Chapter 6: Python Lists

Prepared by: Hanan Hardan

Python Lists

- Lists are used to store multiple items in a single variable.
- Lists are created using square brackets:

Example: Create a List

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

List Items

- List items are ordered, changeable, and allow duplicate values.
- List items are indexed, the first item has index [0], the second item has index [1] etc.
- Ordered: It means that the items have a defined order, and that order will not change.
- If you add new items to a list, the new items will be placed at the end of the list.
- Changeable: meaning that we can change, add, and remove items in a list after it has been created.

List Items

- Allow Duplicates: Since lists are indexed, lists can have items with the same value:

Example

```
thislist = ["apple", "banana", "cherry", "apple", "cherry"]
print(thislist)
```

List Length

To determine how many items a list has, use the len() function:

Example: Print the number of items in the list:

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

List Items - Data Types

- List items can be of any data type:

Example: String, int and boolean data types:

```
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]
```

- A list can contain different data types:

Example

```
list4 = ["abc", 34, True, 40, "male"]
List5= [("abc", 34),["abc", 34], "abc", 34]
```

The list() Constructor

- It is also possible to use the list() constructor when creating a new list.

Example:

```
thislist = list(("apple", "banana", "cherry")) # note the double round-brackets
print(thislist)
```

Access List Items

- List items are indexed and you can access them by referring to the index number:

Example: Print the second item of the list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

- Negative indexing means start from the end

-1 refers to the last item, -2 refers to the second last item etc.

Example: Print the last item of the list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

Range of Indexes

- You can specify a range of indexes by specifying where to start and where to end the range.
- When specifying a range, the return value will be a new list with the specified items.

Example: Return the third, fourth, and fifth item:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

Note: The search will start at index 2 (included) and end at index 5 (not included).

Range of Indexes

- Remember that the first item has index 0.
- By leaving out the start value, the range will start at the first item:

This example returns the items from the beginning to, but NOT including, "kiwi":

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[:4])
```

- By leaving out the end value, the range will go on to the end of the list:

This example returns the items from "cherry" to the end:

```
thislist =["apple", "banana", "cherry", "orange", "kiwi", "mango"]
print(thislist[2:])
```

Range of Negative Indexes

- Specify negative indexes if you want to start the search from the end of the list:
- This example returns the items from "orange" (-4) to, but NOT including "mango" (-1):

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[-4:-1])  →
```

```
Names = ("Ali", "Sami", "Omar", "Hani", "Reem")
print(Names[-1:-4:-1]) ←
print(Names[-1:-4]) →
```

Check if Item Exists

- To determine if a specified item is present in a list use the in keyword:

Example: Check if "apple" is present in the list:

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
        print("Yes, 'apple' is in the fruits list")
```

Change List Items

- To change the value of a specific item, refer to the index number:

Example: Change the second item:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

Change a Range of Item Values

To change the value of items within a specific range, define a list with the new values, and refer to the range of index numbers where you want to insert the new values:

Example

Change the values "banana" and "cherry" with the values "blackcurrant" and "watermelon":

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

Change a Range of Item Values

- If you insert *more* items than you replace, the new items will be inserted where you specified, and the remaining items will move accordingly:

Example

Change the second value by replacing it with *two* new values:

```
thislist = ["apple", "banana", "cherry"]
thislist[1:2] = ["blackcurrant", "watermelon"]
print(thislist)
```

Note: The length of the list will change when the number of items inserted does not match the number of items replaced.

Change a Range of Item Values

- If you insert less items than you replace, the new items will be inserted where you specified, and the remaining items will move accordingly:

Example

Change the second and third value by replacing it with one value:

```
thislist = ["apple", "banana", "cherry"]
thislist[1:3] = ["watermelon"]
print(thislist)
```

Add List Items

Append Items

- To add an item to the end of the list, use the append() method:

Example

Using the append() method to append an item:

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

Add List Items

Insert Items

- To insert a list item at a specified index, use the insert() method.

Example

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
thislist.insert(2, "watermelon")
print(thislist)
```

Note: As a result of the example above, the list will now contain 5 items.

Add List Items

Extend List

- To append elements from another list to the current list, use the extend() method.

Example

Add the elements of tropical to thislist:

```
thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical)
print(thislist)
```

Note:The elements will be added to the *end* of the list.

Add List Items

Add Any Iterable

- The extend() method does not have to append lists, you can add any iterable object (tuples, sets, dictionaries etc.).

Example

Add elements of a tuple to a list:

```
thislist = ["apple", "banana", "cherry"]
thistuple = ("kiwi", "orange")
thislist.extend(thistuple)
print(thislist)
```

Remove List Items

Remove Specified Item

- The remove() method removes the specified item.

Example: Remove "banana":

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

Remove List Items

Remove Specified Index

- The pop() method removes the specified index.

Example: Remove the second item:

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

Remove List Items

- If you do not specify the index, the pop() method removes the last item.

Example: Remove the last item:

```
thislist = ["apple", "banana", "cherry"]
thislist.pop()
print(thislist)
```

Remove List Items

- The del keyword also removes the specified index:Example

Example: Remove the first item:

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

- The del keyword can also delete the list completely.

Example: Delete the entire list:

```
thislist = ["apple", "banana", "cherry"]
del thislist
```

Remove List Items

Clear the List

- The clear() method empties the list.
- The list still remains, but it has no content.

Example: Clear the list content:

```
thislist = ["apple", "banana", "cherry"]
thislist.clear()
print(thislist)
```

Loop Lists

Loop Through a List

- You can loop through the list items by using a for loop:

Example: Print all items in the list, one by one:

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

Loop Lists

Loop Through the Index Numbers

- You can also loop through the list items by referring to their index number.
- Use the range() and len() functions to create a suitable iterable.

Example: Print all items by referring to their index number:

```
thislist = ["apple", "banana", "cherry"]
for i in range(len(thislist)):
  print(thislist[i])
```

List Comprehension

- List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.
- Comprehension syntax:

newlist = [expression for item in list if condition == True]

- The expression is some calculation or operation acting upon the variable item
- The condition is like a filter that only accepts the items that valuate to True.
- The return value is a new list, leaving the old list unchanged.

List Comprehension

Example: Based on a list of fruits, you want a new list, containing only the fruits with the letter "a" in the name.

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

Without list comprehension	With list comprehension

newlist = []
for x in fruits:
  if "a" in x:
  newlist.append(x)
print(newlist)
	newlist = [x for x in fruits if "a" in x]
	print(newlist)

List Comprehension

Example: Only accept items that are not "apple":

```
newlist = [x for x in fruits if x != "apple"]
```

- The condition if x != "apple" will return True for all elements other than "apple", making the new list contain all fruits except "apple".

Example:

```
li = [3, 6, 2, 7]
l1=[elem*2 for elem in li]
print(l1)
```

List Comprehension

- The condition is optional and can be omitted:

Example: With no if statement:

```
newlist = [x for x in fruits]
```

- The iterable can be any iterable object, like a list, tuple, set etc.

Example: You can use the range() function to create an iterable:

```
newlist = [x for x in range(10)]
```

Example: Accept only numbers lower than 5:

```
newlist = [x for x in range(10) if x < 5]
```

List Comprehension

Expression

- The *expression* is the current item in the iteration, but it is also the outcome, which you can manipulate before it ends up like a list item in the new list:

Example: Set the values in the new list to upper case:

```
newlist = [x.upper() for x in fruits]
```

- You can set the outcome to whatever you like:

Example: Set all values in the new list to 'hello':

```
newlist = ['hello' for x in fruits]
```

List Comprehension

- The *expression* can also contain conditions, not like a filter, but as a way to manipulate the outcome:

Example: Return "orange" instead of "banana":

```
newlist = [x if x != "banana" else "orange" for x in fruits]
```

- The *expression* in the example above says:

"Return the item if it is not banana, if it is banana return orange".

Sort Lists

- List objects have a sort() method that will sort the list alphanumerically and numerically, ascending, by default:
- Sort List Alphanumerically

Example:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort()
print(thislist)
```

- Sort the list numerically:

Example:

```
thislist = [100, 50, 65, 82, 23]
thislist.sort()
print(thislist)
```

Sort Lists

- Sort Descending :To sort descending, use the keyword argument reverse = True

Example 1:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort(reverse = True)
print(thislist)
```

Example 2:

```
thislist = [100, 50, 65, 82, 23]
thislist.sort(reverse = True)
print(thislist)
```

Sort Lists

Reverse Order

- What if you want to reverse the order of a list, regardless of the alphabet?

The reverse() method reverses the current sorting order of the elements.

Example: Reverse the order of the list items:

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.reverse()
print(thislist)
```

Join Lists

- There are several ways to join, or concatenate, two or more lists in Python.

One of the easiest ways are by using the + operator.

Example: Join two list:

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]
list3 = list1 + list2
print(list3)
```

Join Lists

Another way to join two lists is by appending all the items from list2 into list1, one by one:

Example: Append list2 into list1:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]
for x in list2:
  list1.append(x)
print(list1)
```

Join Lists

Or you can use the extend() method, which purpose is to add elements from one list to another list:

Example: Use the extend() method to add list2 at the end of list1:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]
list1.extend(list2)
print(list1)
```

Collection data types

- There are four collection data types in the Python programming language:
- **List** is a collection which is ordered and changeable. Allows duplicate members.
- Tuple is a collection which is ordered and unchangeable. Allows duplicate members.
- Set is a collection which is unordered and unindexed. No duplicate members.
- Dictionary is a collection which is ordered* and changeable. No duplicate members.
- *As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.
- When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.