MCT-242 : COMPUTER PROGRAMMING-I

using Python 3.9

Prepared By:

Mr. Muhammad Ahsan Naeem

YouTube Playlist

https://www.youtube.com/playlist?list=PLWF9TXck7O_wMDB-VriREZ6EvwkWLNB7q

LAB 21: List Comprehension: CLO-4

If you print a variable, say `x` , and get result at the output as:

```
[5, 2, 'Computer', 10]
```

You can readily see that `x` is an object of list class because of `[]`. It's not just us who can comprehend the output as list, but the Python interpreter can also comprehend data types by itself and this leads to a special syntax known as **List Comprehension** for lists and the same is for set and dictionary known as Set Comprehension and Dictionary Comprehension. There is no Tuple Comprehension.

List comprehension is a special way to create a list, usually from another list but it can be some other iterable too. If we have a list of numbers and we want to create another list with squared numbers, we can do it without list comprehension using `for` loop as show here:

```
nums=[5,3,4,6,8]
sqNums=[]
for i in nums:
    sqNums.append(i*i)
print(sqNums)
```

The secod possibility is to do it using List Comprehension and is shown here:

```
sqNums=[i*i for i in nums]
print(sqNums3)
```

When the expression is written in between `[]`, the interpreter comprehends it for a list and the final result is list data type. Below is the comparison between the two approaches:

Without List Comprehension	Using List Comprehension

`nums=[5,3,4,6,8]`
`sqNums=[]`
`for i in nums:`
`    sqNums.append(i*i)`
	`nums=[5,3,4,6,8]`
	`sqNums=[i*i for i in nums]`

See carefully placement of different parts in both methods indicated by same color. In **List Comprehension** we first write what we want to generate from the other list and then we specify the list with `for` loop.

At beginning it seems a bit confusing but with little practice you will prefer to create new lists using List Comprehension rather than simple `for` loop.

We can also apply conditions within list comprehension. For example, to create a list of squares of the even numbers only from the original list, we can do it with List Comprehension. Again, the comparison with simple **for** loop and List Comprehension is shown here:

Without List Comprehension	**Using List Comprehension**

`nums=[5,3,4,6,8]`
`sqNums=[]`
`for i in nums:`
`    if i%2==0:`
`        sqNums.append(i*i)`
	`nums=[5,3,4,6,8]`
	`sqNums=[i*i for i in nums if i%2==0]`

Lets see another example:

Suppose we have a list of numbers, and we want to generate a list of Natural Log of those numbers, we can do it as:

```
from math import log
nums=[5,3,4,6,8]
logNums=[log(i) for i in nums]
print(logNums)
```

Filtering the Elements with List Comprehension:

We can also apply the if condition to filter out some elements in list comprehension. For example, if we have a list and we want to generate another list with only the prime numbers from the original list, we can do it as:

```
from math import sqrt
def isPrime(n):
    for i in range(2,int(sqrt(n))+1):
        if (n%i==0):
            return False
    return True


## Main Program ##

nums=[23,12,13,14,21,37,92,12]
primes=[x for x in nums if isPrime(x)]
print(primes)
```

Tasks:

[1] Write a program that will take an integer from the user and will create a list (using list comprehension) of all factors of that number. You should use the range function inside the List Comprehension.

[2] There is a List of numbers. Write a program that will generate a new list with the square of the numbers of the original list if that is an even number.

[3] There is list of inner lists all having the numbers. Generate a list having the sum of inner lists as its elements. For example, if this is the inner list:

```
a=[[4,3,-2,0,1],[6,8,2,9],[5,3,4,-7],[2,9]]
```

Then this should be the output:

```
[6, 25, 5, 11]
```

[4] Suppose a student's information is stored inside a **Tuple** and there is a **List** of ten students as shown here:

```
s1=('Ahmad','Anwar','MCT-UET-01',['CP','LA'])
s2=('Ali','Jamal','MCT-UET-02',['LA'])
s3=('Muneeb','Akhtar','MCT-UET-03',['Phy','CP','LA'])
s4=('Rizwan','Khan','MCT-UET-04',['Statistics','Phy'])
s5=('Akhtar','Hussain','MCT-UET-05',['Phy','LA'])
s6=('Nasir','Saeed','MCT-UET-06',['Phy','Statistics','LA'])
s7=('Bilal','Akram','MCT-UET-07',['Hist','CP','LA'])
s8=('Hamid','Salman','MCT-UET-08',['CP','Phy'])
s9=('Naveed','Majeed','MCT-UET-09',['CP','LA','Hist'])
s10=('Kashif','Lateef','MCT-UET-10',['LA','Phy','Hist'])

allStudents=[s1,s2,s3,s4,s5,s6,s7,s8,s9,s10]
```

Generate these two lists using the **List Comprehension**:

- List of the registration numbers of all students who are registered in the course CP.
- List of the full names of the students who have registered exactly 3 courses.
- List of number of courses registered by each student.

List Comprehension on some other iterable:

We can generate a list using the list comprehension from any iterable other than just a List. For example, here list is generated from the elements of a Tuple:

```
nums=(5,3,4,6,8)
sqNums=[i*i for i in nums]
print(sqNums)
```

The final result is of **List Comprehension** is a List and it can be applied on any iterable or it can even be a List of some other iterable. For example, if we want to generate a List of single-valued Tuples from a list of numbers, we can do it as:

```
nums=[5,3,4,6,8]
a=[(i,) for i in nums]
print(a)
```

Now let's see a better case. Suppose we have two lists, one representing the **x-components** and the other representing the **y-components** of **XY Points** as:

```
x=[1,-1,5,0,3]
y=[2,-1,7,4,10]
```

Now suppose we want to generate a list of XY point where a XY point is a Tuple as **(x,y)**. For that we need to pick one element from list **x** and the second from list **y**. To process two lists, we can process those using the index.

```
a=[(x[i],y[i]) for i in range(len(x))]
print(a)
```

By the way, to process multiple iterables at a time, instead of processing those using the index we can use the Python built-in function **zip()** which takes multiple iterables as input and will zips those together as Tuples; first having the first element of each iterable, second with the second elements of each iterable and so on till the last element of each iterable. We will not see the detail of this zip function but just simply see the output of zip function on the two iterables:

```
x=[1,-1,5,0,3]
y=[2,-1,7,4,10]
a=zip(x,y)
for i in a:
    print(i)
```

It has the output as:

```
(1, 2)
(-1, -1)
(5, 7)
(0, 4)
(3, 10)
```

So in above case we can use it for list comprehension as:

```
x=[1,-1,5,0,3]
y=[2,-1,7,4,10]
```

```
a=[i for i in zip(x,y)]
print(a)
```

Actually, in this case even a better solution is to covert the zip object directly to the list since it already contains those Tuples:

```
x=[1,-1,5,0,3]
y=[2,-1,7,4,10]
a=list(zip(x,y))
print(a)
```

But lets say we want to generate the XY points as inner list `[x,y]` instead of the Tuple then we can do it as:

```
x=[1,-1,5,0,3]
y=[2,-1,7,4,10]
a=[[i,j] for i,j in zip(x,y)]
print(a)
```

Another very good advantage of the zip function over the first approach of processing the two (or more) lists (or other iterables) via index is that if the two lists are of different sizes, there can be error of index out of range (when we use the length of larger list in `range()` function for generating index). But in case of the zip function, it will zip together the elements till the smallest number of iterable.

Tasks:

[5] For the above case, where we have two lists, one representing the x-components and the other representing the y-components of XY Points for example as:

```
x=[1,-1,5,0,3,10,5,-4]
y=[2,-1,7,4,10,4,0,-5]
```

Generate a List of points as `(x,y)` tuple but only those points whose magnitude is less than or equal to 3.

[6] A list contains XY Points as Tuples for example as:

```
[(1, 2),(-1, -1),(5, 7),(0, 4),(3, 10),(10, 4),(5, 0),(-4, -5)]
```

Generate two lists; one having the x-component values and the one having the y-component values from the original list of XY Points.

Using if-else with List Comprehension:

We used the if condition to filter out some elements in **List Comprehension**. But there can be another situation of using the `if-else` statement while deciding the expression to evaluate on the list elements.

For example, suppose we have a list of numbers, and we want to double the number if it is even and multiply it by 3 if it is an odd. Then this `if-else` will be applied in the expression part of the List Comprehension as:

```
nums=[5,3,4,6,8]
a=[i*2 if i%2==0 else i*3 for i in nums]
print(a)
```

See carefully the `if-else` is applied before the `for` loop part. When the `if` statement is used to filter out some elements, that is used after the `for` loop. For example, in above case there might be some elements in the `nums` list other than integers for example as:

```
nums=[5,3,5.3,'Hello',4,6,8]
```

And we just don't want those inside the output list, we can do it as:

```
a=[i*2 if i%2==0 else i*3 for i in nums if type(i)==int]
print(a)
```

Nested for loop in List Comprehension:

Finally, we can have a nested `for` loop inside the **List Comprehension**. The simplest case of nested `for` loop for lists is getting a flat list from a nested list. We had a simple example of creating a flattened list for a nested list using nested `for` loop as given here:

```
numLists=[[3,4,1],[1,5,7,2,3],[2,8]]
flattened=[]
for i in numLists: # i will be one inner list
    for num in i: #num will be a number in nested list i
        flattened.append(num)
print(flattened)
```

Using list comprehension, we can do it like:

```
numLists=[[3,4,1],[1,5,7,2,3],[2,8]]
flattened=[num for i in numLists for num in i]
print(flattened)
```

As another case consider these two lists:

```
x=[5,6,2]
y=[3,4,9,1,7]
```

Now suppose we want to create XY points for all possibilities of x and y values i.e., all values of y for each value of x. We can do it using the nested for loop and list comprehension as:

```
points=[(i,j) for i in x for j in y]
print(points)
```

Tasks:

[7] For the Students task, generate a List of subjects registered by all students, starting from all registered subjects of first student till all registered subjects of the last student. Of course, there will be many repeated values.

[8] Create three lists using List Comprehension with names **x, y** and **z** having 10 floating values generated randomly between 0 and 10. Consider the values of the lists **x, y** and **z** as 3D point in space (X,Y,Z). Generate another list of 3D points as Tuples from the three lists, which lie inside the **Upper Half Sphere** of radius 10. Upper half means the part of the sphere in +ve y-direction only.