**List Comprehension**

In this blog we will discuss list Comprehension in detail which is quite popular in python programming language.

Before diving into list Comprehension, we need to understand the downside of regular function using loop while iterating list items.

**Example**: suppose we want to create a list of squares.

items = [1,2,3,4,5]

Square\_list = []

for x in items:

Square\_list.append(x\*\*2)

print("Square List=",Square\_list)

print("x=",x)

**Output**:

Square List= [1, 4, 9, 16, 25]

x= 5

**Note** that this creates (or overwrites) a variable named x that still exists after the loop completes.

Now dive into list comprehensions.

**List Comprehension:**

List comprehensions provide a concise way to create lists based on existing list. It is more concise and readable expression. The same statement can be written using list compression.

**Example**:

items = [1,2,3,4,5]

Square\_list = [x\*\*2 for x in items]

Square\_list

**Output:**

[1, 4, 9, 16, 25]

Syntaxes of list comprehensions.

**[expression for item in list]**

**Basic**:

**[x for x in list]**

**With IF condition:**

**[x for x in list if condition]**

**With two For statement:**

**[(x,y) for x in list1 for y in list2 if x!=y ]**

Hence, A list comprehension consists of brackets containing an expression followed by a

**for** clause, then zero or more **for** or if clauses. The result will be a new list resulting from

evaluating the expression in the context of the for and if clauses which follow it.

**List comprehension with Conditional statement.**

List comprehensions can also deal with conditional statement to modify existing list or tuple. In this demonstration, we will use conditional statement to make square of only even number between 1 to 20.

**Example:**

**squarelist = [x\*\*2 for x in range(1,21) if x%2 == 0]**

**squarelist**

**output**:

[4, 16, 36, 64, 100, 144, 196, 256, 324, 400]

**IF…ELSE with List comprehension:**

If you want to use IF and else statement in the expression section you can easily use it. See the below example with **IF** and **else** in the list comprehension.

**Example**:

**list1 = ["Even" if x%2==0 else "Odd" for x in range(10)]**

**print(list1)**

**Output:**

['Even','Odd','Even','Odd','Even','Odd','Even','Odd','Even','Odd']

**Nested List in list comprehension:**

Since list comprehension is very flexible, we can also use nested if statement in the list comprehension. In the below program, we are listing only number which is either divisible by 2 or 5.

**Example:**

list\_item = [x for x in range(1,101) if x % 2 == 0 if x % 5 == 0]

print(num\_list)

**Output:**

[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

**Create list of tuples using list comprehension**:

We can also create list of tuples using list comprehension. In the below program ,we are creating tuple of number and square of that number.

**Example:**

**list1 = [(x,x\*\*2) for x in range(10)]**

**list1**

**Output:**

[(0, 0),(1, 1),(2, 4),(3, 9),(4, 16),

(5, 25),(6, 36),(7, 49),(8, 64),(9, 81)]

**Nested ‘FOR’ with List comprehension:**

List comprehension also support an elegant way to use nested ‘**For’** for filtering or updating items. See the below example.

**Example:**

[(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]

**Output:**

**[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]**

**It is equivalent to:**

**list\_tuples = []**

**for x in [1,2,3]:**

**for y in [3,1,4]:**

**if x != y:**

**list\_tuples.append((x, y))**

**list\_tuples**

**Nested List Comprehension:**

List comprehension also facilitate us to work with nested list Let’s suppose we have nested list and we want to convert it to a single list.

**Example:**

matrix = [[1, 2, 3, 4],[5, 6, 7, 8],[9, 10, 11, 12]]

l1 = [x for row in matrix for x in row]

l1

**Output:**

**[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]**

**It is equivalent to:**

**list1 = []**

**for row in matrix:**

**for item in row:**

**list1.append(item)**

**list1**

**Key points to remember.**

* 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 more readable.