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

**Ans:**

Enclosing a list comprehension in square brackets returns a list whereas enclosing a list comprehension in parentheses returns a generator object.

I/P:

l **=** [ele **for** ele **in** range(10)]

print(l, type(l))

g **=** (ele **for** ele **in** range(10))

print(g, type(g))

O/P:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9] <class 'list'>

<generator object <genexpr> at 0x00000193F09C5C10> <class 'generator'>

**2) What is the relationship between generators and iterators?**

**Ans:**

As we are aware of objects, iterator is an object and tuples, list, sets and other these are iterable object, and iterator iterates on countable number of values. Iterators are implemented using a class. It follows lazy evaluation, this helps us in avoiding evaluation repeatedly, as it executes only when specifically, it is called, as the expression is on hold and it is stored in memory. The memory location which I needed for processing the value is 1 and it helps us if we are using a very large dataset, as we will be able to reduce the memory wastage. And also added advantage is we don’t need to load the entire dataset at the same time will not be there. For an iterator: iter() keyword is used to create an iterator containing an iterable object.next() keyword is used to call the next element in the iterable object.

Similarly, Generators are another way of creating iterators in a simple way where it uses the keyword yield statement instead of return statement in a defined function. Generators are implemented using a function. Similar to iterator, lazy evaluation is followed by generators. As we know that the yield function returns the data as it is it don’t exit the function and it leaves the data unaffected. As Yield function, the entire sequence of data is not stored in the memory but it returns the data in a sequential format and as it is in iterable format in order to use the data we need to iterate over the data.

I/P:

*# Example of iterartor*

iter\_str **=** iter(['iNeuron','Full','Stack','Data Science'])

print(type(iter\_str))

print(next(iter\_str))

print(next(iter\_str))

print(next(iter\_str))

print(next(iter\_str))

print(iter\_str) *# After the iterable object is completed, to use them again we have reassign them to the same object.*

*# Example of Generator*

**def** cube\_numbers(in\_num):

**for** ele **in** range(in\_num**+**1):

**yield** ele**\*\***3

out\_num **=** cube\_numbers(4)

print(next(out\_num))

print(next(out\_num))

print(next(out\_num))

print(next(out\_num))

print(next(out\_num))

O/P:

<class 'list\_iterator'>

iNeuron

Full

Stack

Data Science

<list\_iterator object at 0x00000193F0A88BE0>

0

1

8

27

64

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

**Ans:**

A generator function uses a **yield** statement instead of a **return** statement. A generator function will always return an iterable object called generator whereas a normal function can return a **string/list/tuple/dict/NoneType** ... etc.

**4) 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. This allows its code to produce a series of values over time, rather than computing them at once and sending them back like a list.

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

**Ans:**

The main differences between map calls and list comprehensions are:

1. The readability of a List comprehension compared to a map is easy and concise
2. List comprehension allows filtering. In map, we have no such facility. For example, to print all odd numbers in range of 50, we can write **[n for n in range(50) if n%2 != 0].** In map, no alternative is present for this.
3. List comprehension are used when a list of results is required as final output but map only returns a map object. It needs to be explicitly converted to desired data type.
4. Map is slower than a List comprehension when the expression which we need to evaluate are complicated and lengthy
5. Map is faster in case of calling an already defined function on a set of values.