WORKING WITH LISTS IN PYTHON

Working with Lists

Now that we know how to create and access list data, we can start to explore additional ways of working with lists.

In this lesson, you'll learn how to:

- Add and remove items from a list using a specific index.
- Create lists with continuous values.
- Get the length of a list.
- Select portions of a list (called *slicing*).
- Count the number of times that an element appears in a list.
- Sort a list of items.

Note: In some of the exercises, we will be using built-in functions in Python. If you haven't yet explored the concept of a function, it may look a bit new. Below we compare it to the method syntax we learned in the earlier lesson.

Here is a preview:

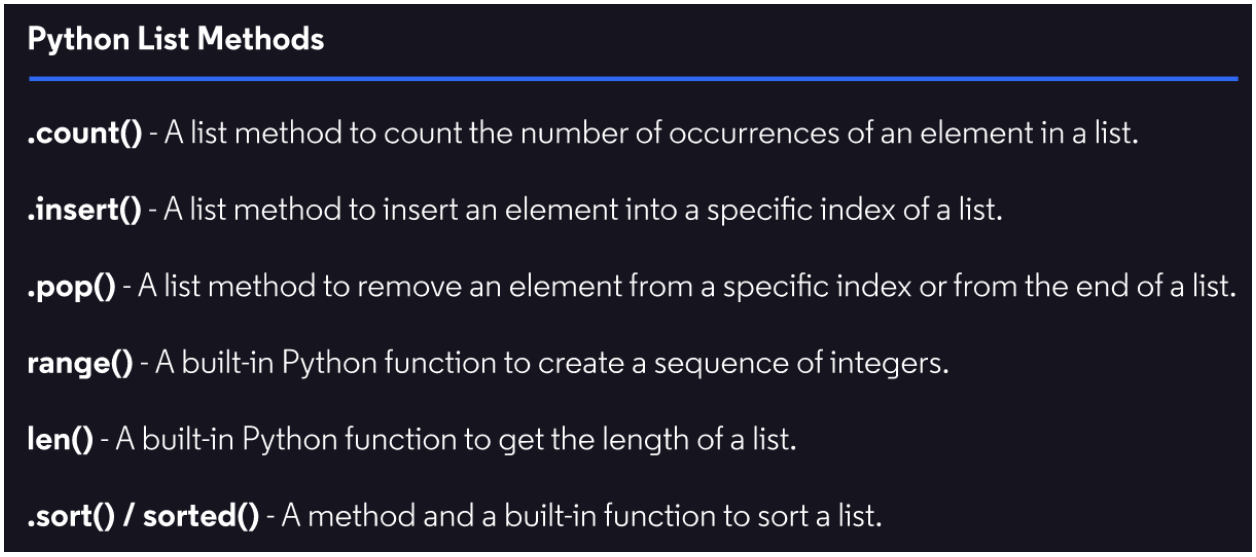
```
# Example syntax for methods
list.method(input)

# Example syntax for a built-in function
builtinfuncion(input)
```

Instructions

Take a second to preview some of the things we will be learning by examining the graphic of common list methods and built-in functions.

When you're ready, continue to the next exercise.



Adding by Index: Insert

The Python list method `.insert()` allows us to add an element to a specific index in a list.

The `.insert()` method takes in two inputs:

1. The index you want to insert into.
2. The element you want to insert at the specified index.

The `.insert()` method will handle shifting over elements and can be used with negative indices.

To see it in action let's imagine we have a list representing a line at a store:

```
store_line = ["Karla", "Maxium", "Martim", "Isabella"]
```

`"Maxium"` saved a spot for his friend `"Vikor"` and we need to adjust the list to add him into the line right behind `"Maxium"`.

For this example, we can assume that `"Karla"` is the front of the line and the rest of the elements are behind her.

Here is how we would use the `.insert()` method to insert `"Vikor"` :

```
store_line.insert(2, "Vikor")
print(store_line)
```

Would output:

```
 ['Karla', 'Maxium', 'Vikor', 'Martim', 'Isabella']
```

Some important things to note:

1. The order and number of the inputs is important. The `.insert()` method expects two inputs, the first being a numerical index, followed by any value as the second input.
2. When we insert an element into a list, all elements from the specified index and up to the last index are shifted one index to the right. This does not apply to inserting an element to the very end of a list as it will simply add an additional index and no other elements will need to shift.

Let's practice using `.insert()`!

Instructions

1.

We are helping out a popular grocery store called Jiho's Produce.

Every week the store has to choose the order in which it displays some of its popular items on sale in the front window to attract customers.

Jiho, the store owner, likes to store the items for the display in a list.

Check out the current display list in our code editor. Click **Run** to print out the list.

2.

Jiho found out some great news! `"Pineapple"` is back in stock.

Jiho would like to put `"Pineapple"` in the front of the list so it is the first item customers see in the display window.

Use `.insert()` to add `"Pineapple"` to the front of the list.

Print the resulting list to see the change.

Note: For this list, the front will be the element at index `0`

Hint

Remember the `.insert()` method takes in two inputs:

1. The index you want to insert into.
2. The element you want to insert at the specified index.

```
list.insert(index, element)
```

script.py

```
front_display_list = ["Mango", "Filet Mignon", "Chocolate Milk"]
print(front_display_list)

# Your code below:
front_display_list.insert(0, 'Pineapple')
print(front_display_list)
```

```
['Mango', 'Filet Mignon', 'Chocolate Milk']
['Pineapple', 'Mango', 'Filet Mignon', 'Chocolate Milk']
```

Removing by Index: Pop

Just as we learned to insert elements at specific indices, Python gives us a method to remove elements at a specific index using a method called `.pop()`.

The `.pop()` method takes an optional single input:

1. The index for the element you want to remove.

To see it in action, let's consider a list called `cs_topics` that stores a collection of topics one might study in a computer science program.

```
cs_topics = ["Python", "Data Structures", "Balloon Making",
"Algorithms", "Clowns 101"]
```

Two of these topics don't look like they belong, let's see how we remove them using `.pop()`.

First let's remove `"Clowns 101"`:

```
removed_element = cs_topics.pop()
print(cs_topics)
print(removed_element)
```

Would output:

```
['Python', 'Data Structures', 'Balloon Making', 'Algorithms']
'Clowns 101'
```

Notice two things about this example:

1. The method can be called without a specific index. Using `.pop()` without an index will remove whatever the last element of the list is. In our case `"Clowns 101"` gets removed.
2. `.pop()` is unique in that it will *return* the value that was removed. If we wanted to know what element was deleted, simply assign a variable to the call of the `.pop()` method. In this case, we assigned it to `removed_element`.

Lastly let's remove `"Balloon Making"`:

```
cs_topics.pop(2)
print(cs_topics)
```

Would output:

```
['Python', 'Data Structures', 'Algorithms']
```

Notice two things about this example:

1. The method can be called with an optional specific index to remove. In our case, the index `2` removes the value of `"Balloon Making"`.

2. We don't have to save the removed value to any variable if we don't care to use it later.

Note: Passing in an index that does not exist or calling `.pop()` on an empty list will both result in an `IndexError`.

Let's apply what we learned about the `.pop()` method.

Instructions

1.
We have decided to pursue the study of data science in addition to our computer science coursework. We signed up for an online school that would help us become proficient.

Check out the current list of topics we will be studying in our code editor.

Click ***Run*** to print out the list.

2.
It looks like we have an overlap with our computer science curriculum for the topic of `"Python 3"`.

Let's remove the topic from the list of `data_science_topics` using our newly learned `.pop()` method.

Print `data_science_topics` to see your result.

Hint
Since `"Python 3"` is the last element in the list we can use the `.pop()` method with no input between the parenthesis to remove the last element.

3.
Looks like there is overlap on the `"Algorithms"` topic as well. Let's use `.pop()` to remove it as well.

Print `data_science_topics` to see the changes.

Hint
Since `"Algorithms"` does not live at the end of the list, we need to use `.pop()` and provide a specific index to the method. For example:

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

# To remove element `2` we would do:
example_list.pop(1)
```

script.py

```
data_science_topics = ["Machine Learning", "SQL", "Pandas", "Algorithms", "Statis
tics", "Python 3"]
print(data_science_topics)

# Your code below:
data_science_topics.pop()
print(data_science_topics)

data_science_topics.pop(3)
print(data_science_topics)
```

```
['Machine Learning', 'SQL', 'Pandas', 'Algorithms',
'Statistics', 'Python 3']
['Machine Learning', 'SQL', 'Pandas', 'Algorithms',
'Statistics']
['Machine Learning', 'SQL', 'Pandas', 'Statistics']
```

Consecutive Lists: Range

Often, we want to create a list of consecutive numbers in our programs. For example, suppose we want a list containing the numbers 0 through 9:

```
my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Typing out all of those numbers takes time and the more numbers we type, the more likely it is that we have a typo that can cause an error.

Python gives us an easy way of creating these types of lists using a built-in function called `range()`.

The function `range()` takes a single input, and generates numbers starting at `0` and ending at the number **before** the input.

So, if we want the numbers from `0` through `9`, we use `range(10)` because `10` is 1 greater than `9`:

```
my_range = range(10)
print(my_range)
```

Would output:

```
range(0, 10)
```

Notice something different? The `range()` function is unique in that it creates a *range object*. It is not a typical list like the ones we have been working with.

In order to use this object as a list, we have to first convert it using another built-in function called `list()`.

The `list()` function takes in a single input for the object you want to convert.

We use the `list()` function on our range object like this:

```
print(list(my_range))
```

Would output:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Let's try out using `range()`!

Instructions

1.

Modify `number_list` so that it is a range containing numbers starting at 0 and up to, but not including, 9.

Hint

Remember the range is non-inclusive of the input number.

If we wanted to generate a range from `0` to `7` we would do:

```
range(8)
```

2.

Create a range called `zero_to_seven` with the numbers 0 through 7.

Print the result in list form.

Hint

Don't forget to convert the range object to a list using the built-in function `list()`. Here is an example:

```
print(list(my_range))
```

script.py

```
# Your code below:

number_list = range(9)
print(list(number_list))
```

```
zero_to_seven = range(8)
print(list(zero_to_seven))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8]
[0, 1, 2, 3, 4, 5, 6, 7]
```

The Power of Range!

By default, `range()` creates a list starting at `0`. However, if we call `range()` with two inputs, we can create a list that starts at a different number.

For example, `range(2, 9)` would generate numbers starting at `2` and ending at `8` (just before `9`):

```
my_list = range(2, 9)
print(list(my_list))
```

Would output:

```
[2, 3, 4, 5, 6, 7, 8]
```

If we use a third input, we can create a list that "skips" numbers.

For example, `range(2, 9, 2)` will give us a list where each number is `2` greater than the previous number:

```
my_range2 = range(2, 9, 2)
print(list(my_range2))
```

Would output:

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

We can skip as many numbers as we want!

For example, we'll start at `1` and skip in increments of `10` between each number until we get to `99` (one before `100`):

```
my_range3 = range(1, 100, 10)
print(list(my_range3))
```

Would output:

```
[1, 11, 21, 31, 41, 51, 61, 71, 81, 91]
```

Our list stops at `91` because the next number in the sequence would be `101`, which is greater than or equal to `100` (our stopping point).

Let's experiment with our additional `range()` inputs!

Instructions

1.

Modify the `range()` function that created the range `range_five_three` such that it:

- Starts at `5`
- Has a difference of `3` between each item
- Ends **before** `15`

Hint

Right now our `range(5, 15, 2)` goes up by steps of `2`. Modify the third input to satisfy the checkpoint.

2.

Create a range called `range_diff_five` that:

- Starts at `0`
- Has a difference of `5` between each item
- Ends **before** `40`

Hint

Remember the input positions for the `range` function and what they represent. For example:

```
range(0, 5, 2)
```

1. The first input represents the starting point. Here that is represented by `0`
2. The second input represents the ending point. Here that is represented by `5`
3. The third input represents in what increments the range will skip as it approaches the ending point. Here that is represented by `2`

script.py

```
# Your code below:

range_five_three = range(5, 15, 3)
print(list(range_five_three))

range_diff_five = range(0, 40, 5)
print(list(range_diff_five))
```

```
[5, 8, 11, 14]
[0, 5, 10, 15, 20, 25, 30, 35]
```

Length

Often, we'll need to find the number of items in a list, usually called its *length*.

We can do this using a built-in function called `len()`.

When we apply `len()` to a list, we get the number of elements in that list:

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

print(len(my_list))
```

Would output:

```
5
```

Let's find the length of various lists!

Instructions

1.

Calculate the length of `long_list` and save it to the variable `long_list_len`.

Hint

To use the `len()` built-in function, pass a list in between the parenthesis `( )`

```
example_list = [1, 5, "Item"]
list_length = len(example_list)
```

2.

Use `print()` to examine `long_list_len`.

Hint

Your output should be `18`.

3.

We have provided a completed `range()` function for the variable `big_range`.

Calculate its length using the function `len()` and save it to a variable called `big_range_length`.

Note: Range objects do not need to be converted to lists in order to determine their length

Hint

We can use the `len()` function to calculate a range's length.

```
# Generates a range of (0, 5)
my_range = range(0, 10, 5)

length_of_range = len(my_range)
```

4.

Use `print()` to examine `big_range_length`.

Hint

Your output should be `300`

5.

Change the `range()` function that generates `big_range` so that it skips `100` instead of `10` steps between items.

How does this change `big_range_length`?

Hint

Changing the third input of a range changes how many elements we skip and thus changing the length of the list. For example:

```
# The list would be [0, 2, 4, 6, 8] with length 5
range(0, 10, 2)

# The list would be [0, 3, 6, 9] with length 4
range(0, 10, 3)
```

script.py

```
long_list = [1, 5, 6, 7, -
23, 69.5, True, "very", "long", "list", "that", "keeps", "going.", "Let's", "prac
tice", "getting", "the", "length"]

big_range = range(2, 3000, 100)

# Your code below:
long_list_len = len(long_list)
print(long_list_len)

big_range_length = len(big_range)
print(big_range_length)
```

Slicing Lists I

In Python, often we want to extract only a portion of a list. Dividing a list in such a manner is referred to as *slicing*.

Lets assume we have a list of `letters`:

```
letters = ["a", "b", "c", "d", "e", "f", "g"]
```

Suppose we want to select from `"b"` through `"f"`.

We can do this using the following syntax: `letters[start:end]`, where:

- `start` is the index of the first element that we want to include in our selection. In this case, we want to start at `"b"`, which has index `1`.
- `end` is the index of *one more than* the last index that we want to include. The last element we want is `"f"`, which has index `5`, so `end` needs to be `6`.

```
sliced_list = letters[1:6]
print(sliced_list)
```

Would output:

```
["b", "c", "d", "e", "f"]
```

Notice that the element at index `6` (which is `"g"`) is *not* included in our selection.

Instructions

1.

Use `print()` to examine the variable `beginning`.

Before hitting ***Run*** think about what elements `beginning` will contain?

2.

Modify `beginning`, so that it selects the first 2 elements of `suitcase`.

Hint

We are looking for the first two elements of our list. This means we want to start at the first element with index `0` and end right after the second element with index `1`.

Remember the ending of our slice is the index of *one more than* the last index that we want to include. For example:

```
elements = [1, 2, 3, 4, 5]
slice = elements[0:3]
print(slice)
```

Would output the first 3 elements since index `3` is not included:

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

3.

Create a new list called `middle` that contains the middle two items (`["pants", "pants"]`) from `suitcase`.

Print `middle` to see the slice!

Hint

There are 6 items in `suitcase`, which means its elements start at index `0` and end at index `5`.

The middle two elements are located at index `2` and `3`, so we want items starting at index `2` and up to, but not including, index `4`.

script.py

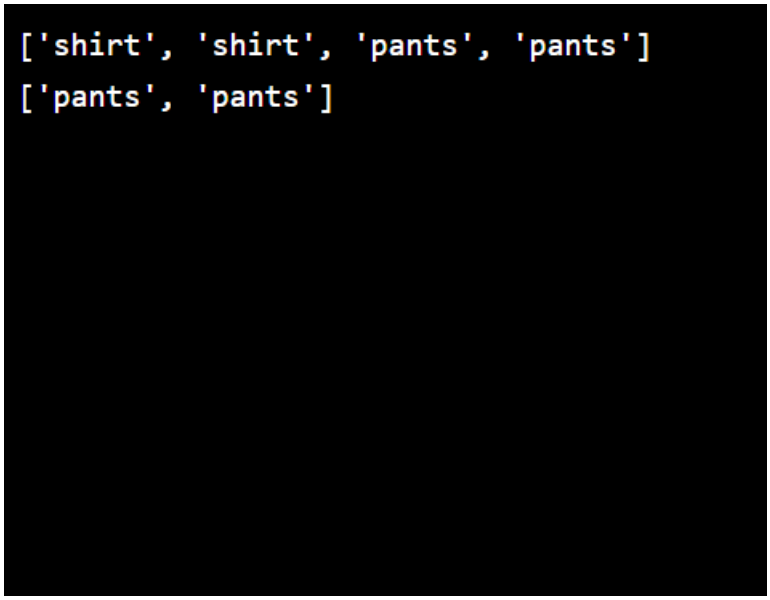
```
suitcase = ["shirt", "shirt", "pants", "pants", "pajamas", "books"]

beginning = suitcase[0:4]

# Your code below:
print(beginning)

beginning = suitcase[0:2]

middle = suitcase[2:4]
print(middle)
```



Slicing Lists II

Slicing syntax in Python is very flexible. Let's look at a few more problems we can tackle with slicing.

Take the list `fruits` as our example:

```
fruits = ["apple", "cherry", "pineapple", "orange", "mango"]
```

If we want to select the *first `n` elements* of a list, we could use the following code:

```
fruits[:n]
```

For our `fruits` list, suppose we wanted to slice the first three elements.

The following code would start slicing from index `0` and up to index `3`. Note that the fruit at index `3` (`orange`) is not included in the results.

```
print(fruits[:3])
```

Would output:

```
['apple', 'cherry', 'pineapple']
```

We can do something similar when we want to slice the *last `n` elements* in a list:

```
fruits[-n:]
```

For our `fruits` list, suppose we wanted to slice the last two elements.

This code slices from the element at index `-2` up through the last index.

```
print(fruits[-2:])
```

Would output:

```
['orange', 'mango']
```

Negative indices can also accomplish taking *all but n last elements* of a list.

```
fruits[:-n]
```

For our `fruits` example, suppose we wanted to slice all but the last element from the list.

This example starts counting from the `0` index up to the element at index `-1`.

```
print(fruits[:-1])
```

Would output:

```
['apple', 'cherry', 'pineapple', 'orange']
```

Let's practice some of these extra slicing techniques!

Instructions

1.

Create a new list called `last_two_elements` containing the final two elements of `suitcase`.

Print `last_two_elements` to see your result.

Hint

To slice the *last `n` elements* in a list:

```
fruits[-n:]
```

If we want to slice off the last two elements, what would `n` be?

2.
Create a new list called `slice_off_last_three` containing all but the last three elements.

Print `slice_off_last_three` to see your result.

Hint
Negative indices can help accomplish taking *all but n last elements* of a list.

```
fruits[:-n]
```

If we want a slice containing all but the last three elements, what would `n` be?

script.py

```
suitcase = ["shirt", "shirt", "pants", "pants", "pajamas", "books"]

# Your code below:
last_two_elements = suitcase[-2:]
print(last_two_elements)

slice_off_last_three = suitcase[:-3]
print(slice_off_last_three)
```

```
['pajamas', 'books']
['shirt', 'shirt', 'pants']
```

Counting in a List

In Python, it is common to want to count occurrences of an item in a list.

Suppose we have a list called `letters` that represents the letters in the word "Mississippi":

```
letters = ["m", "i", "s", "s", "i", "s", "s", "i", "p", "p", "i"]
```

If we want to know how many times `i` appears in this word, we can use the list method called `.count()`:

```
num_i = letters.count("i")
print(num_i)
```
Would output:

```
4
```
Notice that since `.count()` *returns* a value, we can assign it to a variable to use it.

We can even use `.count()` to count element appearances in a two-dimensional list.

Let's use the list `number_collection` as an example:

```
number_collection = [[100, 200], [100, 200], [475, 29], [34, 34]]
```
If we wanted to know how often the sublist `[100, 200]` appears:

```
num_pairs = number_collection.count([100, 200])
print(num_pairs)
```
Would output:

```
2
```
Let's count some list items using the `.count()` method!

Instructions

1.
Mrs. Wilson's class is voting for class president. She has saved each student's vote into the list `votes`.

Use `.count()` to determine how many students voted for `"Jake"` and save the value to a variable called `jake_votes`.

Hint
If we wanted to know how many students voted for `"Laurie"`, we'd use:

```
votes.count("Laurie")
```
2.
Use `print()` to examine `jake_votes`.

script.py

```
votes = ["Jake", "Jake", "Laurie", "Laurie", "Laurie", "Jake", "Jake", "Jake", "L
aurie", "Cassie", "Cassie", "Jake", "Jake", "Cassie", "Laurie", "Cassie", "Jake",
 "Jake", "Cassie", "Laurie"]


# Your code below:
```

```
jake_votes = votes.count("Jake")
print(jake_votes)
```

script.py

```
votes = ["Jake", "Jake", "Laurie", "Laurie", "Laurie", "Jake", "Jake", "Jake", "L
aurie", "Cassie", "Cassie", "Jake", "Jake", "Cassie", "Laurie", "Cassie", "Jake",
 "Jake", "Cassie", "Laurie"]

# Your code below:
jake_votes = votes.count("Jake")
print(jake_votes)
```

Sorting Lists I

Often, we will want to sort a list in either numerical (1, 2, 3, ...) or alphabetical (a, b, c, ...) order.

We can sort a list using the method `.sort()`.

Suppose that we have a list of names:

```
names = ["Xander", "Buffy", "Angel", "Willow", "Giles"]
```

Let's see what happens when we apply `.sort()`:

```
names.sort()
print(names)
```

Would output:

```
['Angel', 'Buffy', 'Giles', 'Willow', 'Xander']
```

As we can see, the `.sort()` method sorted our list of `names` in alphabetical order.

`.sort()` also provides us the option to go in reverse. Instead of sorting in ascending order like we just saw, we can do so in descending order.

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

Would output:

```
['Xander', 'Willow', 'Giles', 'Buffy', 'Angel']
```

Note: The `.sort()` method does not return any value and thus does not need to be assigned to a variable since it modifies the list directly. If we do assign the result of the method, it would assign the value of `None` to the variable.

Let's experiment sorting various lists!

Instructions

1.

Use `.sort()` to sort `addresses`.

Hint

Like with any list method, attach the method to the list you want to apply it on.

```
list.sort()
```

2.

Use `print()` to see how `addresses` changed.

3.

Remove the `#` and whitespace in front of the line `sort(names)`. Edit this line so that it runs without producing a `NameError`.

Hint

Remember, `sort` comes *after* the list:

```
my_list.sort()
```

If you receive an `IndentationError` make sure to check you removed the whitespace after the `#`.

4.

Use `print` to examine `sorted_cities`. Why is it not the sorted version of `cities`?

Hint

The `.sort()` method does not return any value and thus does not need to be assigned to a variable. This is why we will see the value of `None` as our output for `sorted_cities`

5.

Edit the `.sort()` call on `cities` such that it sorts the cities in reverse order (descending).

Print `cities` to see the result.

Hint

To reverse a list using `.sort()`, add an optional input keyword argument and assign it the value of `True`.

```
list.sort(reverse=True)
```

script.py

```
# Checkpoint 1 & 2
addresses = ["221 B Baker St.", "42 Wallaby Way", "12 Grimmauld Place", "742 Ever
green Terrace", "1600 Pennsylvania Ave", "10 Downing St."]
addresses.sort()
print(addresses)

# Checkpoint 3
names = ["Ron", "Hermione", "Harry", "Albus", "Sirius"]
names.sort()


# Checkpoint 4 & 5
cities = ["London", "Paris", "Rome", "Los Angeles", "New York"]
sorted_cities = cities.sort(reverse=True)
print(sorted_cities)
```

```
['10 Downing St.', '12 Grimmauld Place', '1600
Pennsylvania Ave', '221 B Baker St.', '42 Wallaby Way',
'742 Evergreen Terrace']
None
```

Sorting Lists II

A second way of sorting a list in Python is to use the built-in function `sorted()`.

The `sorted()` function is different from the `.sort()` method in two ways:

1. It comes *before* a list, instead of after as all built-in functions do.
2. It generates a new list rather than modifying the one that already exists.

Let's return to our list of names:

```
names = ["Xander", "Buffy", "Angel", "Willow", "Giles"]
```

Using `sorted()`, we can create a new list, called `sorted_names`:

```
sorted_names = sorted(names)
print(sorted_names)
```

This yields the list sorted alphabetically:

```
['Angel', 'Buffy', 'Giles', 'Willow', 'Xander']
```

Note that using `sorted` did not change `names`:

```
print(names)
```

Would output:

```
['Xander', 'Buffy', 'Angel', 'Willow', 'Giles']
```

Instructions

1.

Use `sorted()` to order `games` and create a new list called `games_sorted`.

Hint

As with any built-in function, pass the list you wish to be sorted in between the parenthesis `( )` of the function.

```
sorted(list)
```

2.

Print both `games` and `games_sorted`. How are they different?

Hint

In contrast to the method `.sort()`, the built-in function `sorted()` will not modify the original list.

script.py

```
games = ["Portal", "Minecraft", "Pacman", "Tetris", "The Sims", "Pokemon"]

# Your code below:
games_sorted = sorted(games)
print(games)
print(games_sorted)
```

```
['Portal', 'Minecraft', 'Pacman', 'Tetris', 'The Sims',
'Pokemon']
['Minecraft', 'Pacman', 'Pokemon', 'Portal', 'Tetris',
'The Sims']
```

Review
In this lesson, we learned how to:

- Add elements to a list by index using the `.insert()` method.
- Remove elements from a list by index using the `.pop()` method.
- Generate a list using the `range()` function.
- Get the length of a list using the `len()` function.
- Select portions of a list using slicing syntax.
- Count the number of times that an element appears in a list using the `.count()` method.
- Sort a list of items using either the `.sort()` method or `sorted()` function.

As you go through the exercises, feel free to use `print()` to see changes when not explicitly asked to do so.

Instructions

1.
Our friend Jiho has been so successful in both the flower and grocery business that she has decided to open a furniture store.

Jiho has compiled a list of inventory items into a list called `inventory` and wants to know a few facts about it.

First, how many items are in the warehouse?

Save the answer to a variable called `inventory_len`.

Hint

Use `len` to find the number of items in `inventory`

2.

Select the first element in `inventory`. Save it to a variable called `first`.

Hint

If we wanted to select the second element in `inventory`, we would use:

```
second = inventory[1]
```

Remember that Python lists are *zero-indexed*.

3.

Select the last element from `inventory`. Save it to a variable called `last`.

Hint

The last item has index `-1`.

4.

Select items from the `inventory` starting at index `2` and up to, but not including, index `6`.

Save your answer to a variable called `inventory_2_6`.

Hint

If we wanted to select items from the `inventory` starting at index `5` and up to, but not including, index `7`, we would use:

```
inventory_5_7 = inventory[5:7]
```

5.

Select the first 3 items of `inventory`. Save it to a variable called `first_3`.

Hint

If we wanted to select the first 5 items from `inventory`, we would use:

```
inventory[:5]
```

6.

How many `'twin bed'`s are in `inventory`? Save your answer to a variable called `twin_beds`.

Hint

If we wanted to know how many `'dresser'`s were in `inventory`, we would use:

```
num_dressers = inventory.count('dresser')
```

7.

Remove the 5th element in the inventory. Save the value to a variable called `removed_item`.

Hint

Since lists in Python of zero_indexed, the 5th element will be at index `4`.

Use the `.pop()` method to remove elements from a list by index.

8.

There was a new item added to our inventory called `"19th Century Bed Frame"`.

Use the `.insert()` method to place the new item as the 11th element in our inventory.

Hint

Since lists in Python of zero_indexed, the 11th element will be at index `10`.

9.

Sort `inventory` using the `.sort()` method or the `sorted()` function.

Remember, the `sorted()` function doesn't change the original list — it creates a new list with the elements properly sorted. If you use `sorted()` you'll have to set `inventory` equal to the value returned by `sorted()`.

Print `inventory` to see the result.

Hint

As a reminder here is how you might use `.sort()` or `sorted()`

```
# Using sort
list.sort()

# Using sorted
list = sorted(list)
```

script.py

```
inventory = ["twin bed", "twin bed", "headboard", "queen bed", "king bed", "dresser", "dresser", "table", "table", "nightstand", "nightstand", "king bed", "king bed", "twin bed", "twin bed", "sheets", "sheets", "pillow", "pillow"]

inventory_len = len(inventory)
first = inventory[0]
last = inventory[-1]
inventory_2_6 = inventory[2:6]
first_3 = inventory[:3]
twin_beds = inventory.count("twin bed")
removed_item = inventory.pop(4)
inventory.insert(10, "19th Century Bed Frame")
inventory.sort()
print(inventory)
```

```
['19th Century Bed Frame', 'dresser', 'dresser',
'headboard', 'king bed', 'king bed', 'nightstand',
'nightstand', 'pillow', 'pillow', 'queen bed', 'sheets',
'sheets', 'table', 'table', 'twin bed', 'twin bed',
'twin bed', 'twin bed']
```