Python Programming
Module 3: Introducing Lists

Learning objectives

1. What lists are and how to work with individual items in a list.
2. Define a list and how to add and remove elements.
3. Sort lists permanently and temporarily for display purposes.
4. Find the length of a list.
5. How to avoid index errors when you're working with lists.

LISTS

What is a list?

A list is a collection of items in a particular order. You can make a list that includes the letters of the alphabet, the digits from 0-9, or the names of all the people in your family. A list usually contains more than one element, it's a good idea to make the name of your list plural, such as letters, digits, or names.
In Python, square brackets ([]) indicate a list, and individual elements in the list are separated by commas. Here's a simple example of a list that contains a few kinds of bicycles:

```
bicycles=['trek','cannondale','redline','specialized']
print(bicycles)
```

If you ask Python to print a list, Python returns its representation of the list, including the square brackets:

```
['trek','cannondale','redline','specialized']
```

Accessing Elements in a List

Lists are ordered collections, so you can access any element in a list by telling Python the position, or index, of the item desired. To access an element in a list, write the name of the list followed by the index of the item enclosed in square brackets.
For example, let's pull out the first bicycle in the list bicycles:

```
  bicycles=['trek','cannondale','redline','specialized']
1 print(bicycles[0])
```

The syntax for this is shown at `1`. When we ask for a single item from a list, Python returns just that element without square brackets or quotation marks:

```
trek
```

This is the result you want your users to see clean, neatly formatted output.
You can also use the string methods on any element in a list. For example, you can format the element `'trek'` more neatly by using the `title()` method:

```
bicycles=['trek','cannondale','redline','specialized']
print(bicycles[0].title())
```

This example produces the same output as the preceding example except `'Trek'` is capitalized.

Index Positions Start at 0, Not 1

Python considers the first item in a list to be at position 0, not position 1. This is true of most programming languages, and the reason has to do with how the list operations are implemented at a lower level. If you're receiving unexpected results, determine whether you are making a simple off-by-one error.

The second item in a list has an index of 1. Using this simple counting system, you can get any element you want from a list by subtracting one from its position in the list. For instance, to access the fourth item in a list, you request the item at index-3.
The following asks for the bicycles at index-1 and index-3:

```
bicycles=['trek','cannondale','redline','specialized']
print(bicycles[1])
print(bicycles[3])
```

This code returns the second and fourth bicycles in the list:

```
cannondale
specialized
```

Python has a special syntax for accessing the last element in a list. By asking for the item at index-1, Python always returns the last item in the list:

```
bicycles=['trek','cannondale','redline','specialized']
print(bicycles[-1])
```

This code returns the value `'specialized'`. This syntax is quite useful, because you'll often want to access the last items in a list without knowing exactly how long the list is. This convention extends to other negative index values as well. The index-2 returns the second item from the end of the list, the index-3 returns the third item from the end, and so forth.

Using Individual Values from a List

You can use individual values from a list just as you would any other variable. For example, you can use concatenation to create a message based on a value from a list.
Let's try pulling the first bicycle from the list and composing a message using that value.

```
bicycles=['trek','cannondale','redline','specialized']
1 message="My first bicycle was a "+bicycles[0].title()+"."

print(message)
```

At `1`, we build a sentence using the value at `bicycles[0]` and store it in the variable message. The output is a simple sentence about the first bicycle in the list:

```
My first bicycle was a Trek.
```

Changing, adding, and removing elements

Most lists you create will be dynamic, meaning you'll build a list and then add and remove elements from it as your program runs its course. For example, you might create a game in which a player has to shoot aliens out of the sky. You could store the initial set of aliens in a list and then remove an alien from the

list each time one is shot down. Each time a new alien appears on the screen, you add it to the list. Your list of aliens will decrease and increase in length throughout the course of the game.

Modifying Elements in a List

To change an element, use the name of the list followed by the index of the element you want to change, and then provide the new value you want that item to have.

For example, let's say we have a list of motorcycles, and the first item in the list is `'honda'`. How would we change the value of this first item?

```
1 motorcycles=['honda','yamaha','suzuki']
  print(motorcycles)

2 motorcycles[0]='ducati'
  print(motorcycles)
```

The code at `1` defines the original list, with `'honda'` as the first element. The code at `2` changes the value of the first item to `'ducati'`. The output shows that the first item has indeed been changed, and the rest of the list stays the same:

```
['honda','yamaha','suzuki']
['ducati','yamaha','suzuki']
```

You can change the value of any item in a list, not just the first item.

Adding Elements to a List

You might want to add a new element to a list for many reasons. For example, you might want to make new aliens appear in a game, add new data to a visualization, or add new registered users to a website you've built. Python provides several ways to add new data to existing lists.

Appending Elements to the End of a List

When you append an item to a list, the new element is added to the end of the list. Using the same list that we had in the previous example, we'll add the new element `'ducati'` to the end of the list:

```
motorcycles=['honda','yamaha','suzuki']
print(motorcycles)

1 motorcycles.append('ducati')
  print(motorcycles)
```

The `append()` method at `1` adds `'ducati'` to the end of the list without affecting any of the other elements in the list:

```
['honda','yamaha','suzuki']
['honda','yamaha','suzuki','ducati']
```

The `append()` method makes it easy to build lists dynamically. For example, you can start with an empty list and then add items to the list using a series of `append()` statements. Using an empty list, let's add the elements `'honda'`, `'yamaha'`, and `'suzuki'` to the list:

```
motorcycles=[]

motorcycles.append('honda')
motorcycles.append('yamaha')
motorcycles.append('suzuki')

print(motorcycles)
```

The resulting list looks exactly the same as the lists in the previous examples:

```
['honda','yamaha','suzuki']
```

Inserting Elements into a List

You can add a new element at any position in your list by using the `insert()` method. You do this by specifying the index of the new element and the value of the new item.

```
motorcycles=['honda','yamaha','suzuki']

1 motorcycles.insert(0, 'ducati')
  print(motorcycles)
```

In this example, the code at `1` inserts the value `'ducati'` at the beginning of the list. The `insert()` method opens a space at position 0 and stores the value `'ducati'` at that location. This operation shifts every other value in the list one position to the right:

```
['ducati','honda','yamaha','suzuki']
```

Removing an Item Using the del Statement

If you know the position of the item you want to remove from a list, you can use the `del` statement.

```
motorcycles=['honda','yamaha','suzuki']
print(motorcycles)

1 del motorcycles[0]
  print(motorcycles)
```

The code at `1` uses `del` to remove the first item, `'honda'`, from the list of motorcycles:

```
['honda','yamaha','suzuki']
['yamaha','suzuki']
```

You can remove an item from any position in a list using the `del` statement if you know its index. For example, here's how to remove the second item, `'yamaha'`, in the list:

```
motorcycles=['honda','yamaha','suzuki']
print(motorcycles)

del motorcycles[1]
print(motorcycles)
```

The second motorcycle is deleted from the list:

```
['honda','yamaha','suzuki']
['honda','suzuki']
```

In both examples, you can no longer access the value that was removed from the list after the `del` statement is used.

Removing an Item Using the `pop()` Method

The `pop()` method removes the last item in a list, but it lets you work with that item after removing it. The term `pop` comes from thinking of a list as a stack of items and popping one item off the top of the stack. In this analogy, the top of a stack corresponds to the end of a list.
Let's pop a motorcycle from the list of motorcycles:

```
1 motorcycles=['honda','yamaha','suzuki']
  print(motorcycles)

2 popped_motorcycle=motorcycles.pop()
3 print(motorcycles)
4 print(popped_motorcycle)
```

We start by defining and printing the list motorcycles at `1`. At `2`, we pop a value from the list and store that value in the variable `popped_motorcycle`. We print the list at `3` to show that a value has been removed from the list. Then we print the popped value at `4` to prove that we still have access to the value that was removed.
The output shows that the value `'suzuki'` was removed from the end of the list and is now stored in the variable `popped_motorcycle`:

```
['honda','yamaha','suzuki']
['honda','yamaha']
suzuki
```

How might this `pop()` method be useful? Imagine that the motorcycles in the list are stored in chronological order according to when we owned them. If this is the case, we can use the `pop()` method to print a statement about the last motorcycle we bought:

```
motorcycles=['honda','yamaha','suzuki']

last_owned=motorcycles.pop()
print("The last motorcycle I owned was a "+last_owned.title()+".")
```

The output is a simple sentence about the most recent motorcycle we owned:

```
The last motorcycle I owned was a Suzuki.
```

Popping Items from any Position in a List

You can actually use `pop()` to remove an item in a list at any position by including the index of the item you want to remove in parentheses.

```
motorcycles=['honda','yamaha','suzuki']

1 first_owned=motorcycles.pop(0)
2 print('The first motorcycle I owned was a'+first_owned.title()+'.')
```

We start by popping the first motorcycle in the list at `1`, and then we print a message about that motorcycle at `2`. The output is a simple sentence describing the first motorcycle I ever owned:

```
The first motorcycle I owned was a Honda.
```

Remember that each time you use `pop()`, the item you work with is no longer stored in the list.
If you're unsure whether to use the del statement or the `pop()` method, here's a simple way to decide: when you want to delete an item from a list and not use that item in any way, use the `del` statement; if you want to use an item as you remove it, use the `pop()` method.

Removing an Item by Value

Sometimes you won't know the position of the value you want to remove from a list. If you only know the value of the item you want to remove, you can use the `remove()` method.
For example, let's say we want to remove the value `'ducati'` from the list of motorcycles.

```
motorcycles=['honda','yamaha','suzuki','ducati']
print(motorcycles)

1 motorcycles.remove('ducati')
  print(motorcycles)
```

The code at `1` tells Python to figure out where `'ducati'` appears in the list and remove that element:

```
['honda','yamaha','suzuki','ducati']
['honda','yamaha','suzuki']
```

You can also use the `remove()` method to work with a value that's being removed from a list. Let's remove the value `'ducati'` and print a reason for removing it from the list:

```
1 motorcycles=['honda','yamaha','suzuki','ducati']
  print(motorcycles)

2 too_expensive='ducati'
3 motorcycles.remove(too_expensive)
  print(motorcycles)
4 print("\nA "+too_expensive.title()+" is too expensive for me.")
```

After defining the list at `1`, we store the value `'ducati'` in a variable called `too_expensive` at `2`. We then use this variable to tell Python which value to remove from the list at `3`. At `4`, the value `'ducati'` has been removed from the list but is still stored in the variable `too_expensive`, allowing us to print a statement about why we removed `'ducati'` from the list of motorcycles:

```
['honda','yamaha','suzuki','ducati']
['honda','yamaha','suzuki']

A Ducati is too expensive for me.
```

> *The `remove()` method deletes only the first occurrence of the value you specify. If there's a possibility the value appears more than once in the list, you'll need to use a loop to determine if all occurrences of the value have been removed.*

Organizing a list

Sorting a List Permanently with the `sort()` Method

Python's `sort()` method makes it relatively easy to sort a list. Imagine we have a list of cars and want to change the order of the list to store them alphabetically. To keep the task simple, let's assume that all the values in the list are lowercase.

```
  cars=['bmw','audi','toyota','subaru']
1 cars.sort()
  print(cars)
```

The `sort()` method, shown at `1`, changes the order of the list permanently. The cars are now in alphabetical order, and we can never revert to the original order:

```
['audi','bmw','subaru','toyota']
```

You can also sort this list in reverse alphabetical order by passing the argument reverse = True to the `sort()` method. The following example sorts the list of cars in reverse alphabetical order:

```
cars=['bmw','audi','toyota','subaru']
```

```
cars.sort(reverse=True)
print(cars)
```

Again, the order of the list is permanently changed:

```
['toyota','subaru','bmw','audi']
```

Sorting a List Temporarily with the `sorted()` Function

To maintain the original order of a list but present it in a sorted order, you can use the `sorted()` function. The `sorted()` function lets you display your list in a particular order but doesn't affect the actual order of the list.
Let's try this function on the list of cars.

```
cars=['bmw','audi','toyota','subaru']

1 print("Here is the original list:")
  print(cars)

2 print("\nHere is the sorted list:")
  print(sorted(cars))

3 print("\nHere is the original list again:")
  print(cars)
```

We first print the list in its original order at `1` and then in alphabetical order at `2`. After the list is displayed in the new order, we show that the list is still stored in its original order at `3`.

```
Here is the original list:
['bmw','audi','toyota','subaru']

Here is the sorted list:
['audi','bmw','subaru','toyota']

Here is the original list again:
4 ['bmw','audi','toyota','subaru']
```

Notice that the list still exists in its original order at `4` after the `sorted()` function has been used. The `sorted()` function can also accept a reverse = True argument if you want to display a list in reverse alphabetical order.

> *Sorting a list alphabetically is a bit more complicated when all the values are not in lowercase. There are several ways to interpret capital letters when you're deciding on a sort order, and specifying the exact order can be more complex than we want to deal with at this time. However, most approaches to sorting will build directly on what you learned in this section.*

Printing a List in Reverse Order

To reverse the original order of a list, you can use the `reverse()` method. If we originally stored the list of cars in chronological order according to when we owned them, we could easily rearrange the list into reverse chronological order:

```
cars=['bmw','audi','toyota','subaru']
print(cars)

cars.reverse()
print(cars)
```

Notice that `reverse()` doesn't sort backward alphabetically; it simply reverses the order of the list:

```
['bmw','audi','toyota','subaru']
['subaru','toyota','audi','bmw']
```

The `reverse()` method changes the order of a list permanently, but you can revert to the original order anytime by applying `reverse()` to the same list a second time.

Finding the Length of a List

You can quickly find the length of a list by using the `len()` function. The list in this example has four items, so its length is 4:

```
>>> cars=['bmw','audi','toyota','subaru']
>>> len(cars)
4
```

You'll find `len()` useful when you need to identify the number of aliens that still need to be shot down in a game, determine the amount of data you have to manage in a visualization, or figure out the number of registered users on a website, among other tasks.

> *Python counts the items in a list starting with one, so you shouldn't run into any off-by-one errors when determining the length of a list.*

Avoiding Index errors when working with lists

One type of error is common to see when you're working with lists for the first time. Let's say you have a list with three items, and you ask for the fourth item.

```
motorcycles=['honda','yamaha','suzuki']
print(motorcycles[3])
```

This example results in an `index error`:

```
Traceback (most recent call last):
```

```
      File "motorcycles.py", line 3, in <module>
           print(motorcycles[3])
IndexError: list index out of range
```

Python attempts to give you the item at index-3. But when it searches the list, no item in motorcycles has an index of 3. Because of the off-by-one nature of indexing in lists, this error is typical. People think the third item is item number 3, because they start counting at 1. But in Python the third item is number 2, because it starts indexing at 0.

An index error means Python can't figure out the index you requested. If an index error occurs in your program, try adjusting the index you're asking for by one. Then run the program again to see if the results are correct.

Keep in mind that whenever you want to access the last item in a list you use the index-1. This will always work, even if your list has changed size since the last time you accessed it:

```
motorcycles=['honda','yamaha','suzuki']
print(motorcycles[-1])
```

The index-1 always returns the last item in a list, in this case the value:

```
'suzuki':'suzuki'
```

The only time this approach will cause an error is when you request the last item from an empty list:

```
motorcycles=[]
print(motorcycles[-1])
```

No items are in motorcycles, so Python returns another index error:

```
Traceback (most recent call last):
      File "motorcyles.py", line 3, in <module>
           print(motorcycles[-1])
IndexError: list index out of range
```

If an index error occurs and you can't figure out how to resolve it, try printing your list or just printing the length of your list. Your list might look much different than you thought it did, especially if it has been managed dynamically by your program. Seeing the actual list, or the exact number of items in your list, can help you sort out such logical errors.

Exercises

1. Store names of 10 of your friends in a list called `names`. Print each person's name by accessing each element in the list, one at a time.

2. Start with the list you just created, but instead of just printing each person's name, print a message which says "`Good morning, have a nice day, <Name>.`", to them. The text of each message should be the same, but each message should be personalized with the person's name.

3. Write a python program that takes a list of numbers (for example, `a = [5, 10, 15, 20, 25]`) and makes a new list of only the first and last elements of the given list.

4. Write a python program that takes a list and returns a new list that contains all the elements of the first list minus all the duplicates.

5. Take a list, say for example this one:

   ```
   a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
   ```

   and write a program that prints out all the elements of the list that are less than 5.

6. Take two lists, say for example these two:

   ```
   a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
   b = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
   ```

   and write a program that returns a list that contains only the elements that are common between the lists (without duplicates). Make sure your program works on two lists of different sizes.

7. Let's say we give you a list saved in a variable

   ```
   a = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
   ```

   Write a python program that takes this list `a` and makes a new list that has only the even elements of this list in it.

8. Write a python program to find the sum of all numbers in a list.

9. Write a python program to find largest number in a given list without using `max()`.

10. Write a python program to find the longest word in a given sentence, and print it.

    ```
    'the punishment assigned to a defendant found guilty by a court, or fixed by law for a particular offence.'
    ```

11. ```
    ["www.zframez.com", "www.wikipedia.org", "www.asp.net",
        "www.abcd.in"]
    ```
    ```

Write a python program to print website suffixes (com, org, net, in) from this list.

12. Consider the following list of cities:

```
cities = ["New York", "Shanghai", "Amsterdam", "Nelson",
"Los Angeles", "Mumbai", "Moscow", "Redmond", "Frankfurt",
"Colombo"]
```

a. Print your list in its original order. Don't worry about printing the list neatly, just print it as a raw Python list.
b. Use `sorted()` to print your list in alphabetical order without modifying the actual list.
c. Show that your list is still in its original order by printing it.
d. Use `sorted()` to print your list in reverse alphabetical order without changing the order of the original list.
e. Show that your list is still in its original order by printing it again.
f. Use `reverse()` to change the order of your list. Print the list to show that its order has changed.
g. Use `reverse()` to change the order of your list again. Print the list to show it's back to its original order.
h. Use `sort()` to change your list so it's stored in alphabetical order. Print the list to show that its order has been changed.
i. Use `sort()` to change your list so it's stored in reverse alphabetical order. Print the list to show that its order has changed.