Chapter 4 - Lists

Goals

- learn how to create and edit Python lists
- learn how to use indexing and slicing to fetch items from lists, replace items and delete items
- apply functions and list methods

4.0 Overview

Base Python provides many ways of organizing collections of items including lists, tuples, sets and dictionaries. Python libraries extend these options to numpy arrays, padas dataframes and more. The options can be a bit overwhelming at first. We'll focus on Lists in this chapter and NumPy arrays in Chapter 8.

Lists are one of the most commonly used objects in Python, even though they are not the most effecient for intensive computational work. Many of the concepts developed in this chapter can be direclty transferred to other objects such as NumPy arrays discussed later in this tutorial.

4.1 A list stores many values in a single structure

- A Python list is a built-in data structure that stores an ordered collection of items (which can be of different types).
- Python lists are mutable, meaning they can be modified after they are created
- Lists are created using **square brackets**:

```
my_list = [10, 20, 30, "hello", True, 60]
print('my_list contains these items:  ',my_list)
```

4.2 Use the function `len()` to find the number of elements in a list

```
n = len(my_list)                     # find the number of elements in my_list
print(f'my_list has {n} elements')   # display the result
```

4.3 Use an item's index to fetch it from a list

- Each position in the list (first, second, etc.) is given a number. This number is called an index
- Indices in Python are numbered from 0. So the first item in the list has index 0, the second has index 1, and so on.
- Use the position's index in square brackets to retreive the item from the list
- Note: the terms "item" and "element" are often used interchangeably

```
print("first element (index 0)  = ",my_list[0])
print("second element (index 1) = ",my_list[1])
print("third element (index 2)  = ",my_list[2])
```

✅ Skill Check 1

Print the fourth element of the following array:

```
A = [2,4,6,8,10,12]
```

4.4 The last item in a list has index [-1]

- Python uses negative indices to count backwawrd from the end of the list. Thus the index [-1] is shortcut to quickly access the last element of a list:

```
print('The last item in my list is',my_list[-1])
```

- The next-to-the last item is given by [-2] and so on

```
print('The 2nd-to-the-last item in my list is',my_list[-2])
```

4.5 Indexing in Python starts at zero

This was already covered in the previous section, but it is super, super important. Some programming languages like Fortran, Matlab and R start counting at 1 because that's what people naturally do. Other languages like Python, C, Java, etc. count from 0 because it represents an offset from the first value in the array and is closer to the way computers represent arrays. If you are interested in the historical context, you can check out Mike Hoye's blog post. It takes a bit of getting used to and is why this tutorial stars all chapters and sections with 0 instead of 1.

4.6 Use slicing to fetch multiple items from a list

- A part of a list is called a **slice**. A slice can be as short as a single item or as long as the entire list.
- We take a slice by using `[start:stop]` notation, where `start` is replaces with the index of hte first item we want and `stop` is the index *just after* the last item we want.
- Mathematically, you might say that a slice selects indices n such that $start \leq n < stop$.
- The difference between `stop` and `start` is the length of the list.
- Taking a slice does not change the contents of the original list. Instead, the slice is a copy of the original list.

```
print("The slice new_list[1:4] =",my_list[1:4])
```

- Leaving off `stop` will extend the slice from `start` to the end of the list:

```
print("slice new_list[3:] slice =       ",my_list[3:])
```

- Leaving off `start` will extend the slice from 0 up to `stop`:

```
print("slice new_list[3:] slice =       ",my_list[0:3])
```

✅ Skill Check 2

Print the first 4 elements of the following array:

```
A = [2,4,6,8,10,12]
```

4**.7 A list's values can be replaced by assigning new values to them**

- To replace the 5th item (index = 4), we make the following assignment:

```
my_list[4] = 50
print("updated list = ",my_list)
```

- You can replace a group of elements with a slice as long as number of elements in the slice matches the number of elements specified in the array. In the following example, we replace the first three elements of `my_list` with the letters `a, b, c`.

```
my_list[0:3] = ['a','b','c']
print("updated list = ",my_list)
```

4.8 Use the list method .append() to add items to the end of a list

- Most objects in Python have a set of dedicated functions that operate on the object they are attached to. These dedicated functions are called **methods**.
- To implement a method, we type the name of the object, a "." and then the name of the method: object_name.method_name()
- The add an item to the end of a list we use the `.append()` method. We'll add an element "goodbye" to the end of our list:

```
my_list.append('goodbye')
print('updated my_list = ',my_list)
```

- We can add a list to the end of another list using the `.extend()` method. We'll add a list `[80, 90, 100]' to the end of our list:

```
my_list.extend([80, 90, 100])
print('updated my_list = ',my_list)
```

Double-click (or enter) to edit

✅ Skill Check 3

In the following code, we have defined three lists of prime numbers. Using the methods described in this chapter, combine these lists to a single list and then print it to the screen.

```
baby_primes = [2, 3, 5, 7]
teen_primes = [11, 13, 17, 19]
adult_primes = [23, 27, 29, 31, 37]
```

4.9 use the `del' command to delete an element or slice from a list by index

- We can delete either individual elements or entire slices like this. In this example, we delete the fourth element (index = 3), which contains "hello":

```
del my_list[3]
print('updated my_list = ',my_list)
```

- Let's now delete the last three elements using negative indexing:

```
del my_list[-3:]
```

```
print('updated my_list = ',my_list)
```

✅ Skill Check 4

The following code cell contains a list of letters and numbers. Use methods described in this chapter to remove the numbers from the list.

```
even_numbers = ['a', 'b', 'c', 999, 'd', 999, 'e']
```

4.10 Use the `.remove()` method to delete an element by value

- Instead of providing the index of the element to delete, you can use the `.remove()` method to provide the contents of the element. The `.remove()` command will delete the first occurance of the element from the list. If the value appears in multiple locations in the list, the others will remain

```
my_list.remove('goodbye')
print('updated my_list = ',my_list)
```

4.11 Remove all elements of an array by setting it equal to the empty list `[]`

- Here's an example:

```
my_list = []
print('updated my_list = ',my_list)
```

- If you want to remove the list itself (and not just its contents), you can use the `del` command. After running the following command, `my_list` will no longer show up in the Variable Inspector.

```
del my_list
```

4.12 Swap elements

Python provides a fancy way of swapping elements of a list (or any pair of variables).

First, let's try it with two variables `x` and `y`. We simply reverse the order of the variables across the equals sign like this:

```
x = 1                       # define x
y = 2                       # define y
x, y = y, x                 # swap x and y
print('x =',x,'and y =',y)  # print the result
```

Let's now apply this trick to lists. We'll define a list and then swap the first two elements:

```
A = [10, 20, 30, 40, 50]
A[0], A[1] = A[1], A[0]
print('updated A = ',A)
```

✅ Skill Check 5

Using only thw swap operator, write a series of commands to place the numbers in the following list in order from low to high.

```
my_list = [20, 50, 60, 10, 30, 40, 70]
```

4.13 Use `.copy()` to copy a list

Let's say we want to copy the following list. It is tempting to just use the assignment operator "=" as follows:

```
my_list = [1,2,3]              # define a list
new_list = my_list             # use "=" to set a pointer from the new list to the original
print('new_list = ',new_list)  # show that the new list contains the same elements as the original
```

However, something strange happens if we modify the original list:

```
my_list[0] = 10                # modify the original list
print('my_list = ',my_list)    # show that the orininal list was modified
print('new_list = ',new_list)  # show that the new list is also modified!!
```

- Notice that when we modify the original list, the new list is also modified!!
- In Python, the assignment operator "=" doesn't make a copy, rather it merely points back to the original list.

To make a copy of a list that is not "tethered" to the original one, use the `.copy()` method:

```
my_list  = [1,2,3]             # define a list
new_list = my_list.copy()      # make an independent copy
my_list[0] = 10                # make a change to the original list
print('my_list = ',my_list)    # show that the original list was changed
print('new_list = ',new_list)  # show that the copy stays unchanged
```

- When we use the `.copy()` method the two lists are independent and modifying one won't affect the other.

4.14 Sort list elements

We can sort the elements of a list with the `.sort()` method like this:

```
my_list = [4, 8, 1, 9, 3, 5, 3, 1, 9, 2]

my_list.sort()
print('sorted my_list = ',my_list)
```

✅ Skill Check 6

The following Code Cell contains a list of particle names. Sort the list in alphabetical order and print the result.

```
a = ['proton','quark','gluon','electron','muon','boson']
```

🔆 Example - Largest three values in a list

Suppose we want to find the largest three values in a list. We can accomplish this by sorting the list and then printing the last three elements. Let's assume we don't want to change the order of the elements in the original list.

Solution

Since we don't want the mess with the ordering of our list, we do the following:

- define the list
- make a copy using `.copy()`
- sort the copy
- print the last three elemnts of the sorted list

```
a = [5,7,9,2,1,6,4,8,10,2,7]
b = a.copy()
b.sort()
print("the largest three values are",b[-3:])
```

4.15 Functions and methods that work on lists

List Methods

In Python a method is a function that is defined within an object. Methods are called with the following syntax: object.method(). The previous section described how to use the `sort()` method on lists. Here's a list of several list methods:

- .append(x) - add item `x` to the end of a list
- .extend(a) - add a list to the end of another list
- .sort() - sort the list
- .insert(i,x) - insert the item `x` at index position `i`
- .count(x) - return th number of occurances of `x`
- .reverse() - reverse the ordering of elements in the list
- .copy() - returns a copy of the list
- .index(x) - returns the index of the first occurance of `x`

Built-in functions

The following are functions buit into Python that work on lists:

- len(my_list) - number of elements in the list
- sum(my_list) - sum of numeric elements
- min(my_list) - minimum (smallest) value in the list
- max(my_list) - maximum value

🔆 Example functions and methods

```
my_list = [4,7,1,7,9,0,10]

print("length of my_list = ",len(my_list))
print("sum of my_list = ",sum(my_list))
print("min of my_list = ",min(my_list))
print("index of teh first 7 in my_list = ",my_list.index(7))
```

✅ Skill Check 7

Use the `sum()` function to calculate and print the average of the following list. Remember the average is the sum divided by the number of elements in the list.

```
my_list = [3,6,17,2,1,-8,6,1,3,9]
```

✅ Skill Check 8

The following code cell contains a list of values. Use the methods described in this chapter to create a new list with the minimum value and the maximum value removed. Your method should automatically find the extreme values, so that it will still work if the array elements change. Print your new list. Calculate and print the average of the new list.

```
my_list = [3,6,17,2,1,-8,6,1,3,9]
```

4.16 Character strings can be indexed and sliced like lists

- The characters (individual letters, numbers and so on) in a string are ordered like elements in a list.
- You can retreive characters from text string using indexes and slices in square brackets, just like lists
- The built-in function `len()` counts the unber of characters in a string, just like it returns the number of elements in a list
- Here are some examples:

```
element = 'carbon'
print('length of the element string = ', len(element))
print('first letter of the element (element[0]) = ', element[0])
print('first thee letters (element[:3]) = ', element[:3])
```

Even though strings act like lists in some ways, they are NOT lists.

- Unlike lists, strings are not mutable, meaning that once they are created they cannot be modified
- It is not possible to replace characters in a string using list indexing. The following will throw an error:

```
element[0] = 'C'
```

- If you want to replace all occurances of a particular character with another, you can do so with the `.replace()` method for strings. Because, you cannot edit strings, you have to save the modified version as a new string. And, yes, it is ok to save it with the same name. This bascially overwrites the old string with the new one. You can do much more as well, but that is beyond the scope of the current discussion.

```
element = element.replace('c','C')
print("updated element string =", element)
```

✅ Skill Check 9

The words "pancake" and "feedback" are two examples where reversing the syllables makes a meaningful word, in this case "cakepan" and "backfeed" respectively. Using slicing and concatenation, reverse the syllables of these two words

```
word1 = "pancake"
word2 = "feedback"
```

4.17 Don't confuse lists and tuples

A Python tuple is is similar to a list, but its contents cannot be changed once it is created, i.e. it is immutable. We won't discuss them much at the moment other to say they are created with parentheses instead of square brackets:

```
my_tuple = (1,2,3)
```

For now, the main take-away is: if you accidently use parentheses instead of square brackets when trying to create a list, you will actually be creating a tuple. This can lead to error messages (see below), angst and frustration if done unintentionally. Be careful with the notation!

4.18 Practice problems

4.19 Errors

The following are common errors encountered when using lists:

Mixing up [ ], ( ) and { }

Trying to define a list with parentheses `( )` or curly braces `{ }` instead of square brackets `[ ]` will define the obect as a tuple or a set, respectively, rather than a list. Often, this won't throw an error immediately, since these are also valid objects. Errors will arise, however, if you try to update a tuple since tuples aren't mutable:

Index out of range error

Trying to reference a list element that is beyond the end of the list will throw an error. Remember that if a list has `N` elements, the last element will be `[N-1]` and not `[N]`.

```
my_list = [1,2,3]   # list defined with 3 elements
print(my_list[3])   # valid indices run from 0 to 2, so 3 is out of bounds
```

Array slicing

Array slicing can often lead to errors if you try to get too fancy. Trying to assign a set of elements when the number of elements don't match. Unless you are very careful about what you are doing, this can lead to unexpected results. For example, if you want to modify the first three elements of a list, but accidently assign the three elements to the first position in the target list, you will end up with one of the list elements being a list itself. This is an allowed operation, but not what was intended:

```
a = [10,20,30,40,50]
a[0] = [1,2,3]
print("array a is now = ",a)
```

Here's the correct way:

```
a = [10,20,30,40,50]
a[0:3] = [1,2,3]
print("array a is now = ",a)
```

Key Points

- Lists are one of many ways of organizing data in Python
- Lists are created using square brackets `[item1, item2, item3...]`
- The `len()` function returns the number of elements in a list
- Use an item's index to fetch it from its list, e.g. `my_list[0]`
- The Python shortcut for the last item in a list is `my_list[-1]`

- Python idexing starts at 0 instead of 1
- Use slicing to fetch multiple items from a list: `my_list[start:end]`
- A list's values can be replaced by assigning new values to them: `my_list[3] = 3.14'
- Methods are functions that are tied to specific objects. Python lists have many methods available
- Use the list method `.append()` to add items to the end of a list
- use the `del` command to delete items from a list by index
- use the `.remove()` method to delete items by value
- Remove all elements from a list by setting it equal to the empty list: `my_list = []'
- Swap elements using the notation `a[1],a[2] = a[2],a[1]`
- Cope lists using the `.copy()` method
- Characters in a string can be fetched using the same indexing as lists
- Don't forget to use square brackets to create lists. If you use parentheses `()` you will be creating a tuple, which has very different properties.

This tutorial is a modified adaptation of link text "Python for Physicists" © Software Carpentry