Lists

July 18, 2022

Python Data Structures - Lists

Lists

A list can be defined as a collection of values or items of different types. The items in the list are separated with the comma (,) and enclosed with the square brackets [ ] Refrered as Sequence / Container / Collection of Items

Characteristics of Lists :

The lists are ordered

The element of the list can access by index.

The lists are the mutable type

A list can have any number of items and they may be of different types (Heterogenous)

A list may contain duplicate values

Creating a List

[1]:
```
#Lists in Python can be created by just placing the sequence inside the square␣
↪brackets[] separated by commas (,).

# empty list
emptyList = []
print(emptyList ,":", type(emptyList))

# list of integers
integerElements = [1, 2, 3]
print(integerElements ,":", type(integerElements))

# list of strings
stringElements = ["Welocme", "To", "Python","Tutorial"]
print(stringElements ,":", type(stringElements))
```

```
[] : <class 'list'>
[1, 2, 3] : <class 'list'>
['Welocme', 'To', 'Python', 'Tutorial'] : <class 'list'>
```

[2]:
```
#A list can have any number of items and they may be of different types
```

```
# list with mixed data types - Heterogenous
heterogenousElements = [1, "Hello", 3.4]
print(heterogenousElements,":", type(heterogenousElements))

#List Can have Duplicte Vales
duplicatesList = [1, 2, 4, 4, 3, 3, 3, 6, 5]
print(duplicatesList  ,":", type(duplicatesList))

#A list can also have another list as an item. This is called a nested list.

# nested list
nestedList = ["mouse", [8, 4, 6], ['a']]
print(nestedList ,":", type(nestedList))
```

```
[1, 'Hello', 3.4] : <class 'list'>
[1, 2, 4, 4, 3, 3, 3, 6, 5] : <class 'list'>
['mouse', [8, 4, 6], ['a']] : <class 'list'>
```

[3]:
```
#Taking Input of a List
string = input("Enter elements of a list (Space-Separated): ")
lst = string.split()  # split the strings and store it to a list
print('The list is:', lst)
```

```
Enter elements of a list (Space-Separated): 10 20 30 40 50
The list is: ['10', '20', '30', '40', '50']
```

Accessing elements from the List Using Index

The indexing is processed in the same way as it happens with the strings. The elements of the list can be accessed by using the index operator [ ] - list_name[Index Number] And lists can be sliced, concatenated and so on.

[4]:
```
#The index starts from 0 and goes to length - 1
my_list = ['red', 'green', 'blue', 'yellow', 'black']

#len of list
print(len(my_list))

#Postive Indexing

# first item
print(my_list[0])  # red

# third item
print(my_list[2])  # blue

# fourth item
print(my_list[3])  # yellow
```

```
#Negetive Indexing

# last item
print(my_list[-1]) #black

# fourth last item
print(my_list[-4]) #green
```

```
5
red
blue
yellow
black
green
```

```
[53]: # Error! Only integer can be used for indexing
print(my_list[4.0])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-53-b36699f2f7b3> in <module>
      1 # Error! Only integer can be used for indexing
----> 2 print(my_list[4.0])

TypeError: list indices must be integers or slices, not float
```

```
[54]: # Triggers IndexError: list index out of range
print(my_list[10])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-54-647c57c7b09b> in <module>
      1 # Triggers IndexError: list index out of range
----> 2 print(my_list[10])

IndexError: list index out of range
```

Access Nested List Items

```
[56]: # Nested List
n_list = ["Happy", [2, 0, 1, 5]]

# Nested indexing
print(n_list[0])

print(n_list[1])
print(n_list[1][2])
```

```
Happy
[2, 0, 1, 5]
1
```

[59]:
```
nestedList = ['a', 'b', ['cc', 'dd', ['eee', 'fff']], 'g', 'h']

print(nestedList[2][1])

print(nestedList[2][2]) # Prints ['eee', 'fff']

print(nestedList[2][2][1]) # Prints eee
```

```
dd
['eee', 'fff']
fff
```

Slicing A List

A segment of a list is called a slice and you can extract one by using a slice operator[:]. A slice of a list is also a list.

[8]:
```
L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']



##Slice with postive Indices
print(L[2:7])    # Prints ['c', 'd', 'e', 'f', 'g']

#Slice with Negative Indices
print(L[-7:-2])  # Prints ['c', 'd', 'e', 'f', 'g']

#Slice with Positive & Negative Indices
print(L[2:-5])   # Prints ['c', 'd']

#Specify Step of the Slicing
print(L[2:7:2])  # Prints ['c', 'e', 'g']

#Negative Step Size
print(L[6:1:-2]) # Prints ['g', 'e', 'c']

#Slice at Beginning & End
print(L[:3])     # Prints ['a', 'b', 'c']

print(L[6:])     # Prints ['g', 'h', 'i']

#Reverse a List
print(L[::-1])   # Prints ['i', 'h', 'g', 'f', 'e', 'd', 'c', 'b', 'a']
```

```
['c', 'd', 'e', 'f', 'g']
['c', 'd', 'e', 'f', 'g']
['c', 'd']
['c', 'e', 'g']
['g', 'e', 'c']
['a', 'b', 'c']
['g', 'h', 'i']
['i', 'h', 'g', 'f', 'e', 'd', 'c', 'b', 'a']
```

Change List Elements

Lists are mutable, meaning their elements can be changed unlike string or tuple.. We can use the assignment operator = to change an item or a range of items.

[9]:
```
# Correcting mistake values in a list
odd = [2, 4, 6, 8]
print('Before Changeing :',odd)

# change the 1st item
odd[0] = 1

print('After Changeing 1st Element:',odd)

#Modify multiple list items - change 2nd to 4th items
odd[1:4] = [3, 5, 7]

print('After Changeing Multiple Elements:',odd)
```

```
Before Changeing : [2, 4, 6, 8]
After Changeing 1st Element: [1, 4, 6, 8]
After Changeing Multiple Elements: [1, 3, 5, 7]
```

Add List Elements

We can add one item to a list using the append( ) method

We can add several items using the extend( ) method.

We can insert one item at a desired location by using the method insert( )

Insert multiple items by squeezing it into an empty slice of a list

[10]:
```
# Appending lists in Python
odd = [1, 3, 5]
print('Before Changeing :',odd)

odd.append(7)

print('Appended List :', odd)

#Extending lists in Python
```

```python
odd.extend([9, 11, 13])

print('Extended List :', odd)

# list insert() method
odd = [1, 9]

odd.insert(1,3) # insert(index,value)

print('Inserted List :', odd)

#empty slice of list
#odd[:0]  = Insert at the start

odd[2:2] = [5, 7]

print('Insert in the middle :', odd)

odd[len(odd):] = [11,13]

print('Insert at the end :',odd)
```

```
Before Changeing : [1, 3, 5]
Appended List : [1, 3, 5, 7]
Extended List : [1, 3, 5, 7, 9, 11, 13]
Inserted List : [1, 3, 9]
Insert in the middle : [1, 3, 5, 7, 9]
Insert at the end : [1, 3, 5, 7, 9, 11, 13]
```

Delete List Elements

We can delete one or more items from a list using the Python del statement

We can use remove( ) to remove the given item.

We can use pop( ) to remove an item at the given index.

If we have to empty the whole list, we can use the clear( ) .method

we can also delete items in a list by assigning an empty list to a slice of elements.

[11]:
```python
# Deleting list items
my_list = ['p', 'r', 'o', 'b', 'l', 'e', 'm']
print('Before Deleting :',my_list)

# delete one item
del my_list[2]

print('After Deleting 3rd Element:',my_list)
```

```python
# delete multiple items
del my_list[1:5]

print('After Deleting Multiple Elements :',my_list)

# delete the entire list
del my_list

print('After Deleting Entire List')
# Error: List not defined
print(my_list)
```

```
Before Deleting : ['p', 'r', 'o', 'b', 'l', 'e', 'm']
After Deleting 3rd Element: ['p', 'r', 'b', 'l', 'e', 'm']
After Deleting Multiple Elements : ['p', 'm']
After Deleting Entire List
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-11-263717e52d4f> in <module>
     18 print('After Deleting Entire List')
     19 # Error: List not defined
---> 20 print(my_list)

NameError: name 'my_list' is not defined
```

[60]:
```python
#List Methods - Remove ,Pop , Clear

my_list = ['p','r','o','b','l','e','m']
print('Before Removing :',my_list)

#remove - Triggers ValueError if element not present
my_list.remove('p') # Output: ['r', 'o', 'b', 'l', 'e', 'm']
print('After Removing Element p :',my_list)

#pop
print('Popped Element : ',my_list.pop(1)) # Output: 'o'

print('After Popping Element o using index :',my_list)      # Output: ['r', 'b',␣
 ↪'l', 'e', 'm']

'''
The pop() method removes and returns the last item if the index is not provided.
 ↪
This helps us implement lists as stacks (first in, last out data structure).
'''
```

```
print('Popped Element : ',my_list.pop())  # Output: 'm'

print('After Popping Last Element:',my_list)    # Output: ['r', 'b', 'l', 'e']

#clear
my_list.clear()

print('After Clearing The List:',my_list)    # Output: []
```

```
Before Removing : ['p', 'r', 'o', 'b', 'l', 'e', 'm']
After Removing Element p : ['r', 'o', 'b', 'l', 'e', 'm']
Popped Element :  o
After Popping Element o using index : ['r', 'b', 'l', 'e', 'm']
Popped Element :  m
After Popping Last Element: ['r', 'b', 'l', 'e']
After Clearing The List: []
```

Time Complexity for the above methods, considering n elements in the List and i is the index mentioned:

del O(n - i)

pop O(n - i)

remove O(n)

[13]:
```
#assigning the appropriate slice to an empty list.

L = ['a', 'b', 'c', 'd', 'e']
print('List Before Reassigning :',L)

L[1:5] = []
print('List After Reassigning :',L)  # Prints ['a']

#You can also use the del statement with the same slice.
L = ['a', 'b', 'c', 'd', 'e']

del L[1:5]

print('List After Deleting a Slice :',L)  # Prints ['a']
```

```
List Before Reassigning : ['a', 'b', 'c', 'd', 'e']
List After Reassigning : ['a']
List After Deleting a Slice : ['a']
```

Clone or Copy a List

Using = (assignment operator)

Using the slicing technique

Using the copy( ) method

Using the deepcopy( ) method

[14]:
```
# use the = operator to copy a list.

old_list = [1, 2, 3]

# copy list using =
new_list = old_list

# add an element to list
new_list.append('a')
old_list.append('b')

'''Assignment statements do not copy objects, they create bindings between a␣
 ↪target and an object.
If you modify new_list, old_list is also modified.
It is because the new list is referencing or pointing to the same old_list␣
 ↪object.'''

print('New List:', new_list)
print('Old List:', old_list)
print(old_list is new_list)
```

```
New List: [1, 2, 3, 'a', 'b']
Old List: [1, 2, 3, 'a', 'b']
True
```

Deep copy stores copies of an absolute object's values, whereas a shallow copy stores references which are absolute to the original memory address in the runtime code.

Deep copy doesn't reflect any change made to the new/copied object in the original object; but shallow copy does the same in the run time code.

[15]:
```
# Shallow Copy using the copy

import copy

# mixed list
old_list = [1, 2, [3,5], 4]

# copying a list using copy
new_list = copy.copy(old_list)

# Adding an element to the new list
new_list[2][0] = 7
new_list.append(10)
old_list.append(12)
```

```
'''we don't find the new element that was appended. It's because it's still␣
 ↪pointing to the
old list reference. It won't have any information about the newly added element␣
 ↪done by after copy
command. Now, if we try to change the new list element that also exists in old␣
 ↪list,
you might see the change in old list b as well because it has all the␣
 ↪references of the old except
that, was added later.'''

# Printing new and old list
print('Old List:', old_list)
print('New List:', new_list)
print(old_list is new_list)
print(old_list[0] is new_list[0])
```

```
Old List: [1, 2, [7, 5], 4, 12]
New List: [1, 2, [7, 5], 4, 10]
False
True
```

[16]:
```
# Shallow Copy using the slicing syntax

# mixed list
old_list = [1, 2, [3,5], 4]

# copying a list using slicing
new_list = old_list[:]


# Adding an element to the new list
new_list[2][0] = 11
new_list.append(10)
old_list.append(12)

# Printing new and old list
print('Old List:', old_list)
print('New List:', new_list)
print(old_list is new_list)
print(old_list[0] is new_list[0])
```

```
Old List: [1, 2, [11, 5], 4, 12]
New List: [1, 2, [11, 5], 4, 10]
False
True
```

[17]:
```python
#Deep Copy

import copy

# mixed list
old_list = [[1,2,3],[4,5,6],[7,8,9]]

# copying a list using copy
new_list = copy.deepcopy(old_list)

# Adding an element to the new list
new_list[2][0] = 11
new_list.append(10)
old_list.append(12)
#any changes made to a copy of object do not reflect in the original object

# Printing new and old list
print('Old List:', old_list)
print('New List:', new_list)
print(old_list is new_list)
print(old_list[0] is new_list[0])
```

```
Old List: [[1, 2, 3], [4, 5, 6], [7, 8, 9], 12]
New List: [[1, 2, 3], [4, 5, 6], [11, 8, 9], 10]
False
False
```

Assignment vs Shallow Copy vs Deep Copy

Basic List Operations

Lists respond to the + and * operators much like strings; they mean concatenation and repetition here too

We can test if an item exists in a list or not, using the in , not in operators

Using a for loop we can iterate through each item in a list.

[18]:
```python
# concatenation operator
L = ['red', 'green', 'blue']
L = L + [1,2,3]
print(L)                  # Prints ['red', 'green', 'blue', 1, 2, 3]

# augmented assignment operator
L = ['red', 'green', 'blue']
L += [1,2,3]
print(L)                 # Prints ['red', 'green', 'blue', 1, 2, 3]
```

```
['red', 'green', 'blue', 1, 2, 3]
['red', 'green', 'blue', 1, 2, 3]
```

[19]:
```python
#Replication
L = ['red']
L = L * 3
print(L)                    # Prints ['red', 'red', 'red']
```

```
['red', 'red', 'red']
```

[20]:
```python
# Check for presence using membership operators
L = ['red', 'green', 'blue']

print('green' in L)
print('blue' not in L)

#With If Condition
if 'red' in L:
    print('yes')

if 'yellow' not in L:
    print('yes')
```

```
True
False
yes
yes
```

[21]:
```python
#Iterate through a List
L = ['red', 'green', 'blue']

for item in L:
    print(item)

# Loop through the list and double each item
squareList = [1, 2, 3, 4]
for i in range(len(squareList)):
    squareList[i] = squareList[i] * 2

print(squareList)
```

```
red
green
blue
[2, 4, 6, 8]
```

Python List Methods

append ( ) - Adds an item to the end of the list

insert ( ) - Inserts an item at a given position passed as an argument

extend ( ) - Extends the list by appending all the items from the iterable remove ( ) - Removes first instance of the specified item

pop ( ) - Removes the item at the given position in the list

clear ( ) - Removes all items from the list

copy ( ) - Returns a shallow copy of the list

index ( ) - returns the index of the first matched item

count ( ) - returns the count of the number of items passed as an argument

sort ( ) - sort items in a list in ascending order

reverse ( ) - reverse the order of items in the list

[22]:
```
# Example on Python list methods

my_list = [3, 8, 1, 6, 8, 8, 4]

# Index of first occurrence of 8
print(my_list.index(8))   # Output: 1

# Count of 8 in the list
print(my_list.count(8))  # Output: 3

#reverse the list
my_list.reverse()
print(my_list)
```

```
1
3
[4, 8, 8, 6, 1, 8, 3]
```

[23]:
```
#sorting the list of numbers
my_list.sort()
print(my_list)

#Sort with reverse
my_list.sort(reverse=True)
print(my_list)

#sorting the list of Strings
L = ['red', 'green', 'blue', 'orange']
L.sort()
print(L)

#sort with key
L.sort(key=len)
print(L)
```

```
[1, 3, 4, 6, 8, 8, 8]
[8, 8, 8, 6, 4, 3, 1]
['blue', 'green', 'orange', 'red']
['red', 'blue', 'green', 'orange']
```

Built-in Functions with List

all ( ) - Returns True if all list items are true

any ( ) - Returns True if any list item is true

len ( ) - Returns the number of items in the list

list ( ) - Converts an iterable (tuple, string, set etc.) to a list

enumerate ( ) - Takes a list and returns an enumerate object

max ( ) - Returns the largest item of the list

min ( ) - Returns the smallest item of the list

sorted ( ) - Returns a sorted list

sum ( ) - Sums items of the list

[24]:
```python
# Check if all items in a list are True

L = [1, 1, 1]
print(all(L))   # Prints True

L = [0, 1, 1]
print(all(L))   # Prints False

# Check if any item in a list is True

L = [0, 0, 0]
print(any(L))   # Prints False

L = [0, 1, 0]
print(any(L))   # Prints True
```

```
True
False
False
True
```

[25]:
```python
# Create a list that can be enumerated
L = ['red', 'green', 'blue']
x = list(enumerate(L))
print(x)
# Prints [(0, 'red'), (1, 'green'), (2, 'blue')]

#Iterate Enumerate Object
```

```
for pair in enumerate(L):
    print(pair)
# Prints (0, 'red')
# Prints (1, 'green')
# Prints (2, 'blue')
```

```
[(0, 'red'), (1, 'green'), (2, 'blue')]
(0, 'red')
(1, 'green')
(2, 'blue')
```

[26]:
```
#empty list
L = list()
print(L)
# Prints []

# string into list
T = list('abc')
print(T)
# Prints ['a', 'b', 'c']

# tuple into list
L = list((1, 2, 3))
print(L)
# Prints [1, 2, 3]

# sequence into list
L = list(range(0, 4))
print(L)
# Prints [0, 1, 2, 3]

# dictionary keys into list
L = list({'name': 'Bob', 'age': 25})
print(L)
# Prints ['age', 'name']

# set into list
L = list({1, 2, 3})
print(L)
# Prints [1, 2, 3]
```

```
[]
['a', 'b', 'c']
[1, 2, 3]
[0, 1, 2, 3]
['name', 'age']
[1, 2, 3]
```

```
[27]: #Find Maximum and Minimumin a list
L = [300, 500, 100, 400, 200,100]
print(max(L))    # Prints 500
print(min(L))  # Prints 100


L = ['red', 'green', 'blue', 'black', 'orange']
print(max(L, key=len))    #Prints orange
print(min(L, key=len))    #Prints red
```

```
500
100
orange
red
```

```
[1]: # strings are sorted alphabetically
L = ['red', 'green', 'blue', 'orange']
x = sorted(L)
print(x)
# Prints ['blue', 'green', 'orange', 'red']
'''If you want to sort the list in-place, use built-in sort() method.

sort() is actually faster than sorted() as it doesn't need to create a new list.
 ↪'''
# numbers are sorted numerically
L = [42, 99, 1, 12]
x = sorted(L)
print(x)
# Prints [1, 12, 42, 99]
```

```
['blue', 'green', 'orange', 'red']
[1, 12, 42, 99]
```

```
[29]: # Return the sum of all items in a list
L = [1, 2, 3, 4, 5]
x = sum(L)
print(x)
# Prints 15

# Start with '10' and add all items in a list
L = [1, 2, 3, 4, 5]
x = sum(L, 10)
print(x)
# Prints 25
```

```
15
25
```

Programming Examples

```
[30]: #Example: 1- Write the program to remove the duplicate element of the list.
list1 = [1,2,2,3,55,98,65,65,13,29]
# Declare an empty list that will store unique values
list2 = []
for i in list1:
    if i not in list2:
        list2.append(i)
print(list2)
```

```
[1, 2, 3, 55, 98, 65, 13, 29]
```

```
[31]: #Example:2- Write a program to find the sum of the element in the list.
list1 = [3,4,5,9,10,12,24]
sum = 0
for i in list1:
    sum = sum+i
print("The sum is:",sum)
```

```
The sum is: 67
```

```
[32]: #Example: 3- Write the program to find the lists consist of at least one common␣
 ↪element.
list1 = [1,2,3,4,5,6]
list2 = [7,8,9,2,10]
for x in list1:
    for y in list2:
        if x == y:
            print("The common element is:",x)
```

```
The common element is: 2
```

List Comprehension : Elegant way to create Lists

A comprehension is a compact way of creating a Python data structure from iterators.

With comprehensions, you can combine loops and conditional tests with a less verbose syntax.

```
[33]: #a list of all integer square numbers from 0 to 4 using for loop

L = []
for x in range(5):
    L.append(x**2)
print(L)
```

```
[0, 1, 4, 9, 16]
```

```
[34]: #a list comprehension would build the above list
L = [x**2 for x in range(5)]
print(L)
```

```
[0, 1, 4, 9, 16]
```

[35]:
```
#List comprehensions can iterate over any type of iterable such as lists,␣
↪strings, files, ranges,
# a simple list comprehension that uses string as an iterable

L = [x*3 for x in 'RED']
print(L)
```

```
['RRR', 'EEE', 'DDD']
```

[36]:
```
## Remove whitespaces of list items
colors = ['  red', '  green ', 'blue  ']
L = [color.strip() for color in colors]
print(L)
```

```
['red', 'green', 'blue']
```

[37]:
```
#Following example creates a list of (number, square) tuples
L = [(x, x**2) for x in range(4)]
print(L)
```

```
[(0, 0), (1, 1), (2, 4), (3, 9)]
```

[38]:
```
#list Comprehensions vs for loop --  list comprehensions are quite faster than␣
↪for loop.

# Import required module
import time


# define function to implement for loop
def for_loop(n):
        result = []
        for i in range(n):
                result.append(i**2)
        return result


# define function to implement list comprehension
def list_comprehension(n):
        return [i**2 for i in range(n)]


# Calculate time takens by for_loop()
begin = time.time()
for_loop(10**6)
end = time.time()

# Display time taken by for_loop()
```

```
print('Time taken for_loop:', round(end-begin, 2))

# Calculate time takens by list_comprehension()
begin = time.time()
list_comprehension(10**6)
end = time.time()

# Display time taken by for_loop()
print('Time taken for list_comprehension:', round(end-begin, 2))
```

```
Time taken for_loop: 0.32
Time taken for list_comprehension: 0.28
```

List Comprehension with if Clause

[39]:
```
# Filter list to exclude negative numbers
vec = [-4, -2, 0, 2, 4]
L = [x for x in vec if x >= 0]
print(L)
```

```
[0, 2, 4]
```

[40]:
```
#This list comprehension is the same as a for loop that contains an if␣
↪statement:
vec = [-4, -2, 0, 2, 4]
L = []
for x in vec:
    if x >= 0:
        L.append(x)
print(L)
```

```
[0, 2, 4]
```

[41]:
```
#Even and odd - if - else
lis = ["Even number" if i % 2 == 0 else "Odd number" for i in range(8)]
print(lis)
```

```
['Even number', 'Odd number', 'Even number', 'Odd number', 'Even number', 'Odd
number', 'Even number', 'Odd number']
```

[42]:
```
# Nested IF with List Comprehension
lis = [num for num in range(100) if num % 5 == 0 if num % 10 == 0]
print(lis)
```

```
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
```

Nested List Comprehension

[43]:
```
#simple list comprehension that flattens a nested list into a single list of␣
↪items.
vector = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
L = [number for vec in vector for number in vec]
print(L)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

[44]:
```
# equivalent to the following plain, old nested loop:
vector = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
L = []
for vec in vector:
    for number in vec:
        L.append(number)
print(L)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

[45]:
```
#another list comprehension that transposes rows and columns.
matrix = [[1, 2, 3],
          [4, 5, 6],
          [7, 8, 9]]
L = [[row[i] for row in matrix] for i in range(3)]
print(L)
```

```
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```

[52]:
```
## equivalent to the following nested for loop
transposed = []
for i in range(len(matrix[0])):
    transposed_row = []

    for row in matrix:
        transposed_row.append(row[i])
    transposed.append(transposed_row)

print(transposed)
```

```
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```

[49]:
```
#List Comprehension vs map() + lambda

# With list comprehension
L = [x ** 2 for x in range(5)]
print(L)
# Prints [0, 1, 4, 9, 16]

# With map() function
L = list(map((lambda x: x ** 2), range(5)))
print(L)
# Prints [0, 1, 4, 9, 16]
```

```
[0, 1, 4, 9, 16]
[0, 1, 4, 9, 16]
```

```
[50]: # using lambda along with list compreshion to print table of 10
numbers = list(map(lambda i: i*10, [i for i in range(1, 6)]))

print(numbers)
```

```
[10, 20, 30, 40, 50]
```

```
[51]: #List Comprehension vs filter() + lambda
#List comprehension with if clause can be thought of as analogous to the␣
 ↪filter() function

# With list comprehension
L = [x for x in range(10) if x % 2 == 0]
print(L)

# With filter() function
L = list(filter((lambda x: x % 2 == 0), range(10)))
print(L)
```

```
[0, 2, 4, 6, 8]
[0, 2, 4, 6, 8]
```

Key Points to Remember :

However, when we are working with large lists (e.g. 1 billion elements), list comprehension should be avoided.

It may cause your computer to crash due to the extreme amount of memory requirement.

A better alternative for such large lists is using a generator that does not actually create a large data structure in memory

Python Interview Questions on List Comprehension

What is a Python list comprehension?

How Python list comprehension is useful?

How does list comprehension work in Python?

Give an example of Python list comprehension

Are list comprehensions faster in Python?

To-Do:

Write a Python function to interchange first and last elements in a list

Write a Python function to swap two elements in a list

Write a Python function to Check if element exists in list

Write a Python function to Count occurrences of an element in a list

Write a Python function to count positive and negative numbers in a list

Write a Python function to Find sum and average of List without using sum()

Write a Python function to print and remove duplicates from a list of integers

Write a Python function that takes two lists and returns True if they have at least one common member

Write a Python function that returns a list that contains only the elements that are common between the two lists (without duplicates). Make sure your program works on two lists of different sizes.

Write a Python function to find largest , smallest , second largest number in a list

Write a Python function to Concatenate two list of lists Row-wise

Write a Python function to Reverse each string in tuple using list comprehension

Write a Python function to Square Each Odd Number in a List using List Comprehension

Write a Python function to Transpose of a Matrix using List Comprehension - Matrix = [[1, 2], [3,4], [5,6], [7,8]]

Write a Python function to Get the strings that end with the letter “b” and have a length greater than 2 using list comprehension