List Revision

Lecture 18

COMPSCI 101, S2 2020

The Python List

A list is a collection of objects.

- Can contain objects of different types.
- Can contain duplicate members.

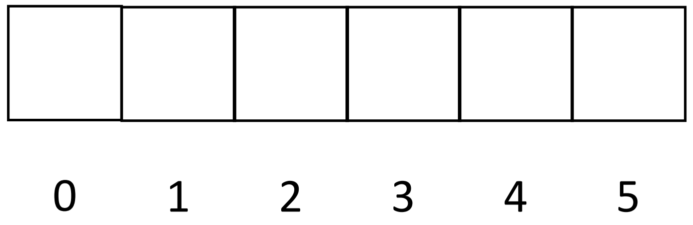
A list with elements can be created by using square brackets [ ] around a comma separated list of items:

```
integer_list = [1, 2, 3, 4]
string_list = ["hello", "world"]
mixed_list = [1, "hello", 2, "world", 2]
```

An empty list can be created using:

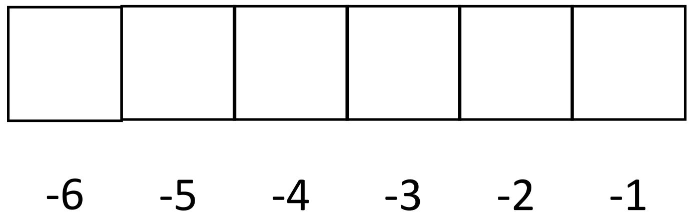
- Square brackets with no items in between: `a_list = []`
- Using the list() function: `a_list = list()`

Lists Are Ordered



Positive indices:

- First item at index 0. Last item at index (length of list – 1).



Negative indices:

- First item at index (- length of list). Last item at index -1.

Accessing List Items

Items in a list can be accessed using their index number:

```
mixed_list = [1, "hello", 2, "world", 2]
print(mixed_list[0])
print(mixed_list[3])
print(mixed_list[-1])
print(mixed_list[-5])
```

```
1
world
2
1
```

Lists Are Mutable

Lists are mutable:

- The value of a list item at a particular index can be changed.

```
a_list = [1, 2, 3, 4]
a_list[0] = 9
print(a_list)  [9, 2, 3, 4]
a_list[-2] = 13
print(a_list)  [9, 2, 13, 4]
```

Slicing Lists

List slicing is performed in much the same way string slicing is:

- `a_list[start:end:step]`
- `start` indicates the starting index of the range of indices you are interested in. The item at the start index is included in the slice. If you omit the start index, the slice starts at index 0.
- `end` indicates the ending index of the range of indices you are interested in. The item at the end index is not included in the slice. If you omit the end index, the slice goes up to and including the last item in the list.
- `step` indicates the gap between indices for each element included in the slice. The default gap is 1 (which is what is used when you omit the step).

A list slice is itself a list object.

Slicing Lists

```
a_list = ["cat", "dog", "rat", "bird", "hamster"]
slice1 = a_list[1:3]
print(slice1)  ['dog', 'rat']
print(type(slice1)) <class 'list'>
slice2 = a_list[:2]
print(slice2)  ['cat', 'dog']
slice3 = a_list[3:]
print(slice3)  ['bird', 'hamster']
slice4 = a_list[::2]
print(slice4)  ['cat', 'rat', 'hamster']
```

Finding The Length Of A List

The length of a list can be obtained by using the in-built function `len()`.

```
a_list = ["cat", "dog", "rat", "bird", "hamster"]

print(len(a_list))  5

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

print(len(a_list))  10
```

Other In-Built Functions That Work With Lists

- `min(a_list)` returns the minimum item in the list.
- `max(a_list)` returns the maximum item in the list.
- `sum(a_list)` returns the sum of the items in the list (only for lists with numerical items).

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

print(min(a_list))  1

print(max(a_list))  10

print(sum(a_list))  55
```

List Methods – `sort()`

The list method sort(), sorts the items of a list in place.

- If the items are numerical, the items are sorted in ascending order (smallest to largest).
- If the items are strings, the items are sorted alphabetically.

Note that the sort() method only alters the order of the items of a list. It does not return a new list object.

```
a_list = [4, 6, 66, 1, -8, 23]
a_list.sort()
print(a_list) [-8, 1, 4, 6, 23, 66]
a_list = ["dog", "bird", "zebra", "cat"]
a_list.sort()
print(a_list) ['bird', 'cat', 'dog', 'zebra']
```

List Methods – `reverse()`

The reverse() list method reverses the items in a list, in place. Only the order of the list items is modified. It does not return a new list object.

```
a_list = [4, 6, 66, 1, -8, 23]
a_list.reverse()
print(a_list)    [23, -8, 1, 66, 6, 4]
a_list = ["dog", "bird", "zebra", "cat"]
a_list.reverse()
print(a_list)    ['cat', 'zebra', 'bird', 'dog']
```

List Methods – `index(x)`

The list method `index(x)` returns the index of the first item from the left in the list with a value equal to `x`.

- Python throws an error if there is no such item in the list.

```
a_list = [1, 24, 67, 75]

print(a_list.index(67))
```

```
2
```

```
print(a_list.index(100))
```

```
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    print(a_list.index(100))
ValueError: 100 is not in list
```

List Methods – `pop(index)`

The `pop(index)` list method removes and returns the item at the position given by the parameter `index`.

- The 'popped’ item is returned by the method.
- An error results if there is no such index in the list.
- Using the `pop()` method without a parameter removes and returns the last item in the list.

```
a_list = [4, 6, 66, 1, -8, 23]

item = a_list.pop()

print("Item:",item,"a_list:",a_list)
```

```
Item: 23 a_list: [4, 6, 66, 1, -8]
```

```
item = a_list.pop(2)

print("Item:",item,"a_list:",a_list)
```

```
Item: 66 a_list: [4, 6, 1, -8]
```

List Methods – `insert(index, x)`

The list method `insert(index, x)` inserts the object `x` into a list at the given `index`.

- Items at subsequent indices are shifted over by 1.

```
a_list = [10, 20, 30, 40, 50]
a_list.insert(1,15)
print(a_list)  [10, 15, 20, 30, 40, 50]
a_list.insert(3, 25)
print(a_list)  [10, 15, 20, 25, 30, 40, 50]
```

List Methods – `append(x)`

The list method `append(x)` adds the object `x` to the end of a list.

```
a_list = [10, 20, 30, 40, 50]
a_list.append(45)
print(a_list)  [10, 20, 30, 40, 50, 45]
a_list.append(35)
print(a_list)  [10, 20, 30, 40, 50, 45, 35]
a_list.append(100)
print(a_list)  [10, 20, 30, 40, 50, 45, 35, 100]
```

Operators That Can Be Used With Lists – +

The + operator is used to concatenate two lists, returning a new list containing all the items of the first list followed by all the items of the second list.

- You can only concatenate a list with another list. You cannot concatenate a list to another object like a string, integer etc.

```
list1 = [1, 2, 3]

list2 = [4, 5, 6]

new_list = list1 + list2

print(new_list)
```

```
[1, 2, 3, 4, 5, 6]
```

```
new_list = list1 + 3
```

```
Traceback (most recent call last):
  File "<pyshell#41>", line 1, in <module>
    new_list = list1 + 3
TypeError: can only concatenate list (not "int") to list]
```

Differences Between Using `append()` And List Concatenation

The append() list method adds the object passed as its parameter to the end of a list. An existing list is updated. A new list is not created.

```
a_list = [1, 2, 3, 4]
a_list.append(5)
print(a_list)
```

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

The list object assigned to `a_list` is updated: the integer 5 is added to the end of the list.

```
a_list = [1, 2, 3, 4]
a_list = a_list + [5]
print(a_list)
```

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

A new list object is created by the concatenation, containing the elements of the list object initially assigned to `a_list` (1, 2, 3, and 4) and the element of the second list (5). This new list object is assigned to `a_list`.

Differences Between Using **`append()`** And List Concatenation

You can use the append() list method to add any object to the end of a list:

```
a_list = [1, 2, 3]
a_list.append(4)
print(a_list)
```

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

```
a_list.append("5")
print(a_list)
```

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

You can only concatenate a list object with another list object:

```
list1 = [1, 2, 3]
new_list = list1 + 3
```

```
Traceback (most recent call last):
  File "<pyshell#59>", line 1, in <module>
    new_list = list1 + 3
TypeError: can only concatenate list (not "int") to list
```

What Happens When You Use The `append()` Method With A List Object As Its Parameter?

When you use the `append()` list method with a list object as its parameter, this list object will be added as an item to the end of a list:

```
first_list = [1, 2, 3]
second_list = [4, 5, 6]
first_list.append(second_list)
print(first_list) [1, 2, 3, [4, 5, 6]]
```

Compare this to what happens when concatenation is used:

```
first_list = [1, 2, 3]
second_list = [4, 5, 6]
first_list = first_list + second_list
print(first_list) [1, 2, 3, 4, 5, 6]
```

Operators That Can Be Used With Lists – *

You can use the * operator with a list object as one operand and an integer as the other.

- The result is a new list object containing the items of the list object used as an operand, repeated a certain number of times (as dictated by the integer operand).

```
a_list = [1, 2]
new_list = a_list * 2
print(new_list)
```

```
[1, 2, 1, 2]
```

```
new_list = a_list * 4
print(new_list)
```

```
[1, 2, 1, 2, 1, 2, 1, 2]
```

Operators That Can Be Used With Lists – `in`

The `in` operator can be used to check for list membership.

- The first operand is the object you are looking for in a list.
- The second operand is the list object you are searching within.
- The in operator returns `True` if the object is in the list, and `False` otherwise.

```
a_list = [1, 2, 3, 4]
print(3 in a_list)  True
print(55 in a_list) False
```

You should use the in operator to check whether an object is a member of a list before using this object as a parameter to the `index()` list method.

Comparing Lists Using The `==` And `is` Operators

You can use the == operator to compare 2 list objects:

- The == operator returns `True` if both list objects contain the same items in the same order and `False` otherwise.

```
list1 = [1, 2, 3]
list2 = [1, 2, 3]
print(list1 == list2)
```

```
True
```

```
list3 = [3, 2, 1]
print(list1 == list3)
```

```
False
```

Comparing Lists Using The `==` And `is` Operators

You can use the `is` operator to compare 2 list objects:

- The `i s` operator returns `True` if both list objects are in fact the same list object, and `False` otherwise.

```
list1 = [1, 2, 3]

list2 = [1, 2, 3]

print(list1 is list2)  False

list3 = list1

print(list1 is list3)  True
```

Iterating Through The Items In A List Using A `for … in range()` Loop

If you want to loop through the items in a list using the list indices, you should use the `for … in range()` loop:

```
a_list = [1, 2, 3, 4, 5]
for i in range(len(a_list)):
    a_list[i] = a_list[i] * 2
print(a_list)
```

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

Iterating Through The Items In A List Using A `for … in` Loop

If you want to loop through the items in a list and the list indices are not required, you should use the `for … in` loop:

```
a_list = [1, 2, 3, 4, 5]
for number in a_list:
    if number % 2 == 0:
        print(number, "is even.")
    else:
        print(number, "is odd.")
```

```
1 is odd.
2 is even.
3 is odd.
4 is even.
5 is odd.
```

Changing The Contents Of A List Using A Function

Since list objects are mutable, you can change the contents of a list by passing it to a function.

- A function can change the contents of the list in place. If this is the case, you do not need to create and return a new list object.

```
def double_list_item_value(a_list):
    for i in range(len(a_list)):
        a_list[i] = a_list[i] * 2

def main():
    a_list = [1, 2, 3, 4]
    print("Before function call:", a_list)
    double_list_item_value(a_list)
    print("After function call:", a_list)

main()
```

```
Before function call: [1, 2, 3, 4]
After function call: [2, 4, 6, 8]
```