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

List Comprehension with square brackets produces list.

**Code:**

lst = [i for i in range(10)]

lst

List Comprehension with parentheses creates generators

**Code:**

ls = (i for i in range(10))

ls

list(ls)

2) What is the relationship between generators and iterators?

***Iterator***: An iterator is an object that can be iterated upon. Thus, iterators contain a countable number of values. All iterators are also iterable. However, every iterable is not necessarily an iterator. An iterable produces an iterator only once it is iterated on. List, which is an iterable, and produce an iterator by calling the iter().

**Code**:

list\_instance = [1, 2, 3, 4]

print(iter(list\_instance))

Although the list by itself is not an iterator, calling the iter() function converts it to an iterator and returns the iterator object. next() keyword is used to call the next element in the iterable object.

***Generator***: A special type of function which does not return a single value: it returns an iterator object with a sequence of values. A generator object does not return items. Instead, it uses the yield keyword to generate items on the fly. By using next() function we can iterate the output of generator function.

**Code**:

def factors(n):

for val in range(1, n+1):

if n % val == 0:

yield val

print(factors(20))

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

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?

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.

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 is a substitute for the lambda function, map(), filter() and reduce().

Map VS List Comprehension:

* List comprehension is more concise and easier to read as compared to map.
* List comprehension allows filtering. In map, we have no such facility. For example, to print all even numbers in range of 100, we can write [n for n in range(100) if n%2 == 0]. There is no alternate for it in 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.
* 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 (as no lambda is required).