Python Programming Language

Prepared by: Mohamed Ayman

Algorithm Engineer at Valeo

Deep Learning Researcher and Teaching Assistant

at The American University in Cairo (AUC)

spring 2020

sw.eng.MohamedAyman@gmail.com

linkedin.com/in/cs-MohamedAyman

github.com/cs-MohamedAyman

codeforces.com/profile/Mohamed_Ayman

Lecture 8

Lists

Course Roadmap

Part 2: Python Collections and Strings

Lecture 7: Strings

Lecture 8: Lists

Lecture 9: Tuples

Lecture 10: Dictionaries

Lecture 11: Sets

Lecture 12: Numbers

Lecture Agenda

We will discuss in this lecture the following topics

1- Introduction to List

2- Basic List Operations

3- List Comprehension

4- Multi-dimensional Lists

5- Built-in List Functions


Let's
START UP

Lecture Agenda

Section 1: Introduction to List

Section 2: Basic List Operations

Section 3: List Comprehension

Section 4: Multi-dimensional Lists

Section 5: Built-in List Functions

Introduction to List

- The List is the most versatile data type available in Python, which can be written as a list of comma-separated value between square brackets. Important thing about a list is that the items in a list need not be of the same type. In Python programming, a list is created by placing all the items (elements) inside a square bracket [ ], separated by commas. It can have any number of items and they may be of different types (integer, float, string etc.).

- List is an ordered sequence of items. It is one of the most used data type in Python and is very flexible. All the items in a list do not need to be of the same type, Declaring a list is pretty straight forward. Items separated by commas are enclosed within brackets [ ]. Lists need not be homogeneous always which makes it a most powerful tool in Python. 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.

```
x = [2, 'python', -3.7]
y = x
print(x)                    [2, 'python', -3.7]
print(y)                    [2, 'python', -3.7]
x[0] = -8
y[1] = 'c++'
print(x)                    [-8, 'c++', -3.7]
print(y)                    [-8, 'c++', -3.7]
```

Accessing Values in Lists

- To access value in lists, use the square brackets for slicing along with the index of indices to obtain value available at that index. Since lists are sequences, indexing and slicing work the same way for lists as they do for strings. We can use the index operator [] to access an item in a list. Index starts from 0. So, a list having 5 elements will have index from 0 to 4.
- Trying to access an element other that this will raise an IndexError. The index must be an integer. We can't use float or other types, this will result into TypeError. Nested list are accessed using nested indexing.
- Python allows negative indexing for its sequences. The index of -1 refers to the last item, -2 to the second last item and so on. We can access a range of items in a list by using the slicing operator (colon). Slicing can be best visualized by considering the index to be between the elements as shown below. So if we want to access a range, we need two index that will slice that portion from the list.

Accessing Values in Lists

Example:

```
x = ['c++', 123, 'abcd', 2.3, 'python']
print(x)
print(len(x))
print(x[3])
print(x[2:4])
print(x[:4])
print(x[2:])
print(x[-1])
print(x[-4:])
print(x[:-3])
```

Output:

```
['c++', 123, 'abcd', 2.3, 'python']
5
2.3
['abcd', 2.3]
['c++', 123, 'abcd', 2.3]
['abcd', 2.3, 'python']
'python'
[123, 'abcd', 2.3, 'python']
['c++', 123]
```

```
'c++', 123, 'abcd', 2.3, 'python'
0      1     2      3     4
-5     -4    -3     -2    -1
```

Python Lists

Example:

```
x = ['python', 2000, 'c++', 3.2, 'java']
print(x)
print(len(x))
print(x[::4])
print(x[::3])
print(x[::2])
print(x[::1])
print(x[::-1])
print(x[::-2])
print(x[::-3])
print(x[::-4])
```

```
'python', 2000, 'c++', 3.2, 'java'
0         1     2      3     4
-5        -4    -3     -2    -1
```

Output:

```
['python', 2000, 'c++', 3.2, 'java']
5
['python', 'java']
['python', 3.2]
['python', 'c++', 'java']
['python', 2000, 'c++', 3.2, 'java']
['java', 3.2, 'c++', 2000, 'python']
['java', 'c++', 'python']
['java', 2000]
['java', 'python']
```

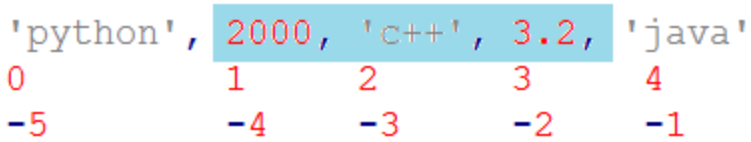
Python Lists

Example:

```
x = ['python', 2000, 'c++', 3.2, 'java']
print(x)
print(len(x))
print(x[1:4:4])
print(x[1:4:3])
print(x[1:4:2])
print(x[1:4:1])
print(x[-2:0:-4])
print(x[-2:0:-3])
print(x[-2:0:-2])
print(x[-2:0:-1])
```

Output:

```
['python', 2000, 'c++', 3.2, 'java']
5
[2000]
[2000]
[2000, 3.2]
[2000, 'c++', 3.2]
[3.2]
[3.2]
[3.2, 2000]
[3.2, 'c++', 2000]
```



Lecture Agenda

- ✔ Section 1: Introduction to List
- **Section 2: Basic List Operations**
- Section 3: List Comprehension
- Section 4: Multi-dimensional Lists
- Section 5: Built-in List Functions

Python Lists

- List is an ordered sequence of items. It is one of the most used data type in Python and is very flexible. All the items in a list do not need to be of the same type, Declaring a list is pretty straight forward.
- Items separated by commas are enclosed within brackets [ ]. To some extent, lists are similar to arrays in C. One of the differences between them is that all the items belonging to a list can be of different data types.
- The value stored in a list can be accessed using the slice operator ([] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.
- Lists need not be homogeneous always which makes it a most powerful tool in Python. 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.

Python Lists

Example:

```
x = ['python', 2000, 'c++', 3.2, 'java']
print(x)
print(len(x))
print(x[1] + x[3])
print(x[0] + x[-1])
print(x[:2] + x[3:])
print(x[2] * 3)
print(x[1] * 3)
print(x[2:4] * 2)
print((x[1:3] + x[4:5]) * 2)
```

Output:

```
['python', 2000, 'c++', 3.2, 'java']
5
2003.2
pythonjava
['python', 2000, 3.2, 'java']
c++c++c++
6000
['c++', 3.2, 'c++', 3.2]
[2000, 'c++', 'java', 2000, 'c++', 'java']
```

```
'python', 2000, 'c++', 3.2, 'java'
0         1     2      3     4
-5        -4    -3     -2    -1
```

Updating Lists

- You can update single or multiple elements of lists by giving the slice on the left-hand side of the assignment operator. When you assign list to other list, these lists will be have a shared address in memory. List are mutable, meaning, their elements can be changed. We can use assignment operator (=) to change an item or a range of items. We can add one item to a list using append() method or add several items using extend() method. Furthermore, we can insert one item at a desired location by using the method insert() or insert multiple items by squeezing it into an empty slice of a list.

- To remove a list elements, you can use either the del statement if you know exactly which element(s) you can deleting. You can use the remove() method if you do not know exactly which item to delete. We can delete one or more items from a list using the keyword del. It can even delete the list entirely. We can use remove() method to remove the given item or pop() method to remove an item at the given index. The pop() method removes and returns the last item if index is not provided. This helps us implement lists as stacks (first in, last out data structure). We can also use the clear() method to empty a list.

Updating Lists

Example:

```
x = [3, 'abc', 5.2, 6, 1.7, 'ijk']
print(x)
x[5:] = ['def']
print(x)
x[:2] = [8.2, 7]
print(x)
x[4:5] = ['were', 9.4, 'the']
print(x)
x[3:7] = ['you']
print(x)
x = x[:2] + x[-2:]
print(x)
x[2] = 1.1
print(x)
```

Output:

```
[3, 'abc', 5.2, 6, 1.7, 'ijk']

[3, 'abc', 5.2, 6, 1.7, 'def']

[8.2, 7, 5.2, 6, 1.7, 'def']

[8.2, 7, 5.2, 6, 'were', 9.4, 'the', 'def']

[8.2, 7, 5.2, 'you', 'def']

[8.2, 7, 'you', 'def']

[8.2, 7, 1.1, 'def']
```

Updating Lists

Example:

```
x = ['p', 'y', 't', 'h', 'o', 'n']
print(x)
del x[2]
print(x)
del x[3:5]
print(x)
x[1:3] = []
print(x)
del x
print(x)
```

Output:

```
['p', 'y', 't', 'h', 'o', 'n']

['p', 'y', 'h', 'o', 'n']

['p', 'y', 'h']

['p']

Traceback (most recent call last):
  File "main.py", line 10, in <module>
    print(x)
NameError: name 'x' is not defined
```

Lecture Agenda

- ✔ Section 1: Introduction to List
- ✔ Section 2: Basic List Operations
- **Section 3: List Comprehension**
- Section 4: Multi-dimensional Lists
- Section 5: Built-in List Functions

List Comprehension

- Using generator comprehensions to initialize lists is so useful that Python actually reserves a specialized syntax for it, known as the list comprehension. A list comprehension is a syntax for constructing a list, which exactly mirrors the generator comprehension syntax:

```
[<expression> for <var> in <iterable> {if <condition>}]
```

- For example, if we want to create a list of square-numbers, we can simply write:

```
x = [i**2 for i in range(7)]
print(x)
```

```
[0, 1, 4, 9, 16, 25, 36]
```

- This produces the exact same result as feeding the list function a generator comprehension. However, using a list comprehension is slightly more efficient than is feeding the list function a generator comprehension.

List Comprehension

- Generate a list with zeros

```
z = [0 for i in range(6)]
print(z)                                    [0, 0, 0, 0, 0, 0]
z = [0] * 6
print(z)                                    [0, 0, 0, 0, 0, 0]
```

- Let's appreciate how economical list comprehensions are.
- The following code stores words that contain the letter 'o', in a list:

```
words = ['python', 'like', 'you', 'mean', 'it']
res = []
for i in words:
    if 'o' in i:
        res += [i]
```

- This can be written in a single line, using a list comprehension:

```
words = ['python', 'like', 'you', 'mean', 'it']
res = [i for i in words if 'o' in i]
print(res)                                  ['python', 'you']
```

List Comprehension

Example:

```
# Input list
x = [int(i) for i in input().split()]
print(x)
x = list(map(int, input().split()))
print(x)

# Copy the values of x in y
y = [i for i in x]
y = x[:]

# Updating list x and y
x[0] = 2.7
y[1] = 3.2
print(x)
print(y)

# Reverse x, y
print(x[::-1])
print(y[::-1])
```

Output:

```
[7, 8, 9, 6, 5, 4]
[7, 8, 9, 6, 5, 4]

[2.7, 8, 9, 6, 5, 4]
[7, 3.2, 9, 6, 5, 4]

[4, 5, 6, 9, 8, 2.7]
[4, 5, 6, 9, 3.2, 7]
```

Lecture Agenda

- ✔ Section 1: Introduction to List
- ✔ Section 2: Basic List Operations
- ✔ Section 3: List Comprehension
- **Section 4: Multi-dimensional Lists**
- Section 5: Built-in List Functions

Multi-dimensional Lists

- A matrix is a two-dimensional data structure. In real-world tasks you often have to store rectangular data table. The table below shows the marks of three students in different subjects.
- Such tables are called matrices or two-dimensional arrays. In python any table can be represented as a list of lists (a list, where each element is in turn a list).

S.No	Student Name	Science	English	History	Arts	Maths
1	Roy	80	75	85	90	95
2	John	75	80	75	85	100
3	Dave	80	80	80	90	95

- In the above example A represents a 3*6 matrix where 3 is number of rows and 6 is number of columns.

```
A = [['Roy',80,75,85,90,95],['John',75,80,75,85,100],['Dave',80,80,80,90,95]]
```

Multi-dimensional Lists

- In python, matrix is a nested list. A list is created by placing all the items (elements) inside a square bracket [ ], separated by commas.
- Here's a program that creates a numerical table with 3 rows and 5 columns.
- In the second examples a is a matrix as well as nested list where as b is a nested list but not a matrix.

```
# X is 2-D matrix
x = [['Roy', 80, 75, 85, 90],
     ['John', 70, 80, 75, 85],
     ['Dave', 85, 70, 80, 90]]

print(x)

# Y is nested list but not 2-D matrix
y = [['Roy', 80, 75, 85, 90],
     ['John', 70, 80, 75],
     ['Dave', 85, 70, 80, 90]]

print(y)
```

Multi-dimensional Lists

- Create a dynamic matrix using for loop. A possible way: you can create a matrix of n*m elements by first creating a list of n elements (say, of n zeros) and then make each of the elements a link to another one-dimensional list of m elements

Example

```
n, m = 3, 4
x = [[0]*m for i in range(n)]
print(x)
```

Output

```
[[0, 0, 0, 0],
 [0, 0, 0, 0],
 [0, 0, 0, 0]]
```

```
n, m = 3, 4
x = []
for i in range(n):
    x += [list(map(int, input().split()))]
print(x)
```

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

Quiz

- Which of the following expressions evaluates to the list [1,2,3,4,5]?

- ☐ `list(range(1,6,1))`
- ☐ `list(range(1,6))`
- ☐ `range(1,6)`
- ☐ `list(range(5))`

- If we want to split a list `my_list` into two halves, which of the following uses slices to do so correctly?

  More precisely, if the length of `my_list` is 2n, i.e., even, then the two parts should each have length n. If its length is 2n+1, i.e., odd, then the two parts should have lengths n and n+1.

- ☐ `x[:len(x)//2] and x[len(x)//2:]`
- ☐ `x[0:len(x)//2-1] and x[len(x)//2:len(x)]`
- ☐ `x[0:len(x)//2] and x[len(x)//2+1:len(x)]`
- ☐ `x[0:len(x)//2] and x[len(x)//2:len(x)]`

Quiz Solution

- Which of the following expressions evaluates to the list [1,2,3,4,5]?

- [x] `list(range(1,6,1))`
- [x] `list(range(1,6))`
- [ ] `range(1,6)`
- [ ] `list(range(5))`

- If we want to split a list `my_list` into two halves, which of the following uses slices to do so correctly?

  More precisely, if the length of `my_list` is 2n, i.e., even, then the two parts should each have length n. If its length is 2n+1, i.e., odd, then the two parts should have lengths n and n+1.

- [x] `x[:len(x)//2] and x[len(x)//2:]`
- [ ] `x[0:len(x)//2-1] and x[len(x)//2:len(x)]`
- [ ] `x[0:len(x)//2] and x[len(x)//2+1:len(x)]`
- [x] `x[0:len(x)//2] and x[len(x)//2:len(x)]`

Quiz

- If n and m are non-negative integers, consider the list final_list computed by the code snippet below.

```
init_list = list(range(1, n))
final_list = init_list * m
```

The length of this list depends on the particular values of n and m used in computation. Which option below correctly expresses the length of final_list in terms of n and m?

- ☐ `n × m`
- ☐ `n × (m - 1)`
- ☐ `n + m`
- ☐ `(n - 1) × m`

- If n is a non-negative integer, consider the list split_list computed by the code snippet below.

```
test_string = 'xxx' + ' ' * n + 'xxx'
split_list = test_string.split(' ')
```

The length of this list depends on the particular values of n used in computation. Which option below correctly expresses the length of split_list in terms of n?

- ☐ `3`
- ☐ `n + 1`
- ☐ `2`
- ☐ `n`

Quiz Solution

- If n and m are non-negative integers, consider the list final_list computed by the code snippet below.

```
init_list = list(range(1, n))
final_list = init_list * m
```

The length of this list depends on the particular values of n and m used in computation. Which option below correctly expresses the length of final_list in terms of n and m?

- ☐ n × m
- ☐ n × (m − 1)
- ☐ n + m
- ☑ (n − 1) × m

- If n is a non-negative integer, consider the list split_list computed by the code snippet below.

```
test_string = 'xxx' + ' ' * n + 'xxx'
split_list = test_string.split(' ')
```

The length of this list depends on the particular values of n used in computation. Which option below correctly expresses the length of split_list in terms of n?

- ☐ 3
- ☑ n + 1
- ☐ 2
- ☐ n

Lecture Agenda

- ✔ Section 1: Introduction to List
- ✔ Section 2: Basic List Operations
- ✔ Section 3: List Comprehension
- ✔ Section 4: Multi-dimensional Lists
- Section 5: Built-in List Functions

Built-in List Functions

1- len() Method
2- list() Method
3- max(), min() Methods
4- append() Method
5- insert() Method
6- pop() Method
7- remove() Method
8- count() Method
9- index() Method
10- extend() Method
11- sort() Method
12- reverse() Method
13- sum() Method
14- clear() Method
15- copy() Method
16- all(), any() Methods
17- enumerate() Method
18- map(), filter() Methods

len() Method

Example:

```
x = [5, 8, 2, 9, 10, 4]
print(x)
print(len(x))

x = ['python', 'c++', 'java']
print(x)
print(len(x))
```

Output:

```
[5, 8, 2, 9, 10, 4]
6

['python', 'c++', 'java']
3
```

list() Method

Example:

```
x = 'python'
print(x)
y = list(x)
print(y)

x = (5, 16, 21, 4)
print(x)
y = list(x)
print(y)

x = {4, 8, 9, 3, 1}
print(x)
y = list(x)
print(y)

x = {'b':4, 'c':8, 'a':9}
print(x)
y = list(x)
print(y)
```

Output:

```
python

['p', 'y', 't', 'h', 'o', 'n']

(5, 16, 21, 4)

[5, 16, 21, 4]

{1, 3, 4, 8, 9}

[1, 3, 4, 8, 9]

{'b':4, 'c':8, 'a':9}

['b', 'c', 'a']
```

max(), min() Methods

Example:

```
x = [1, 2, 6]
print(x)
print(max(x))
print(min(x))
y = [4, 2]
print(y)
print(max(y))
print(min(y))
print(max(x, y))
print(min(x, y))
```

Output:

```
[1, 2, 6]
6
1
[4, 2]
4
2
[4, 2]
[1, 2, 6]
```

max(), min() Methods

Example:

```
x = ['php', 'c++', 'java']
print(x)
print(max(x))
print(min(x))
y = ['python', 'c#']
print(y)
print(max(y))
print(min(y))
print(max(x, y))
print(min(x, y))
```

Output:

```
['php', 'c++', 'java']
php
c++

['python', 'c#']
python
c#
['python', 'c#']
['php', 'c++', 'java']
```

append() Method

Example:

```
x = ['php', 'c++', 'java']
print(x)
x.append('python')
print(x)
x.append('c#')
print(x)
```

Output:

```
['php', 'c++', 'java']
['php', 'c++', 'java', 'python']
['php', 'c++', 'java', 'python', 'c#']
```

insert() Method

Example:

```
x = ['php', 'c++', 'java']
print(x)
x.insert(0, 'c#')
print(x)
x.insert(3, 'math')
print(x)
x.insert(len(x), 'c')
print(x)
x.insert(-2, 'R')
print(x)
```

Output:

```
['php', 'c++', 'java']
['c#', 'php', 'c++', 'java']
['c#', 'php', 'c++', 'math', 'java']
['c#', 'php', 'c++', 'math', 'java', 'c']
['c#', 'php', 'c++', 'math', 'R', 'java', 'c']
```

pop() Method

Example:

```
x = ['c#', 'php', 'c++', 'math', 'R', 'c']
print(x)
x.pop()
print(x)
x.pop(0)
print(x)
x.pop(2)
print(x)
x.pop(-2)
print(x)
```

Output:

```
['c#', 'php', 'c++', 'math', 'R', 'c']
['c#', 'php', 'c++', 'math', 'R']
['php', 'c++', 'math', 'R']
['php', 'c++', 'R']
['php', 'R']
```

remove() Method

Example:

```
x = ['c#', 'R', 'c#', 'R', 'c++']
print(x)
x.remove('c++')
print(x)
x.remove('c#')
print(x)
x.remove('python')
print(x)
```

Output:

```
['c#', 'R', 'c#', 'R', 'c++']

['c#', 'R', 'c#', 'R']

['R', 'c#', 'R']

Traceback (most recent call last):
  File "main.py", line 7, in <module>
    x.remove('python')
ValueError: list.remove(x): x not in list
```

count() Method

Example:

```
x = ['c#', 'R', 'c#', 'R', 'c++']
print(x)
print(x.count('R'))
print(x.count('c#'))
print(x.count('c++'))
print(x.count('python'))
```

Output:

```
['c#', 'R', 'c#', 'R', 'c++']
2
2
1
0
```

index() Method

Example:

```
x = ['c#', 'c++', 'c#', 'python']
print(x)
print(x.index('c#'))
print(x.index('c++'))
print(x.index('java'))
```

Output:

```
['c#', 'c++', 'c#', 'python']
0
1
Traceback (most recent call last):
  File "main.py", line 5, in <module>
    print(x.index('java'))
ValueError: 'java' is not in list
```

Practice

Practice Problems

1- Implement a function which count the number of items in a list
2- Implement a function which converts any collections to list
3- Implement a function which finds the maximum value in a list
4- Implement a function which finds the minimum value in a list
5- Implement a function which adds a new item in the end of the list
6- Implement a function which adds a new item in the list in giving valid index (positive or negative)
7- Implement a function which deletes an item in the list in giving valid index (positive or negative)
8- Implement a function which deletes an item in the list by a giving value
9- Implement a function which deletes all items in the list by a giving value
10- Implement a function which counts the number of occurrences of a given item in a list
11- Implement a function which concatenates two lists in one list
12- Implement a function which finds the maximum element and its index in a given list
13- Implement a function which finds the minimum element and its index in a given list
14- Implement a function which removes all appearance of a given element in a given list
15- Implement a function which removes all appearance of a given element in a given list

Built-in List Functions

1- len() Method

2- list() Method

3- max(), min() Methods

4- append() Method

5- insert() Method

6- pop() Method

7- remove() Method

8- count() Method

9- index() Method

10- extend() Method

11- sort() Method

12- reverse() Method

13- sum() Method

14- clear() Method

15- copy() Method

16- all(), any() Methods

17- enumerate() Method

18- map(), filter() Methods

extend() Method

Example:

```
x = [1, 2, 6]
y = ['python', 'c++', 'java']
print(x)
print(y)
x.extend(y)
print(x)
print(y)
```

Output:

```
[1, 2, 6]
['python', 'c++', 'java']

[1, 2, 6, 'python', 'c++', 'java']
['python', 'c++', 'java']
```

sort() Method

Example:

```
x = ['python', 'java', 'c#', 'c++']
print(x)
x.sort()
print(x)

x = [4, 6, 1, 9, 7, 3]
print(x)
x.sort()
print(x)
```

Output:

```
['python', 'java', 'c#', 'c++']
['c#', 'c++', 'java', 'python']

[4, 6, 1, 9, 7, 3]
[1, 3, 4, 6, 7, 9]
```

reverse() Methods

Example:

```
x = ['python', 'java', 'c#', 'c++']
print(x)
x.reverse()
print(x)

x = [4, 6, 1, 9, 7, 3]
print(x)
x.reverse()
print(x)
```

Output:

```
['python', 'java', 'c#', 'c++']
['c++', 'c#', 'java', 'python']

[4, 6, 1, 9, 7, 3]
[3, 7, 9, 1, 6, 4]
```

sum() Method

Example:

```
x = [4, 6, 1, 9, 7, 3]
print(x)
print(sum(x))

x = ['python', 'java', 'c#', 'c++']
print(x)
print(sum(x))
```

Output:

```
[4, 6, 1, 9, 7, 3]
30

['python', 'java', 'c#', 'c++']
Traceback (most recent call last):
  File "main.py", line 7, in <module>
    print(sum(x))
TypeError: unsupported operand type(s)
for +: 'int' and 'str'
```

clear() Method

Example:

```
x = [4, 6, 1, 9, 7, 3]
print(x)
print(len(x))
x.clear()
print(x)
print(len(x))
```

Output:

```
[4, 6, 1, 9, 7, 3]
6
[]
0
```

copy() Method

Example:

```
x = [25, 45, 35, 15]
y = x.copy()
print(x)
print(y)

x[1] = 55
y[0] = 65
print(x)
print(y)
```

Output:

```
[25, 45, 35, 15]
[25, 45, 35, 15]

[25, 55, 35, 15]
[65, 45, 35, 15]
```

all(), any() Methods

Example:

```
x = [True, False, False]
print(all(x))
print(any(x))

x = [True, True, False]
print(all(x))
print(any(x))

x = [False, False, False]
print(all(x))
print(any(x))

x = [True, True, True]
print(all(x))
print(any(x))
```

Output:

```
False
True

False
True

False
False

True
True
```

enumerate() Method

Example:

```
x = [25, 45, 35, 15]
print(x)

e = enumerate(x)
print(type(e))
print(list(e))

e = enumerate(x, 3)
print(type(e))
print(list(e))
```

Output:

```
[25, 45, 35, 15]

<class 'enumerate'>
[(0, 25), (1, 45), (2, 35), (3, 15)]

<class 'enumerate'>
[(3, 25), (4, 45), (5, 35), (6, 15)]
```

enumerate() Method

Example:

```
x = [25, 45, 35, 15]
print(x)

for i in enumerate(x):
    print(i)


for i, j in enumerate(x):
    print(i, j)
```

Output:

```
[25, 45, 35, 15]

(0, 25)
(1, 45)
(2, 35)
(3, 15)
0 25
1 45
2 35
3 15
```

map(), filter() Methods

Example:

```
# Program to filter out only the even items from a list
def filter_fun(x):
    return x % 2 == 0

x = [1, 4, 6, 5, 8]
r = list(filter(filter_fun, x))
print(r)

# Program to double each item in a list using map()
def map_fun(x):
    return x * 2

x = [1, 5, 4, 6, 8]
r = list(map(map_fun, x))
print(r)
```

Output:

```
[4, 6, 8]

[2, 10, 8, 12, 16]
```

map(), filter() Methods

Example:

```
# Program to filter out only the even items from a list

x = [1, 4, 6, 5, 8]
r = list(filter(lambda x: (x%2 == 0) , x))
print(r)

# Program to double each item in a list using map()

x = [1, 5, 4, 6, 8]
r = list(map(lambda x: x * 2 , x))
print(r)
```

Output:

```
[4, 6, 8]

[2, 10, 8, 12, 16]
```

Practice

Practice Problems

1- Implement a function which finds the index of a given item in a list
2- Implement a function which reverse a given list
3- Implement a function which calculate the sum a given list with type Number
4- Implement a function which calculate the sum a given list with type String
5- Implement a function which create a copy of a given list
6- Implement a function which takes a list as a parameter, then return a list of tuples (index, item)
7- Implement a function which transpose a matrix
8- Implement a function which calculate the addition of two metrics
9- Implement a function which calculate the subtraction of two metrics
10- Implement a function which calculate the multiplication of two metrics
11- Implement a function which calculate the multiplication of scalar with matrix
12- Implement a function which return two lists of the indices for positive and negative numbers in a given list
13- Implement a function which return two lists of positive and negative numbers in a given list
14- Implement a function which checks if all elements in a given list are even or not
15- Implement a function which sort a given list in non-ascending order

Built-in List Functions

1- len() Method

2- list() Method

3- max(), min() Methods

4- append() Method

5- insert() Method

6- pop() Method

7- remove() Method

8- count() Method

9- index() Method

10- extend() Method

11- sort() Method

12- reverse() Method

13- sum() Method

14- clear() Method

15- copy() Method

16- all(), any() Methods

17- enumerate() Method

18- map(), filter() Methods

Lecture Agenda

- ✔ Section 1: Introduction to List
- ✔ Section 2: Basic List Operations
- ✔ Section 3: List Comprehension
- ✔ Section 4: Multi-dimensional Lists
- ✔ Section 5: Built-in List Functions


DO
MORE.