1. What's the difference between using square brackets and parentheses to enclose a list comprehension?

Ans-Square brackets are lists while parentheses are **tuples**

The difference between the two kinds of expressions is that the List comprehension **is enclosed in square brackets []** while the Generator expression is enclosed in plain parentheses ().

1. What is the relationship between generators and iterators?

Ans:- Iterators: Iterator are objects which uses next() **method to get next value of sequence**.

Generators: A generator is a function that produces or yields a sequence of values using yield method.

A generator in python makes use of the 'yield' keyword. A python iterator doesn't. Python generator saves the states of the local variables every time 'yield' pauses the loop in python. An iterator does not make use of local variables, **all it needs is iterable to iterate on**.

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

Ans:- **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

A generator is a special type of function which does not return a single value, instead, it returns **an iterator object with a sequence of values**. In a generator function, a yield statement is used rather than a return statement.

1. What is the purpose of a yield statement?

Ans:- The yield statement **suspends function's execution and sends a value back to the caller, but retains enough state to enable function to resume where it is left off**. When resumed, the function continues execution immediately after the last yield run.

In its simplest form, 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**.

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

Ans:- List comprehension is **more concise and easier to read as compared to map**. List comprehension are used when a list of results is required as map only returns a map object and does not return any list. Map is faster in case of calling an already defined function (as no lambda is required).

**Syntax**

Map comprehension

map( expression, iterable)

List comprehension

**Syntax**

resultant\_list = [ <variable\_expression> for <variable> in <input\_list> ]

              or

resultant\_list = [ <variable\_expression> for <variable> in <input\_list> if <condition> ]

* List comprehension has a simpler configuration than the map function.
* 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.
* List comprehension returns a list, whereas the map function returns an object of Iterable.
* List comprehension execution is faster than that of map function when the formula expression is huge and complex.
* 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.

**Comparing Execution Time**

Now to examine the execution performance of list comprehension and map function, we will import a module "timeit" to check the execution time. Let us see the result with a variety of methods.

Without lambda: Map is faster than List Comprehension when function is already defined in case of map function.

**Example code 1: This code will print the time taken to evaluate numbers from 1 to 50. Map function is used without lambda.**

import timeit

# list comprehension

l1 = timeit.timeit( '[ l for l in range(50)]' , number = 999999)

print (l)

#map function

f= 'def num( ) : print (n)'

m1 = timeit.timeit( ' map (num, range(50))' , number = 999999, setup = f )

print (m)

**With lambda in map:** List comprehension is better than map function when we don't define the function beforehand and use lambda expression inside map.

**Example code 2:** This code will print the time taken to add a number to itself and this is applied for each element of the list. The expression is defined as lambda expression.

import timeit

# list comprehension

l2 = timeit.timeit( '[ n+n for n in range(50)]' , number = 999999)

print (l)

#map function

m2 = timeit.timeit( ' map (lambda a: a+a, range(50))' , number = 999999, setup = f )

print (m)

 You will see that for the first case, m1 is very less than l1. That means the map works faster than list comprehension. For the second case, m2 is greater than l2, implying that list comprehension is faster than map function when map function is used with Lambda expression.

**Conclusion**

There are no clear answers about which is the better option, in Python Map Vs List comprehension. We should know what the problem statement is, then use the method which suits us better because our main objective is to calculate the solution with optimal time and space consumption.