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

Enclosing a list comprehension in square brackets returns a list.but where as enclosing a list comprehension in parentheses returns a generator object

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

print(l, type(l))

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

print(g, type(g))

**Output:**

[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?**

An iterator is an object which contains a countable number of values and it is used to iterate over iterable objects like list, tuples etc Using 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.

Generator It is another way of creating iterators in a simple way where it uses the keyword “yield” instead of returning it in a defined function Generators are implemented using a function Here, the yield function returns the data without affecting or exiting the function. It will return a sequence of data in an iterable format where we need to iterate over the sequence to use the data as they won’t store the entire sequence in the memory By using next() function we can iterate the output of generator function

*# 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))

**Output:**

<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?**

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

**4) What is the purpose of a yield statement?**

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.**

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.)

If we already have a function defined, it is often good to use map For example, map(sum, myLists) is more neat than [sum(x) for x in myLists]. You gain the elegance of not having to make up a dummy variable (e.g. sum(x) for x... or sum(*) for*... or sum(readableName) for readableName...) which you have to type twice, just to iterate.

List Comprehension: List Comprehension is a substitute for the lambda function, map(), filter() and reduce()

The main differences between map calls and list comprehensions are:

1. List comprehension is more concise and easier to read as compared to map.
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]. There is no alternate for it in map
3. List comprehension are used when a list of results is required, whereas map only returns a map object and does not return any list.
4. List comprehension is faster than map when we need to evaluate expressions that are too long or complicated to express
5. Map is faster in case of calling an already defined function on a set of values.