M110: Python Programming

Meetings #5

Collection Data Types
Lists, Tuples

AOU
الجامعة العربية المفتوحة
Arab Open University
Lebanon

Prepared by Dr. Ahmad Mikati

Contents

- Collections
- Introduction to lists
- List Slicing
- Finding items in lists with the **in** operator
- List Methods and useful built-in Functions
- Two-dimensional lists
- Introduction to Tuples
- Tuple Methods and useful built-in Functions
- Converting Between Lists and Tuples
- Adding Items to a Tuple
- Changing Tuple Values

AOU
الجامعة العربية المفتوحة
Arab Open University
Lebanon

Introduction

- Python programming language has four collection data types- list, tuple, sets and dictionary. They have different characteristics based on the declaration and the usage.

- A sequence is an object that contains multiple items of data. The items that are in a sequence are stored one after the other.
- Python provides various ways to perform operations on the items that are stored in a sequence.
- There are several different types of sequence objects in Python.

- In this lecture, we will look at two of the fundamental sequence types: lists and tuples.
- Both lists and tuples are sequences that can hold various types of data.

Lists

- A list is an object that contains multiple data items.
- Lists are mutable, which means that their contents can be changed during a program's execution.
- Lists are dynamic data structures, meaning that items may be added to them or removed from them.
- You can use indexing, slicing, and various methods to work with lists in a program.

Lists

- A **list** is a data structure in Python that is a mutable (or changeable), ordered sequence of elements.
- Each element or value that is inside of a list is called an **item**.
- Lists are defined by having values between square brackets [ ].
- Lists are great to use when you want to work with many related values.
- They enable you to keep data together that belongs together, condense your code, and perform the same methods and operations on multiple values at once.
- To create a list, we use square brackets to indicate the start and end of the list and separate the items by commas.

Here is a statement that creates a list of integers:

```
List1= [1,2,3,4,5]
```

Lists

List1= [1,2,3,4,5]

- You can use the **print** function to print the entire contents of a list:

```
print(List1)  #[1, 2, 3,4,5] will be printed
```

- Lists are mutable: individual elements can be reassigned in place.

```
List1[0] = 6
print(List1)  #[6, 2, 3,4,5] will be printed
```

- The empty list is []. It is the list equivalent of 0 or empty string ‘’.

If you have a long list to enter, you can split it across several lines, like below:

```
nums = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
31, 32, 33, 34, 35, 36, 37, 38, 39, 40]
```

Lists

- An example of a list with three integers is `[4, 2, -9]`.

Lists can be assigned to variables, e.g., **`temperatures= [4, 2,-9]`**. The variable name should reflect the content of the list to make the program easier to understand.

There are several things which work the same way for lists as for strings.

- **len():** where you can use len(L) to know the number of items in a list L.

Example

- **in:** operator which tells you if a list contains something.

`len(temperatures)` computes the length of temperatures (3)

```
if 2 in temperatures:
    print('Your list contains number 2.')
if 0 not in temperatures:
    print('Your list has no zeroes.')
```

Lists

Iterating over a List with the for Loop

You can also iterate over a list with the for loop as follows:

1- iterate-by-item

2- iterate-by-position (index)

1- iterate-by-item:

The for loop is used to iterate over the items of any sequence using **in** operator

2- iterate-by-position:

The for loop is also used to access elements from a sequence via their position using built-in function **range()**.

Example: Both of the following examples print out the items of a list

iterate-by-item

```
L=[6,7,8,9,10]
for i in L:
    print(i, end=' ')
```

6 7 8 9 10

iterate-by-position

```
L=[6,7,8,9,10]
for i in range(5):
    print(L[i], end=' ')
```

6 7 8 9 10

Lists

The Repetition Operator

The repetition operator makes multiple copies of a list and joins them all together.

Here is the general format: ***list * n***

In the general format, *list* is a list, and *n* is the number of copies to make.

Example:

```
List1= [1,2,3,4,5]
List2=List1*2
print(List2)
```

```
[1, 2, 3, 4, 5, 1, 2, 3, 4,
5]
```

Lists

Concatenating Lists

To concatenate means to join two things together. You can use the **+** operator to concatenate two lists.

Example:

```
list1 = [1, 2, 3, 4]
list2 = [5, 6, 7, 8]
list3 = list1 + list2
print(list3)
```

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

You can also use the **+=** augmented assignment operator to concatenate one list to another.

Example:

```
list1 = [1, 2, 3,
4]
list2 = [5, 6, 7,
8]
list
pri
```

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

N.B: If you try to concatenate a list with something that is ***not a list***, an exception will be raised.

AOU
الجامعة العربية المفتوحة
Arab Open University
Lebanon

Lists

Indexing and slicing work exactly as with strings: L=[5,7,9, 8,10]

Indexing: To retrieve an element of the list, we use the index operator ([]):

L[0] is the first item of the list L. ➔5

Sometimes, you want to select more than one element from a sequence.

In Python, you can write expressions that select subsections of a sequence, known as **slices**.

Slicing: A slice is a subset of list elements that are taken from a sequence. When you take a slice from a list, you get a span of elements from within the list.

To get a slice of a list, write the following general format:

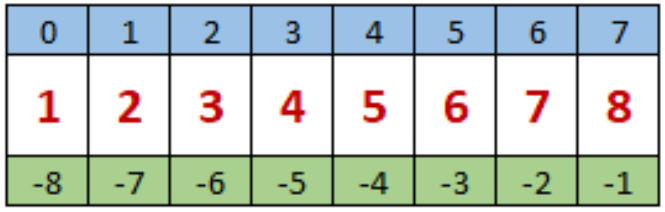
list_name[start : stop : step]

The expression returns a list containing a copy of the elements from *start* up to (but not including) *stop*. The *step* is an integer number specifying the step of the slicing.

Lists

Examples:

```
list1 = [1, 2, 3, 4,5,6,7,8]
```



0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8
-8	-7	-6	-5	-4	-3	-2	-1

```
print(list1[1:4])
```
[2, 3, 4]

```
print(list1[:3])
```
[1, 2, 3]

```
print(list1[4:])
```
[5, 6, 7, 8]

```
print(list1[1:len(list1)])
```
[2, 3, 4, 5, 6, 7, 8]

```
print(list1[::2])
```
[1, 3, 5, 7]

```
print(list1[-1:-5:-1])
```
[8, 7, 6, 5]

AOU
الجامعة العربية المفتوحة
Arab Open University
Lebanon

List Methods and Built-in Functions

Lists have numerous methods that allow you to add elements, remove elements, change the ordering of elements, and so forth.

Below are some useful methods:

Method	Description
append(item)	Adds item to the end of the list.
index(item)	Returns the index of the first element whose value is equal to item.
insert(index, item)	Inserts item into the list at the specified index. When an item is inserted into a list, the list is expanded in size to accommodate the new item. The item that was previously at the specified index, and all the items after it, are shifted by one
sort()	Sorts the items in the list so they appear in ascending order (from the lowest value to the highest value)
remove(item)	Removes the first occurrence of item from the list.
reverse()	Reverses the order of the items in the list.
count (item)	returns the number of occurrences of item in the list
pop(index)	removes the item at the specified index and returns its value

AOU
الجامعة العربية المفتوحة
Arab Open University
Lebanon

List Methods and Built-in Functions

Examples:

```
L1 = [0,3,2]
L1.append(1)
print(L1)
```

[0, 3, 2, **1**]

```
L1 = [0,3,2]
print(L1.index(3))
```

1

```
L1 = [0,3,2]
L1.insert(2,4)
print(L1)
```

[0, 3, **4**, 2]

```
L1 = [0,3,2]
L1.sort()
print(L1)
```

[0, 2, 3]

```
alist=[10,12,15,16,12,13]
alist.remove(12)
print(alist)
```

[10, 15, 16, 12, 13]

```
alist=[10,12,15,16,12,13]
alist.reverse()
print(alist)
```

[13, 12, 16, 15, 12, 10]

```
alist=[10,12,15,16,12,13]
print(alist.count(12))
```

2

```
alist=[10,12,15,16,12,13]
alist.pop(2)
print(alist)
```

[10, 12, 16, 12, 13]

List Methods and Built-in Functions

There are several built-in functions that operate on lists.

Below are some useful functions

Function	Description
len	Returns the number of items in the list
sum	Returns the sum of the items in the list
min	Returns the minimum of the items in the list
max	Returns the maximum of the items in the list

For example, the following computes the average of the values in the list L:

```
L=[5,7,9, 8,10]
average=
sum(L)/len(L)
print(average)
```

➔7.8

Two-Dimensional Lists (Lists of Lists)

A two-dimensional list is a list that has other lists as its elements. Lists of lists are also known as nested lists, or two-dimensional lists. It is common to think of a two-dimensional list as having rows and columns of elements

Example:

List=[['Ali', 5, 10,15],['Naji',12,12,15],['Fadi',10,14,12], ['Rajaa',18,16,14]]

List is a list of lists. Each item in this list is a list that is composed of ,say, name and three grades.
List[1] will report ['Naji', 12, 12, 15]
List[0][0] will report Ali
List[2][2] will report 14

```
1  List=[['Ali', 5, 10,15],['Naji',12,12,15],['Fadi',10,14,12],['Rajaa',18,16,14]]
2  print(List[1])
3  print(List[0][0])
4  print(List[2][2])
```

```
['Naji', 12, 12, 15]
Ali
14
```

List of lists Example

- Suppose we have the following list:

List=[['Ali', 5, 10,15],['Naji',12,12,15],['Fadi',10,14,12], ['Rajaa',18,16,14]]

- Print the average of the second grade for all students.

This is not the function sum! **Why can't we use the sum function here?**

```
#Print the average of the second grade for all students
List=[['Ali',5,10,15],['Naji',12,12,15],['Fadi',10,14,12],['Rajaa',18,16,14]]
Sum=0
for i in range(len(List)):
    Sum+=List[i][2]
averageG2=Sum/len(List)
print(averageG2)
```

Accessing the 2nd grade

```
13.0
```

- Print the name of the student and his/her average grade

In [8]:

```
#Print the name of the student and their average grade
for i in range(len(List)):
    Total=0
    Name=List[i][0]
    for j in range(1,len(List[i])):
        Total+=List[i][j]
    StdAvg=Total/(len(List[i])-1)
    print(Name,':',StdAvg)

```

```
Ali : 10.0
Naji : 13.0
Fadi : 12.0
Rajaa : 16.0
```

OR

In [10]:

```
#Print the name of the student and their average grade

for item in List:
    Total=0
    Name=item[0]
    for j in range(1,len(item)):
        Total+=item[j]
    StdAvg=Total/(len(item)-1)
    print(Name,':',StdAvg)
```

```
Ali : 10.0
Naji : 13.0
Fadi : 12.0
Rajaa : 16.0
```

Tuples

A ***tuple*** is a sequence, very much like a list. The primary difference between tuples and lists is that tuples are immutable. That means once a tuple is created, it cannot be changed.
When you create a tuple, you enclose its elements in a set of parentheses, as shown in the following example:

```
my_tuple = (1, 2, 3, 4, 5)
print(my_tuple)
```

(1, 2, 3, 4, 5)

Initialize a Tuple

There are two ways to initialize an empty tuple.

- You can initialize an empty tuple by having () with no values in them

```
my_Tuple = ()
```

- You can also initialize an empty tuple by using the **tuple** function.

```
my_Tuple = tuple()
```

Tuples

Iteration by item :

The following example shows how a for loop can iterate by the items in a tuple:

```
my_tuple = (1, 2, 3, 4, 5)
for item in my_tuple:
    print(item, end=' ')
```

1 2 3 4 5

Iteration by index:

The following example shows how a for loop can iterate by the index in a tuple:

```
my_tuple = (1, 2, 3, 4, 5)
for i in range(len(my_tuple)):
    print(my_tuple[i], end=' ')
```

1 2 3 4 5

Tuples

- Tuples support all the same operations as lists, except those that change the contents of the list.
- Tuples support the following:
  - Subscript indexing (for retrieving element values only)
  - Methods such as ***index()***
  - Built-in functions such as ***len()***, ***min()***, and ***max()***
  - Slicing expressions
  - The **in** operator
  - The **+** and ***** operators

- **Tuples do not support methods such as append(), remove(), insert(), reverse(), and sort().**

NOTE: If you want to create a tuple with just one element, you must write a trailing comma after the element's value, otherwise Python will not recognize it as a tuple.

```
my_tuple = (1,) # Creates a tuple with one element.
```

Tuples

You might wonder why tuples exist, if the only difference between lists and tuples is immutability!

One reason that tuples exist is performance- faster than processing a list

Another reason is that tuples are safe- not allowed to change the contents of a tuple

Converting Between Lists and Tuples

You can use the built-in **list()** function to convert a tuple to a list, and the built-in **tuple()** function to convert a list to a tuple.

```
my_tuple = (1, 2, 3, 4, 5)
my_list= list(my_tuple)
print(my_list)
```

[1, 2, 3, 4, 5]

```
list1=['Ali', 'Mazen',
'Nawaf']
tuple1= tuple(list1)
print(tuple1)
```

('Ali', 'Mazen',
'Nawaf')

Tuples

Change Tuple Values

Once a tuple is created, you cannot change its values. Tuples are immutable.

But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

```
x = ("apple", "banana",
"cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)
print(x)
```

('apple', 'kiwi', 'cherry')

Tuples

Add Items

Since tuples are immutable, they do not have a build-in append() method, but there are other ways to add items to a tuple.

a) **Convert into a list**: Just like the workaround for changing a tuple, you can convert it into a list, add your item(s), and convert it back into a tuple.

```
my_tuple = ("apple", "banana",
"cherry")
y = list(my_tuple)
y.append("orange")
my_tuple = tuple(y)
print(my_tuple)
```

('apple', 'banana', 'cherry', 'orange')

b) **Add tuple to a tuple.**

```
my_tuple = ("apple", "banana",
"cherry")
y = ("orange",)
my_tuple += y
print(my_tuple)
```

('apple', 'banana', 'cherry', 'orange')

Exercise

Given the following list of lists: **grades_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]].** Convert it to a list of tuples.

```
grades_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
tuples_list=[]
# Convert list of lists to a list of tuples
for item in grades_list:
    tuples_list=tuples_list+[tuple(item)]

print(tuples_list)
```

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

or `tuples_list+=[tuple(item)]`

Extra Exercises

1. Find the second largest element in a list of integers
2. Find the nth largest element in a list of integers
3. Given a list of integers, find the first number that is divisible by 3
4. Given a list of strings, count the number of strings that are longer than 5 characters.
5. Given a list of strings, find the longest string in the list.
6. Write a program that takes in a list of numbers and finds the largest prime number in the list.
7. Given a **list of tuples**, find the average of the second elements of all the tuples.

Summary

This lecture covered:

Lists, including:

- Repetition and concatenation operators
- Indexing
- Techniques for processing lists
- Slicing and copying lists
- List methods and built-in functions for lists
- Two-dimensional lists

Tuples, including:

- Immutability
- Difference from and advantages over lists

AOU
الجامعة العربية المفتوحة
Arab Open University
Lebanon