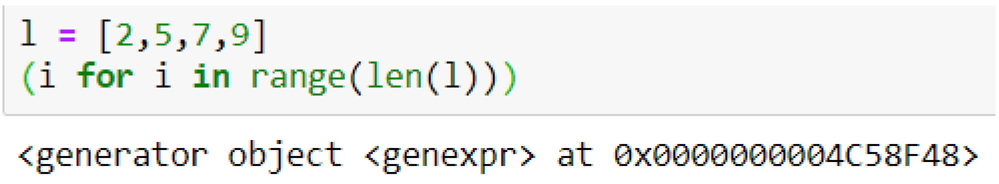
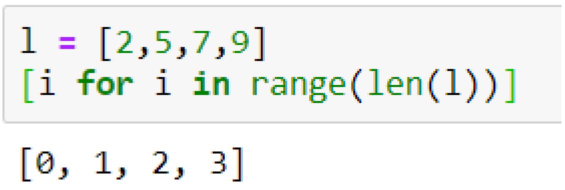
1) . What is the difference between enclosing a list comprehension in square brackets and parentheses?

Ans: List comprehension in square bracket return a list. Where as list comprehension in parentheses return a generator object.



```
l = [2,5,7,9]
(i for i in range(len(l)))
```

```
<generator object <genexpr> at 0x0000000004C58F48>
```



```
l = [2,5,7,9]
[i for i in range(len(l))]
```

```
[0, 1, 2, 3]
```

2) What is the relationship between generators and iterators?

Ans:

Iterator

Suppose we are creating a list, l = [4,6,7,9]. This list is iterable using the below code:

```
l = [2,5,7,9]

for i in l:

   print(i, end = ' ')
```

Now if I want to create iterator then we have to call function iter(), which will create an iterator on iterable objects.

l1 = iter(l)

Now if we write next(l1) then it will provide 2

Again next(l1) will provide 5

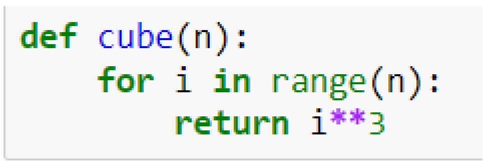
Again next(l1) will provide 7

Again next(l1) will provide 9

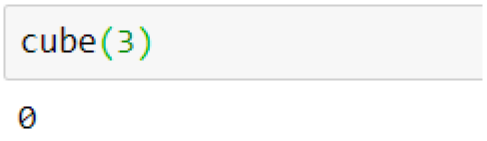
Again next(l1) will provide exception “Stop iteration”.

Generator

We can create a iterator using generator technique



```python
def cube(n):
    for i in range(n):
        return i**3
```

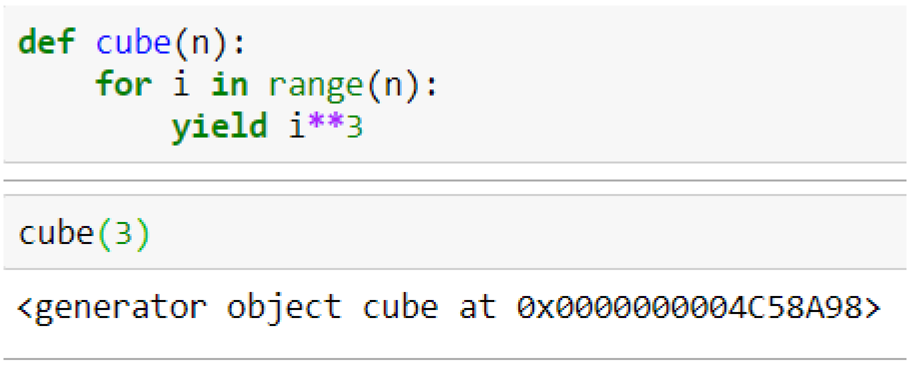


```python
cube(3)
```

```
0
```

Above function is used to return cube of range between 0 to n. But for the first value i = 0 it will just return 0.

But instead of writing return if we write keyword yield then it will return a generator object as shown below



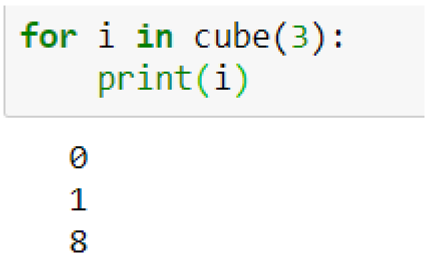
```python
def cube(n):
    for i in range(n):
        yield i**3
```

```python
cube(3)
```

```
<generator object cube at 0x0000000004C58A98>
```

Here generator is used to create an iterator

Now if we print it, it will produce



```python
for i in cube(3):
    print(i)
```

```
0
1
8
```

So the difference between generator and iterator is

1. To create a iterator we use iter(). Whereas to create generator we use function followed by keyword yield.

2. Generator use yield keyword. It saves the local variable value and it can return the local variable value.

3. Generator is used to write fast and compact code.

4. Python iterator is much more memory efficient.

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

Ans: Generator function must be followed by yield keyword.

4) What is the purpose of a yield statement?

Ans: Yield keyword is used to create a generator object which is basically an iterator. It stores value in local variable and return also return the local variable value.

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

Map function return a map object which is an iterator.

Syntax : map(functionname, iterables)

Functionname is the name of the function which is already defined and is to be executed for each item.

Iterables can be list, tuples or any other iterable object.

Map function returns a map object after applying the given function to each item of a given iterable (list, tuple etc.).

Example:

```
a = [4,5,67,89]
print(map(str, a))
list(map(str, a))
```

```
   <map object at 0x0000000004B3A940>

['4', '5', '67', '89']
```

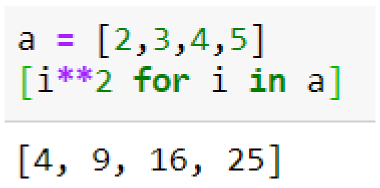
```
a = [2,3,4,5]
print(map(lambda i: i**2, a))
list(map(lambda i: i**2, a))
```

```
   <map object at 0x0000000004B3A1D0>

[4, 9, 16, 25]
```

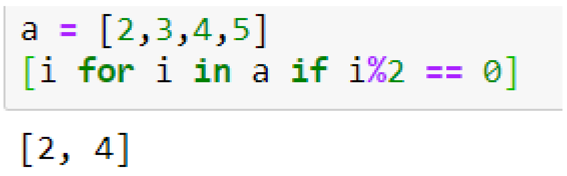
List comprehension is a substitute of map function and lambda function.

Syntax: [**expression** for **item** in **list** if **conditional**]



```
a = [2,3,4,5]
[i**2 for i in a]
```

```
[4, 9, 16, 25]
```



```
a = [2,3,4,5]
[i for i in a if i%2 == 0]
```

```
[2, 4]
```

<u>Difference between list comprehension and map function</u>

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.

3. List comprehension are used when a list of results is required as 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.