-build objects in a list.

QUESTION 2 : What is the relationship between generators and iterators?

ANSWER 2 :

- Classes are used to Implement the iterators, iterators are the objects that use the `next()` method to get the next value of the sequence.
- Functions are used to implement the generator, a generator is a function that yields a sequence of values using a `yield` statement.

Also, generator in python is a subclass of Iterator. This can easily be checked using the following code.

>>> import collections, types
>>> issubclass(types.GeneratorType, collections.Iterator)
>>>True

QUESTION 3 : What are the signs that a function is a generator function?

ANSWER 3:

Generator functions are those functions that, instead of returning a single value, return an iterable generator object. Such a function has at-least one **`yield`** keyword in it.

QUESTION 4 : What is the purpose of a yield statement in Python?

ANSWER 4 :

When we are working with a larger data, for example, if we want to print one million numbers in fibonacci series and we use return statement inside the function then such a function will take up a lot of computer's resources and will give a result only when all the one million numbers are computed but if we choose to use yield statement then it will make the function a generator and much less resources will be utilized by the computer, yield statement does it by suspending function's execution and sending a value back to the caller, without destroying the states of its local variables to enable function to resume where it is left off.

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

ANSWER 5:

Python Map function and Python list comprehension are features that work differently but have some similarities. Their performance varies with the parameters that are being used in them.

If we make a comparison between the two we find that,

1. The map function takes an Expression and an Iterable. The output will be an Iterable object where the expression will work on each element of the given Iterable. The output of each expression will be an element of the resultant Iterable.
   Syntax
   ```
   map( expression, iterable)
   ```
   Parameters
   It accepts two parameters as arguments: "expression "and "iterable".
   - Expression is the formula you need to calculate the result.
   - Iterable is whose each element will be calculated from the expression.

2. Python List Comprehension is used for creating a list where each element is generated by applying a simple formula on the given list.
   Syntax
   ```
   resultant_list = [ <variable_expression> for <variable>
   in <input_list> ]
                  or
   resultant_list = [ <variable_expression> for <variable>
   in <input_list> if <condition> ]
   ```

If we make a contrast between the two, i.e., **Map Vs List Comprehension**

1. List comprehension has a simpler configuration than the map function.
2. List comprehension can be used together with if condition as replacement of filter method. Map function has no such functionality. However, we can feed the map function output to the filter function.
3. List comprehension returns a list, whereas the map function returns an object of Iterable.
4. List comprehension execution is faster than that of map function when the formula expression is huge and complex.
5. Map function is faster than list comprehension when the formula is already defined as a function earlier. So, that map function is used without lambda expression.