

NAME: SIDDHI PAREKH
REG NO.: 221071047
BATCH: C

SY CE

Experiment: 2

AIM: Programs to implement List Comprehensions in Python

THEORY:

List comprehension is an elegant way to define and create lists based on existing lists. List comprehension is generally more compact and faster than normal functions and loops for creating list. However, we should avoid writing very long list comprehensions in one line to ensure that code is user-friendly.

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

The Syntax

`newlist = [expression for item in iterable if condition == True]`

The return value is a new list, leaving the old list unchanged.

List comprehension can be done in following ways:

```
lst=[1,2,3,4,5]
```

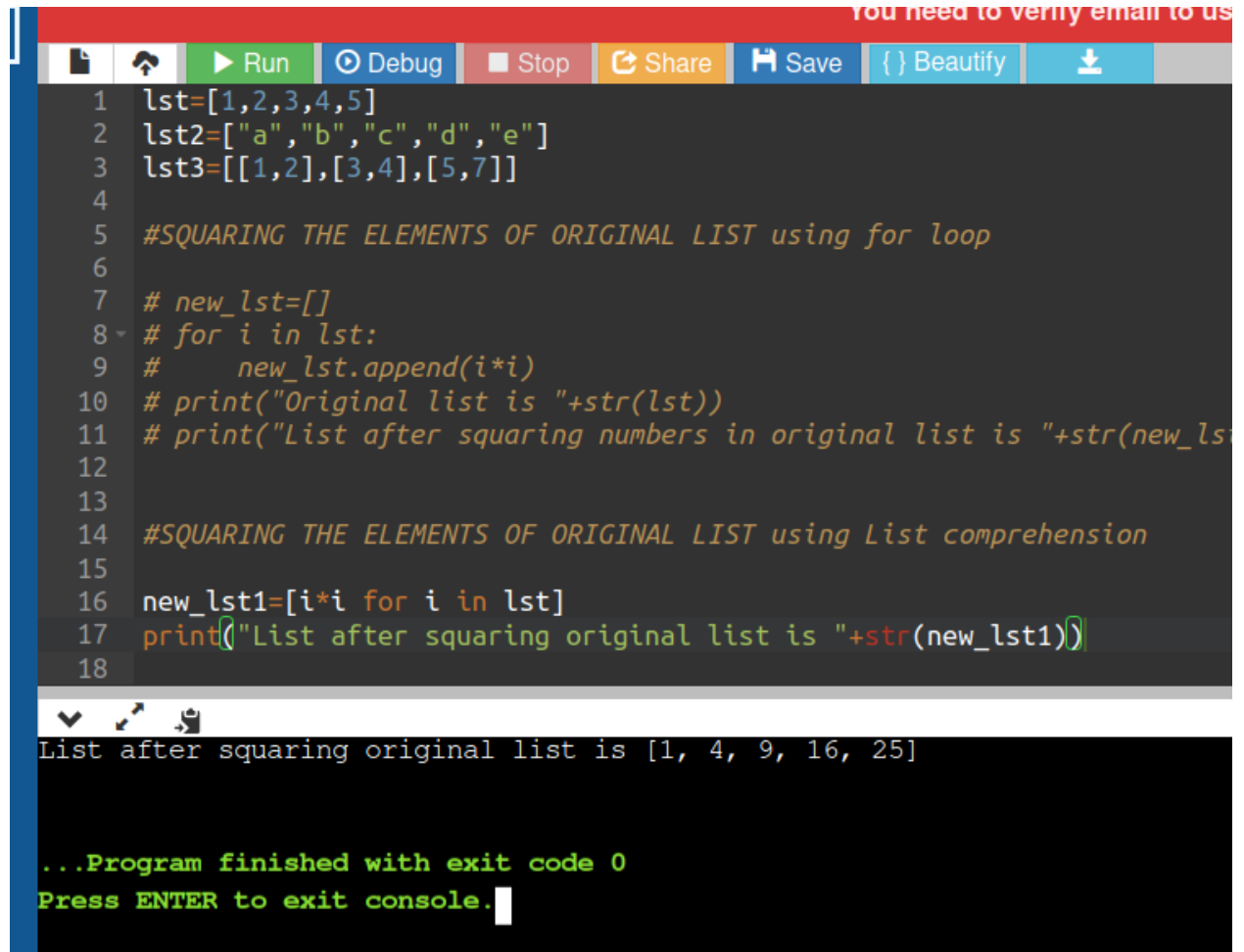
```
lst2=["a","b","c","d","e"]
```

```
lst3=[[1,2],[3,4],[5,7]]
```

1. Squaring the elements original list to store them in new list

CODE :

```
new_lst1=[i*i for i in lst]
print("List after squaring original list is "+str(new_lst1))
```



The screenshot shows a Python IDE with a dark theme. The top toolbar includes icons for file operations, a 'Run' button, 'Debug', 'Stop', 'Share', 'Save', 'Beautify', and a download icon. A red banner at the top right says 'You need to verify email to us'. The code editor contains the following Python code:

```
1 lst=[1,2,3,4,5]
2 lst2=["a","b","c","d","e"]
3 lst3=[[1,2],[3,4],[5,7]]
4
5 #SQUARING THE ELEMENTS OF ORIGINAL LIST using for loop
6
7 # new_lst=[]
8 # for i in lst:
9 #     new_lst.append(i*i)
10 # print("Original list is "+str(lst))
11 # print("List after squaring numbers in original list is "+str(new_lst))
12
13
14 #SQUARING THE ELEMENTS OF ORIGINAL LIST using List comprehension
15
16 new_lst1=[i*i for i in lst]
17 print("List after squaring original list is "+str(new_lst1))
18
```

The output console at the bottom shows the result of the execution:

```
List after squaring original list is [1, 4, 9, 16, 25]

...Program finished with exit code 0
Press ENTER to exit console.
```

2. Using conditions:

THEORY:

In 1 line, for loop and if condition can be used.

CODE:

```
new_lst2=[i for i in range(50) if i%10 == 0]
print("List after finding multiples of 10 from 0 to 50 is "
      +str(new_lst2))
```

```
19 new_lst2=[i for i in range(50) if i%10 == 0]
20 print("List after finding multiples of 10 from 0 to 50 is "+str(new_lst2))
21

List after finding multiples of 10 from 0 to 50 is [0, 10, 20, 30, 40]

...Program finished with exit code 0
Press ENTER to exit console.
```

```
new_lst3=[i for i in range(80) if i%10 == 0 and i%7 == 0]
print("List after finding multiples of 10 and 7 from 0 to 80 is "
      +str(new_lst3))
```

```
21
22 new_lst3=[i for i in range(80) if i%10 == 0 and i%7 == 0]
23 print("List after finding multiples of 10 and 7 from 0 to 80 is "+str(new_lst3))
24

List after finding multiples of 10 and 7 from 0 to 80 is [0, 70]

...Program finished with exit code 0
Press ENTER to exit console.
```

3. List comprehensions in strings:

CODE:

```
new_lst4=[i for i in "siddhi"]
print("List containing letters of string are "+str(new_lst4))
```

```
24
25 new_lst4=[i for i in "siddhi"]
26 print("List containing letters of string are "+str(new_lst4))
27
```

▼ ↗ 📄

List containing letters of string are ['s', 'i', 'd', 'd', 'h', 'i']

...Program finished with exit code 0
Press ENTER to exit console. □

4. Using nested for loops:

THEORY:

Writing for loops in 1 line implies a nested loop, where first for loop is executed first and then the second.

CODE:

```
new_lst4=[(i,j) for i in lst for j in lst2]
print(new_lst4)
```

```
28
29 new_lst4=[(i,j) for i in lst for j in lst2]
30 print(new_lst4)
31
```

▼ ↗ 📄

input

[(1, 'a'), (1, 'b'), (1, 'c'), (1, 'd'), (1, 'e'), (2, 'a'), (2, 'b'), (2, 'c'), (2, 'd'), (2, 'e'), (3, 'a'), (3, 'b'), (3, 'c'), (3, 'd'), (3, 'e'), (4, 'a'), (4, 'b'), (4, 'c'), (4, 'd'), (4, 'e'), (5, 'a'), (5, 'b'), (5, 'c'), (5, 'd'), (5, 'e')]

...Program finished with exit code 0
Press ENTER to exit console. □

THEORY:

Down below, the for loop written outside the brackets is executed first and then the second is executed.

The following code is used to find the transpose of the matrix inside the main list lst3.

CODE:

```
new_lst5=[[j[i] for j in lst3]for i in range(len(lst3[0]))]  
print(new_lst5)
```

```
28  
29 # new_lst4=[(i,j )for i in lst for j in lst2]  
30 # print(new_lst4)  
31  
32 new_lst5=[[j[i] for j in lst3]for i in range(len(lst3[0]))]  
33 print(new_lst5)  
34
```

```
[[1, 3, 5], [2, 4, 7]]
```

...Program finished with exit code 0
Press ENTER to exit console.

5. Using lambda functions:

CODE:

```
new_lst6 = list(map(lambda i: i*10, [i for i in lst]))  
print(new_lst6)
```

```
34
35 new_lst6 = list(map(lambda i: i*10, [i for i in lst]))
36 print(new_lst6)
37
[10, 20, 30, 40, 50]

...Program finished with exit code 0
Press ENTER to exit console.
```

6. Comparing For loop and List comprehension to prove that List Comprehension is more efficient.

THEORY:

To compare time complexity, time library is imported which contains functions like begin, end, round, which count time of the functions made by the user, and hence its found out that list comprehensive is less time consuming than using for loop alone.

CODE:

```
import time

new_lst7=[]
new_lst8=[]

def for_loop(n):
    for i in range(n): new_lst7.append(i*10)
    #print(new_lst7)
```

```
def List_comprehension(n):  
    new_lst8 = [i*10 for i in range(n)]  
    #print(new_lst8)
```

```
begin = time.time()  
for_loop(10**7)  
end = time.time()
```

```
print("Time taken by the for loop is: ", round(end-begin , 2))
```

```
begin = time.time()  
List_comprehension(10**7)  
end = time.time()
```

```
print("Time taken by list comprehension is: ",  
      round(end-begin , 2))
```

```

37
38 import time
39
40 new_lst7=[]
41 new_lst8=[]
42
43 def for_loop(n):
44     for i in range(n): new_lst7.append(i*10)
45     #print(new_lst7)
46
47 def List_comprehension(n):
48     new_lst8 = [i*10 for i in range(n)]
49     #print(new_lst8)
50
51 begin = time.time()
52 for_loop(10**7)
53 end = time.time()
54
55 print("Time taken by the for loop is: ", round(end-begin , 2))
56
57 begin = time.time()
58 List_comprehension(10**7)
59 end = time.time()
60
61 print("Time taken by list comprehension is: ", round(end-begin , 2))
62

```

Time taken by the for loop is: 1.02
Time taken by list comprehension is: 0.58

...Program finished with exit code 0
Press ENTER to exit console.

7. Using list comprehension using functions:

THEORY:

Below code shows how functions can be called inside the lists, along with for loop or condition if any.

CODE:

```
lst4 = [31,24,55,66]
def sum(n):
    summ=0
    for i in str(n):
        summ = summ + int(i)
    return summ

new_lst9 = [sum(i) for i in lst4 if (i & 1)]

print(new_lst9)
```

```
62
63 lst4 = [31,24,55,66]
64 def sum(n):
65     summ=0
66     for i in str(n):
67         summ = summ + int(i)
68     return summ
69
70 new_lst9 = [sum(i) for i in lst4 if (i & 1)]
71
72 print(new_lst9)
73
74
75
```

[4, 10]

...Program finished with exit code 0
Press ENTER to exit console.

CONCLUSION :

1. Thus list comprehension is quite an efficient way of dealing with lists.
2. It also has a function called map,lambda, which is also efficient.
3. Thus list comprehensions and its various types using for loops, if conditions, nested for loops is studied.