List Methods

List methods are functions that are built into Python that allow us to modify, sort, and do things with lists. We're going to take a look at some of the most common list methods and how we can use them.

Why is this important?

We know now how to create lists, how to get a part of a list, how to modify items in them and how to get the length of list. These are important, but we're not changing the list.

In real life we want to be able to add items in a list, remove items in a list, and sort items in a list. We're going to take a look at how we can do that.

You might be thinking "Where is this actually used"?

- Anything that is a table or layout with multiple items is essentially a list or list of lists so if you get anything say from an external database this would be a list.
- Anything that uses data science, artificial intelligence, and machine learning uses lists heavily to store data and train models.

What are we going to do?

Steps

1. Let's create a file called `list_methods.py`

2. Let's create a list of student grades

We want to do some deeper analysis of our student grades with python, so we're going to create a list of student grades.

- using the `[]` brackets, we can create a list of items that are separated by commas. Like we've done before.

```
student_grades = [88, 65, 91, 72, 98]
print("Our student grades are: ")
print(student_grades)
```

- Note here that we're using numbers instead of strings. We can use any data type in a list but the fundamentals of lists stay the same.

3. Let's sort our list of student grades.

We want to list the student from highest to lowest grade we can use the `sort` method to do that. The method `sort` is built into the python language when you're using lists.

- How we do this is we add the `.` after the variable name and then we add the method name after that. We also have to include the `()` brackets after the method name. to actually call the function.

```
student_grades = [88, 65, 91, 72, 98]


print("Our student grades are: ")
print(student_grades)


print("Our student grades sorted are: ")
student_grades.sort()
print(student_grades)
```

- The output should look like the following

```
$ python list_methods.py
Our student grades are:
[88, 65, 91, 72, 98]
Our student grades sorted are:
[65, 72, 88, 91, 98]
```

- Note that have modified the existing list order to be sorted from lowest to highest!
  - Note there's a standalone method called `sorted` that does the same thing but it returns a new list instead of modifying the existing list.
    - You would use it like this `sorted(student_grades)` we'll try it out.
- Note that we've listed our `student_grades` from lowest to highest, but we want to list them from highest to lowest. There's a couple of ways to achieve this.
  - we could use the `reverse` method on the list.
    - we would do this by adding the `.` after the variable name and then adding the method name after that. We also have to include the `()` brackets after the method name. to actually call the function.
    - we would do this like this `student_grades.reverse()`
  - or we could change the `sort` method to `sort(reverse=True)`
    - this would sort the list in reverse order.
    - we would do this like this `student_grades.sort(reverse=True)`
  - let's try both of these out.

```
student_grades = [88, 65, 91, 72, 98]

print("Our student grades are: ")
print(student_grades)

print("Our student grades sorted are: ")
student_grades.sort(reverse=True)
print(student_grades)
```

4. We forgot a few grades, so we're going to add them to the list.

We forgot to add a grade to the list, so we're going to add it to the list. We can do this by using the `append` method. The way to use the `append` method is to add the `.` after the variable name and then add the method name after that. We also have to include the `()` brackets after the method name with the item that we want to add to the list in the `()` brackets.

- Let's add `100` to the list.

```
student_grades = [88, 65, 91, 72, 98]

print("Our student grades are: ")
print(student_grades)

print("Our student grades sorted are: ")
student_grades.sort(reverse=True)
print(student_grades)

# Let's add 100 to the student grades.
student_grades.append(100)
print("After we've added grades, the new list is: ")
print(student_grades)
```

- You can see the output is as follows:

```
$ python list_methods.py
Our student grades are:
[88, 65, 91, 72, 98]
Our student grades sorted are:
[98, 91, 88, 72, 65]
After we've added grades, the new list is: :
[98, 91, 88, 72, 65, 100]
```

- Note here that we've added `100` to the end of the list.
    - `append` always adds to the end of the list.
    - note as well that the list is no longer sorted perfectly (you can fix this by calling the sort method)
- What if we wanted to add `50` at index `2` (as the third item) in the list?
    - we can do this by using the `insert` method.
    - the way to use the `insert` method is to add the `.` after the variable name and then add the method name after that. We also have to include the `()` brackets after the method name with the index that we want to insert the item at and the item that we want to insert in the `()` brackets.
    - let's try it out.

```
student_grades = [88, 65, 91, 72, 98]

# code removed for brevity

# Let's add 100 to the student grades.
student_grades.append(100)
student_grades.insert(2,50) # inserted at index 2
print("After we've added grades, the new list is: ")
print(student_grades)
```

- the output should look as follows

```
$ python list_methods.py
Our student grades are:
[88, 65, 91, 72, 98]
Our student grades sorted are:
[98, 91, 88, 72, 65]
After we've added grades, the new list is:
[98, 91, 50, 88, 72, 65, 100]
```

- So in the output, you can see that `50` is now at index `2` (the third item) in the list and you can see that `100` was added to the end of the list.

5. We're going to remove some grades from the list.

Just as we can add to a list we can also remove. There are a couple of ways to remove items from a list, specifically by index with `pop` and by value with `remove`, we're going to take a look at both of these.

- Let's remove the fourth item from the list.
    - we can do this by using the `pop` method.

    - the way to use the `pop` method is to add the `.` after the variable name and then add the method name after that. We also have to include the `()` brackets after the method name with the index that we want to remove the item at, in the `()` brackets.
    - let's try it out.

```
# other code removed for brevity

# Let's remove a few items from the list.
student_grades.pop(3) # remove at index 2

print("After we've removed grades, the new list is: ")
print(student_grades)
```

- the output should look as follows

```
$ python list_methods.py
Our student grades are:
[88, 65, 91, 72, 98]
Our student grades sorted are:
[98, 91, 88, 72, 65]
After we've added grades, the new list is:
[98, 91, 50, 88, 72, 65, 100]
After we've removed grades, the new list is:
[98, 91, 50, 72, 65, 100]
```

- You can see at the end the fourth item in the list or item at index `3` was removed from the list (this was the `88` in the list)

    - So to use the pop method you can either specify the index or not.
        - if you don't specify the index it will remove the last item in the list. Usage is `student_grades.pop()` here would remove `100` from the list.
        - if you do specify the index it will remove the item at that index. We used `student_grades.pop(3)` to remove `88` from the list.

- Now you might be thinking to yourself what if I don't know the index of the item that I want to remove, well you can use the `remove` method.

    - the way to use the `remove` method is to add the `.` after the variable name and then add the method name after that. We also have to include the `()` brackets after the method name with the value that we want to remove in the `()` brackets.
    - let's try it out by removing the `98` from the list.

```
# other code removed for brevity


# Let's remove a few items from the list.
student_grades.pop(3) # remove at index 2
student_grades.remove(98) # remove the value 98 from the list.
print("After we've removed grades, the new list is: ")
print(student_grades)
```

- You should see the output as follows

```
$ python list_methods.py
Our student grades are:
[88, 65, 91, 72, 98]
Our student grades sorted are:
[98, 91, 88, 72, 65]
After we've added grades, the new list is:
[98, 91, 50, 88, 72, 65, 100]
After we've removed grades, the new list is:
[91, 50, 72, 65, 100]
```

- so now you can see that you can remove an item from the list by value using `.remove(value)`.

- What if you try to remove something that isn't there?

    - try to remove `9001` from the list.
    - What you're going to see is a `ValueError` exception, we're going to learn how to handle this in the coming week.

6. How do we check if something is `in` the list?

One common thing you're going to do is check if something is in a list. You can do this by using the `in` operator (pretty handy naming if you ask me). The `in` operator returns a boolean, either `True` if it's in the list or `False` if it is not.

- Let's check if a student a got a perfect grade (or `100`) in our list of `student_grades`.

```
# other code removed for brevity
print("Has a student got a perfect grade?")
print(100 in student_grades)
```

- You should see the output as follows

```
$ python list_methods.py
Our student grades are:
[88, 65, 91, 72, 98]
Our student grades sorted are:
[98, 91, 88, 72, 65]
After we've added grades, the new list is:
[98, 91, 50, 88, 72, 65, 100]
After we've removed grades, the new list is:
[91, 50, 72, 65, 100]
Has a student got a perfect grade?
True
```

- Great you can see that we got `True` back, so we know that there is a perfect grade in the list.
- Let's check if a student mailed it in and got a `0` in the course

```
# other code removed for brevity
print("Has a student got a perfect grade?")
print(100 in student_grades)
print("Has a student gotten a 0 in the course?")
print(0 in student_grades)
```

- You should see the output as follows

```
$ python list_methods.py
Our student grades are:
[88, 65, 91, 72, 98]
Our student grades sorted are:
[98, 91, 88, 72, 65]
After we've added grades, the new list is:
[98, 91, 50, 88, 72, 65, 100]
After we've removed grades, the new list is:
[91, 50, 72, 65, 100]
Has a student got a perfect grade?
True
Has a student gotten a 0 in the course?
False
```

- Great! You can see that we got `False` back, so we know that there is no `0` in the list.
- We're going to use this a lot with if statements so that we can execute some code if something is in the list or not.
- This is also a really good way to try to avoid the `ValueError` when removing items from a list.

Conclusion

We've learned how to use some of the most common list methods in python. We've learned how to sort a list, how to add items to a list, how to remove items from a list, and how to check if something is in a list. There are many more things we can do with lists, but this is a great starting point.