Week 5: Lists and List Manipulation

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 to 9, or the names of all the people in your family. You can put anything you want into a list, and the items in your list don't have to be related in any particular way. Because 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']

Because this isn't the output you want your users to see, let's learn how to access the individual items in a list.

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']
print(bicycles[0])
```

When we ask for a single item from a list, Python returns just that element without square brackets:

trek

You can also use the string methods title() method on any element in this list. For example, you can format the element 'trek' to look more presentable by using the title() method:

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

Trek

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, ask yourself if you're making a simple but common off-by-one error. The second item in a list has an index of 1. Using this 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. If you ask for the item at index -1, Python always returns the last item in the list:

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

specialized

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 **f-strings** 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']
message = f"My first bicycle was a
{bicycles[0].title()}."
print(message)
```

My first bicycle was a Trek.

Modifying, 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 increase and decrease in length throughout the course of the game.

Modifying Elements in a List

The syntax for modifying an element is similar to the syntax for accessing an element 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, say we have a list of motorcycles and the first item in the list is 'honda'. We can change the value of this first item after the list has been created:

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
motorcycles[0] = 'ducati'
print(motorcycles)
```

['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

The simplest way to add a new element to a list is to append the item to the list. When you append an item to a list, the new element is added to the end of the list. Using the same list 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)
motorcycles.append('ducati')
print(motorcycles)
```

Here the append() method 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() calls. 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']
motorcycles.insert(0, 'ducati')
print(motorcycles)
```

In this example, we insert 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:

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

Removing Elements from a List

Often, you'll want to remove an item or a set of items from a list. For example, when a player shoots down an alien from the sky, you'll most likely want to remove it from the list of active aliens. Or when a user decides to cancel their account on a web application you created, you'll want to remove that user from the list of active users. You can remove an item according to its position in the list or according to its value.

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

Here we use the del statement to remove the first item, 'honda', from the list of motorcycles:
['yamaha', 'suzuki']

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:

```
motorcycles = ['honda', 'yamaha', 'suzuki']#1
print(motorcycles)
popped_motorcycle = motorcycles.pop()#2
print(motorcycles)#3
print(popped_motorcycle)#4
```

We start by defining and printing the list motorcycles 1. Then we pop a value from the list, and assign that value to the variable popped_motorcycle 2. We print the list 3 to show that a value has been removed from the list. Then we print the popped value 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 assigned to the variable popped_motorcycle:

['honda', 'yamaha', 'suzuki']
['honda', 'yamaha']
Suzuki

Popping Items from Any Position in a List

You can use pop() to remove an item from any position in a list by including the index of the item you want to remove in parentheses:

```
motorcycles = ['honda', 'yamaha', 'suzuki']
first_owned = motorcycles.pop(0)
print(f"The first motorcycle I owned was a
{first_owned.title()}.")
```

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

The first motorcycle I owned was a Honda.

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, say we want to remove the value 'ducati' from the list of motorcycles:

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

['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:

```
motorcycles = ['honda', 'yamaha', 'suzuki', 'ducati']#1
print(motorcycles)
too_expensive = 'ducati'#2
motorcycles.remove(too_expensive)#3
print(motorcycles)
print(f"\nA {too_expensive.title()} is too expensive for me.")#4
```

After defining the list 1, we assign the value 'ducati' to a variable called too_expensive 2. We then use this variable to tell Python which value to remove from the list 3. The value 'ducati' has been removed from the list 4 but is still accessible through 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.

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']
print(cars)
cars.sort()
print(cars)
```

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

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

Again, the order of the list is permanently changed:

['bmw', 'audi', 'toyota', 'subaru']
['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']
print("Here is the original list:")#1
print(cars)
print("\nHere is the sorted list:")#2
print(sorted(cars))
print("\nHere is the original list again:")#3
print(cars)
```

We first print the list in its original order 1 and then in alphabetical order 2. After the list is displayed in the new order, we show that the list is still stored in its original order 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:
['bmw', 'audi', 'toyota', 'subaru']

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']

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']
print(len(cars))
```

4