PD LAB
ASSIGNMENT - 7

Name: Raunak Thanawala

Registration Number: 231070051

Branch: Computer Engineering

Batch: 3

Aim:-

Write programs implements List comprehensions

Theory:-

Syntax:

- newList = [expression(element) for element in oldList if condition]

Parameter:

- **expression**: Represents the operation you want to execute on every item within the iterable.
- **element**: The term “variable” refers to each value taken from the iterable.

- **iterable**: specify the sequence of elements you want to iterate through.(e.g., a list, tuple, or string).
- **condition**: (Optional) A filter helps decide whether or not an element should be added to the new list.

Return:

- The return value of a list comprehension is a new list containing the modified elements that satisfy the given criteria.

Here's how list comprehension is used in the code:

1. **Doubling elements**:
    - [x*2 for x in numbers]
    - Creates a new list by multiplying each element by 2.
2. **Squaring elements**:
    - [x*x for x in numbers]
    - Squares each element in the list.
3. **Copying a list**:
    - [n for n in numbers]
    - copies numbers into a new list.
4. **Filter even numbers**:
    - [i for i in range(11) if i % 2 == 0]

    - filters even numbers from 0 to 10.
5. **Matrix generation**:
    - [[i for i in range(3)] for j in range(3)]
    - creates a 3x3 matrix.
6. **String to list of characters**:
    - [ch for ch in "Geeks for Geeks"]
    - creates a list of characters from the string.
7. **Benchmarking**:
    - It compares the time taken for list comprehension versus traditional for-loops.
8. **Matrix creation**:
    - [[j for j in range(5)] for i in range(3)]
    - generates a 3x5 matrix.
9. **Mapping with lambda**:
    - list(map(lambda i:i*10, [i for i in range(1,6)]))
    - multiplies each element by 10.
10. **Conditional comprehension**:
    - ["Even" if i % 2 == 0 else "Odd" for i in range(8)]
    - labels elements as "Even" or "Odd".
11. **Multiple conditions**:
    - [n for n in range(100) if n % 10 == 0 if n % 3 == 0]

    - filters numbers divisible by 10 and 3.
12. **Matrix transpose**:
    - [[i[j] for i in twoDMatrix]
    - for j in range(len(twoDMatrix[0]))]` transposes the 2D matrix.
13. **Flattening**:
    - [i for sublist in twoDMatrix for i in sublist]
    - flattens a matrix into a single list.
14. **Toggle case**:
    - list(map(lambda i: chr(ord(i) ^ 32), string))
    - toggles the case of each character.
15. **Reversing strings**:
    - [string[::-1] for string in ('Geeks', 'for', 'Geeks')]
    - reverses each word in a tuple.
16. **Zipping lists**:
    - [(name, age) for name, age in zip(names, ages)]
    - zips two lists into tuples.
17. **Sum of digits**:
    - [sum(n) for n in FL if n & 1]
    - filters odd numbers and computes the sum of their digits.
18. **Cubing elements**:

    - [i**3 for i in numbers]
    - computes the cube of each element.

19. **Length of names:**
    - [len(i) for i in names]
    - returns the length of each name in the list.

Code and Output:

```python
import time

numbers = [1,2,3,4,5]
doubled = [x*2 for x in numbers]
print(doubled)

squared = [x*x for x in numbers]
print(squared)

copy = [n for n in numbers]
print(copy)

l = [i for i in range(11) if i % 2 == 0]
print(l)

matrix = [[i for i in range(3)] for j in range(3)]
print(matrix)

List = [ch for ch in "Geeks for Geeks"]
print(List)

def for_loop(n):
   result = []
   for i in range(n):
       result.append(i**2)
   return result
begin = time.time()
```

```
for_loop(10**6)
end = time.time()
print('Time taken for_loop:', round(end-begin, 5))

def list_comprehension(n):
   return [i**2 for i in range(n)]
begin = time.time()
list_comprehension(10**6)
end = time.time()
print('Time taken for list_comprehension:', round(end-begin, 5))

mat = [[j for j in range(5)] for i in range(3)]
print(mat)

num = list(map(lambda i:i*10, [i for i in range(1,6)]))
print(num)

lis =[ "Even" if i%2 == 0 else "Odd" for i in range(8)]
print(lis)

li = [ n for n in range(100) if n%10==0 if n%3==0]
print(li)

twoDMatrix = [[10, 20, 30],
           [40, 50, 60],
           [70, 80, 90]]
trans = [[i[j] for i in twoDMatrix] for j in range(len(twoDMatrix[0]))]
print(trans)

flat = [i for sublist in twoDMatrix for i in sublist]
print(flat)

string = "Geeks4Geeks"
LL = list(map(lambda i: chr(ord(i) ^ 32), string))
print(LL)

NewL = [string [::-1] for string in ('Geeks', 'for', 'Geeks')]
print(NewL)

names = ['Rahul', 'Vansh', 'Sid']
```

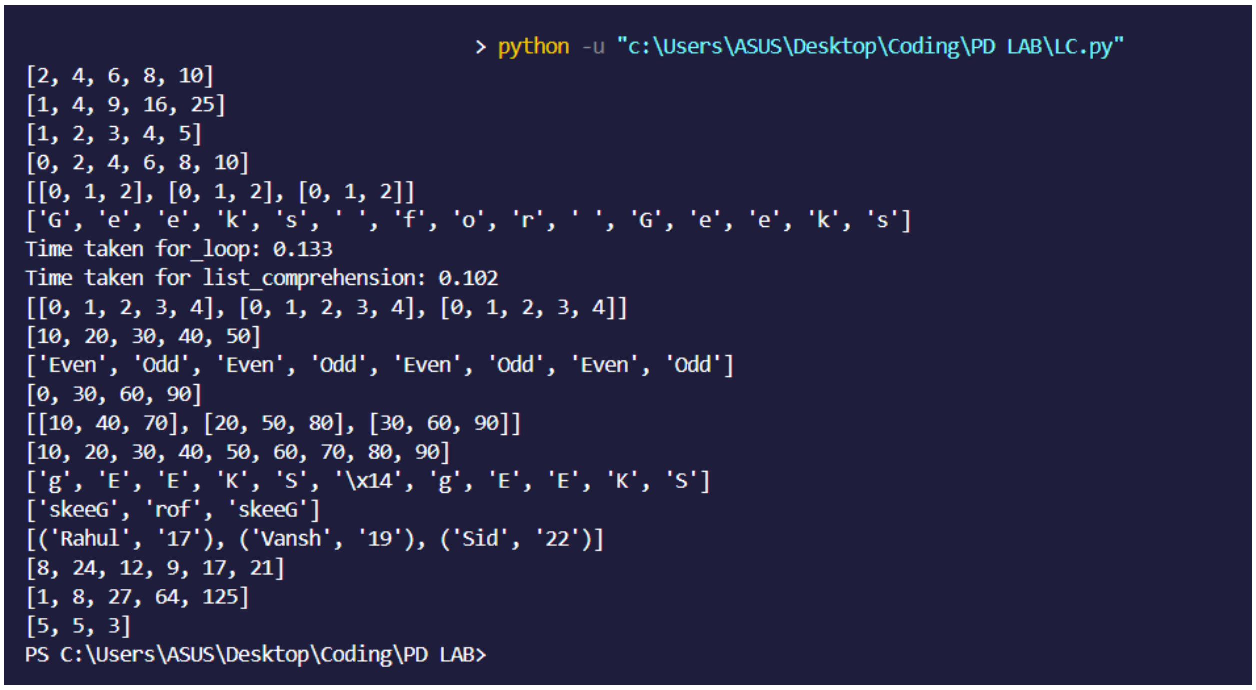
```
ages = ['17', '19', '22']
NAtuples = [(name,age) for name,age in zip(names, ages)]
print(NAtuples)

def sum(n):
    s = 0
    for i in str(n):
        s+=int(i)
    return s
FL = [251, 556, 969, 345, 772, 801, 4373, 9921]
FFL = [sum(n) for n in FL if n&1]
print(FFL)

cube = [i**3 for i in numbers]
print(cube)

length = [len(i) for i in names]
print(length)
```

OUTPUT:



```
                                             > python -u "c:\Users\ASUS\Desktop\Coding\PD LAB\LC.py"
[2, 4, 6, 8, 10]
[1, 4, 9, 16, 25]
[1, 2, 3, 4, 5]
[0, 2, 4, 6, 8, 10]
[[0, 1, 2], [0, 1, 2], [0, 1, 2]]
['G', 'e', 'e', 'k', 's', ' ', 'f', 'o', 'r', ' ', 'G', 'e', 'e', 'k', 's']
Time taken for_loop: 0.133
Time taken for list_comprehension: 0.102
[[0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]
[10, 20, 30, 40, 50]
['Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd']
[0, 30, 60, 90]
[[10, 40, 70], [20, 50, 80], [30, 60, 90]]
[10, 20, 30, 40, 50, 60, 70, 80, 90]
['g', 'E', 'E', 'K', 'S', '\x14', 'g', 'E', 'E', 'K', 'S']
['skeeG', 'rof', 'skeeG']
[('Rahul', '17'), ('Vansh', '19'), ('Sid', '22')]
[8, 24, 12, 9, 17, 21]
[1, 8, 27, 64, 125]
[5, 5, 3]
PS C:\Users\ASUS\Desktop\Coding\PD LAB>
```

CONCLUSION:

Thus we have written a program that shows the various ways we can use list comprehension and the advantages of using list comprehension.
We have also explained what each snippet of code does.Thus we have understood the use cases of list comprehension.