Assignment 25 Solutions

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 but enclosing a list comprehension in parentheses returns a `generator` object. Example is mentioned below:

In [1]:

```
l = [ele for ele in range(10)]
print(l, type(l))
g = (ele for ele in range(10))
print(g, type(g))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9] <class 'list'>
<generator object <genexpr> at 0x0000027ACF4FF740> <class 'generator'>
```

2) What is the relationship between generators and iterators?

ANS: An `iterator` is an object which contains a countable number of values and it is used to iterate over iterable objects like list, tuples, sets, etc. Iterators are implemented using a class. It follows lazy evaluation where the evaluation of the expression will be on hold and stored in the memory until the item is called specifically which helps us to avoid repeated evaluation. As lazy evaluation is implemented, it requires only 1 memory location to process the value and when we are using a large dataset then, wastage of RAM space will be reduced the 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 an 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. Just as iterators, generators also follow `lazy evaluation` . 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.

In [2]:

```python
# Example of iterator
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 rec

# 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))
```

```
<class 'list_iterator'>
iNeuron
Full
Stack
Data Science
<list_iterator object at 0x0000027ACF599610>
0
1
8
27
64
```

3) 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.

4) What is the purpose of a yield statement?

ANS: 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.

ANS: `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, ma
p(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 it
erate.

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

Comparision:

```
    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 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
    3. List comprehension are used when a list of results is required, where as ma
p 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 (as no lambda
 is required)
```