# List Comprehensions

## Mutable vs immutable sequences

- **Immutable** sequences: strings, tuples, and bytes.
- **Mutable** sequences:lists and byte arrays. Differ from their immutable in that they can be changed after creation.

## Lists recap

- Definition of list
- List of numbers (how we define it? Brackets, commas…)
- List of strings
- List of numbers and variables

## List - definition

- A single list may contain DataTypes like **Integers**, **Strings**, as well as **Objects**.
- Lists are mutable, and hence, they can be altered even after their creation.
- List in Python are ordered: The elements in a list are indexed according to a definite sequence and the indexing of a list is done with 0 being the first index. Each element in the list has its definite place in the list, which allows duplicating of elements in the list, with each element having its own distinct place and credibility.
- Lists can be created in many different ways, let's see an example:

```
>>> [] # empty list

>>> list() # same as []

>>> [1, 2, 3] # items are comma separated
```

## List methods

list.**append**(*x*) – Add an item to the end of the list;

list.**extend**(*L*) – Extend the list by appending all the items in the given list;

list.**insert**(*i*, *x*) – Insert an item at a given position. The first argument is the index of the element before which to insert, so `a.insert(0, x)` – inserts at the front of the list, and `a.insert(len(a), x)` is equivalent to `a.append(x)`.

list.**remove**(*x*) – Remove the first item from the list whose value is *x*. It is an error if there is no such item.

list.**pop**([*i*]) –Remove the item at the given position in the list, and return it. If no index is specified, `a.pop()` removes and returns the last item in the list.

list.**index**(*x*) –Return the index in the list of the first item whose value is *x*

list.**count**(*x*) –Return the number of times *x* appears in the list.

list.**sort**(*cmp=None*, *key=None*, *reverse=False*) – Sort the items of the list in place

list.**reverse**() –Reverse the elements of the list, in place.

## List methods - Example

```
>>> a = [1, 2, 1, 3]
>>> a.append(13)  # we can append anything at the end
>>> a
[1, 2, 1, 3, 13]
>>> a.count(1)  # how many `1` are there in the list?
2
>>> a.extend([5, 7])  # extend the list by another (or sequence)
>>> a
[1, 2, 1, 3, 13, 5, 7]
>>> a.index(13)  # position of `13` in the list (0-based indexing)
4
>>> a.insert(0, 17)  # insert `17` at position 0
>>> a
[17, 1, 2, 1, 3, 13, 5, 7]
>>> a.pop()  # pop (remove and return) last element
7
>>> a.pop(3)  # pop element at position 3
1
>>> a
[17, 1, 2, 3, 13, 5]
>>> a.remove(17)  # remove `17` from the list
>>> a
[1, 2, 3, 13, 5]
>>> a.reverse()  # reverse the order of the elements in the list
```

## List methods - Example

```
>>> a = list('hello')  # makes a list from a string
>>> a
['h', 'e', 'l', 'l', 'o']
>>> a.append(100)  # append 100, heterogeneous type
>>> a
['h', 'e', 'l', 'l', 'o', 100]
>>> a.extend((1, 2, 3))  # extend using tuple
>>> a
['h', 'e', 'l', 'l', 'o', 100, 1, 2, 3]
>>> a.extend('...')  # extend using string
>>> a
['h', 'e', 'l', 'l', 'o', 100, 1, 2, 3, '.', '.', '.']
```

```
>>> a.sort()  # sort the list
>>> a
[1, 2, 3, 5, 13]
>>> a.clear()  # remove all elements from the list
>>> a
[]
```

## List operations - Example

```
>>> a = [1, 3, 5, 7]
>>> min(a)  # minimum value in the list
1
>>> max(a)  # maximum value in the list
7
>>> sum(a)  # sum of all values in the list
16
>>> len(a)  # number of elements in the list
4
>>> b = [6, 7, 8]
>>> a + b  # `+` with list means concatenation
[1, 3, 5, 7, 6, 7, 8]
>>> a * 2  # `*` has also a special meaning
[1, 3, 5, 7, 1, 3, 5, 7]
```

## Lists recap

Access some elements

> l = [1, 2, 3, 4]

> l[1] if I do it what will I take?

> l[-1] if I do it what will I take?

> l[2:4] if I do it what will I take?

> l.append(´Hola’) Can I do this?

> l.remove(‘3’)

## Lists recap

- sum(l) Can I do it?
- l.remove(´Hola’)
- sum(l)

- sort(l)
- sort(l, reverse=True)

## Lists

I don’t want to write

```
> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

How can I write it with a loop?

## List comprehensions

- List comprehensions provide a concise way to create lists.
- It consists of brackets containing an expression followed by a for clause, then zero or more for or if clauses. The expressions can be anything, meaning you can put in all kinds of objects in lists.

```
new_list = [expression(i) for i in old_list if filter(i)]
```

```
lst = [i for i in range(10)]
print(lst)
```

```
lst = [i+1 for i in range(10)]
print(lst)
```

What will be the output?

## Conditions to lists

```
lst = [i for i in range(10) if i >= 5]
print(lst)

lst = [i for i in range(20) if i >= 5]
print(lst)
```

## Nested loops

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

lst = []

for x in lst_lst:
    for y in x:
        lst.append(y)

print(lst)

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

## Nested loops

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

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

Do it your own with list comprehension!

```
lst = [y for x in lst_lst for y in x]
print(lst)
```

## Nested loops with conditions

```
lst_lst = [[1,2,3,4,5], [6,7,8], [9,10]]

lst = [y for x in lst_lst if len(x) < 4 for y in x if y % 2 == 0]
print(lst)
```

Do you understand something? Me neither. Sometimes it is better to keep the simple syntax. Also, nested loops are not a good idea generally.