# List Functions

# List Functions

There are various different actions that we can take on a list once it's been created. Each of these actions can be taken by calling a function on the list!

Most of these list functions will change the contents of the list you call them on, without returning a value.

Here's a list *(ha)* of the functions we're gonna cover today:

- .append()
- .extend()
- .insert()
- .sort()
- .reverse()
- .count()
- .remove()
- .index()
- .pop()

# .append()

Parameter(s): A new element

What it do: Adds the new element to the end of the list.

Returns: None

```
my_list = [1, 2, 3, 4, 5]
print(my_list)
my_list.append(6)
print(my_list)
```

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

# .append()

Parameter(s): A new element

What it do: Adds the new element to the end of the list.

Returns: None

```
my_list = [1, 2, 3, 4, 5]
print(my_list)
my_list.append(6)
print(my_list)
```

```
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5, 6]
```

# Appending a list

If we use a list as a parameter to the .append() function, what will we get?

```
my_list = [1, 2, 3, 4, 5]
print(my_list)
my_list.append([6, 7])
print(my_list)
```

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

# Appending a list

If we use a list as a parameter to the .append() function, what will we get?

```
my_list = [1, 2, 3, 4, 5]
print(my_list)
my_list.append([6, 7])
print(my_list)
```

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

# .extend()

Parameter(s): A list value

What it do: Adds the parameter list to the end of the original (similar to concatenation with strings).

Returns: None

```
my_list = [1, 2, 3]
print(my_list)
my_list.extend([6, 7, 8])
print(my_list)
```

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

# .extend()

Parameter(s): A list value

What it do: Adds the parameter list to the end of the original (similar to concatenation with strings).

Returns: None

```
my_list = [1, 2, 3]
print(my_list)
my_list.extend([6, 7, 8])
print(my_list)
```

```
[1, 2, 3]
[1, 2, 3, 6, 7, 8]
```

# .extend() part deux

We can create the exact same behavior as the extend() function if we use the + operator.

We can concatenate lists in the exact same way as we can strings and tuples!

```
my_list = [1, 2, 3]
print(my_list)
my_list += [6, 7, 8]
print(my_list)
```

```
[1, 2, 3]
[1, 2, 3, 6, 7, 8]
```

# .insert()

Parameter(s): index, new element.

What it do: Puts the provided element into the list at the given index. All elements currently at or after that index are shifted back.

Returns: None

```
my_list = [1, 2, 3, 4, 5]
print(my_list)
my_list.insert(2, 10)
print(my_list)
```

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

# .insert()

Parameter(s): index, new element.

What it do: Puts the provided element into the list at the given index. All elements currently at or after that index are shifted back.

Returns: None

```
my_list = [1, 2, 3, 4, 5]
print(my_list)
my_list.insert(2, 10)
print(my_list)
```

```
[1, 2, 3, 4, 5]
[1, 2, 10, 3, 4, 5]
```

# .sort()

Parameter(s): *(optional)* reverse = True/False. Default is False.

What it do: Puts everything in the list into order. If list contains strings, sorts alphabetically.

Returns: None

```
my_list = [8, 6, 2, 4, 9]
print(my_list)
my_list.sort()
print(my_list)
my_list.sort(reverse = True)
print(my_list)
```

```
[8, 6, 2, 4, 9]
```

# .sort()

Parameter(s): *(optional)* reverse = True/False. Default is False.

What it do: Puts everything in the list into order. If list contains strings, sorts alphabetically.

Returns: None

```
my_list = [8, 6, 2, 4, 9]
print(my_list)
my_list.sort()
print(my_list)
my_list.sort(reverse = True)
print(my_list)
```

```
[8, 6, 2, 4, 9]
[2, 4, 6, 8, 9]
```

# .sort()

Parameter(s): *(optional)* reverse = True/False. Default is False.

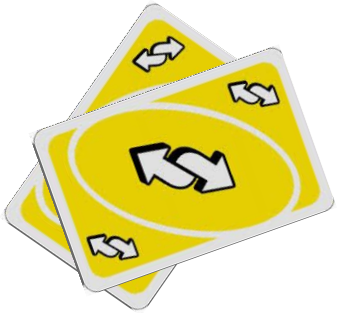
What it do: Puts everything in the list into order. If list contains strings, sorts alphabetically.

Returns: None

```
my_list = [8, 6, 2, 4, 9]
print(my_list)
my_list.sort()
print(my_list)
my_list.sort(reverse = True)
print(my_list)
```

```
[8, 6, 2, 4, 9]
[2, 4, 6, 8, 9]
[9, 8, 6, 4, 2]
```

# .reverse()



Parameter(s): N/A

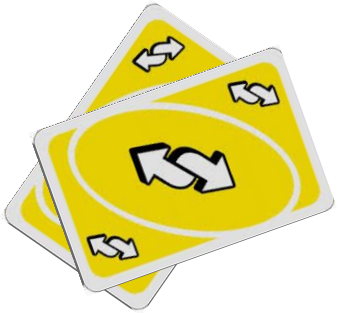
What it do: Flips the list around, back to front.

Returns: None

```
my_list = [8, 6, 2, 4, 9]
print(my_list)
my_list.reverse()
print(my_list)
```

```
[8, 6, 2, 4, 9]
```

# .reverse()



Parameter(s): N/A

What it do: Flips the list around, back to front.

Returns: None

```
my_list = [8, 6, 2, 4, 9]
print(my_list)
my_list.reverse()
print(my_list)
```

```
[8, 6, 2, 4, 9]
[9, 4, 2, 6, 8]
```

# .count()

Parameter(s): Element value

What it do: Counts the number of times the element provided can be found in the list.

Returns: Number of times found

```
my_list = [1, 2, 3, 2, 3]
print(my_list.count(2))
print(my_list.count(50))
```

# .count()

Parameter(s): Element value

What it do: Counts the number of times the element provided can be found in the list.

Returns: Number of times found

```
my_list = [1, 2, 3, 2, 3]
print(my_list.count(2))
print(my_list.count(50))
```

```
2
```

# .count()

Parameter(s): Element value

What it do: Counts the number of times the element provided can be found in the list.

Returns: Number of times found

```
my_list = [1, 2, 3, 2, 3]
print(my_list.count(2))
print(my_list.count(50))
```

```
2
0
```

# .remove()

Parameter(s): The value of an element in the list.

What it do: Finds the first copy of that element, then removes it from the list. Everything after it slides down by 1 index.

Returns: None

```
my_list = [3, 2, 3, 4, 5]
print(my_list)
my_list.remove(3)
print(my_list)
```

```
[3, 2, 3, 4, 5]
```

# .remove()

Parameter(s): The value of an element in the list.

What it do: Finds the first copy of that element, then removes it from the list. Everything after it slides down by 1 index.

Returns: None

```
my_list = [3, 2, 3, 4, 5]
print(my_list)
my_list.remove(3)
print(my_list)
```

```
[3, 2, 3, 4, 5]
[2, 3, 4, 5]
```

# .remove() part deux

One warning for use with the .remove() function:

**If the value isn't in the list your code will crash.**

You'll get an error that looks a little something like this: list.index(x): x not in list

Let's talk about what the .index() function does!

# .index()

Parameter(s): The value of an element in the list, *(optional)* starting index, *(optional)* ending index

What it do: Finds the first copy of that element

Returns: The index where the element can be found.

```
my_list = [3, 2, 3, 4, 5]
print(my_list)
print(my_list.index(3))
```

```
[3, 2, 3, 4, 5]
```

# .index()

Parameter(s): The value of an element in the list, *(optional)* starting index, *(optional)* ending index

What it do: Finds the first copy of that element

Returns: The index where the element can be found.

```
my_list = [3, 2, 3, 4, 5]
print(my_list)
print(my_list.index(3))
```

```
[3, 2, 3, 4, 5]
0
```

# .index() 2

The .index() function is almost identical the the .find() function with strings. They take similar parameters, and will return similar things. There is a major difference though:

.find() will return -1 if the value is not found.

.index() will cause a `ValueError` exception to occur when the value is not found, crashing your program.

If we add a `count(element)` before we try to `index()` or `remove()` that element, we can ensure we aren't going to be accessing something that isn't in our list.

# .pop()

Parameter(s): The index of an element in your list.

What it do: Removes the element at the specified index from the list. Everything behind slides down by 1 index.

Returns: The value of the removed element.

```
my_list = [1, 2, 3, 4, 5]
print(my_list)
print(my_list.pop(2))
print(my_list)
```

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

# .pop()

Parameter(s): The index of an element in your list.

What it do: Removes the element at the specified index from the list. Everything behind slides down by 1 index.

Returns: The value of the removed element.

```
my_list = [1, 2, 3, 4, 5]
print(my_list)
print(my_list.pop(2))
print(my_list)
```

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

# .pop()

Parameter(s): The index of an element in your list.

What it do: Removes the element at the specified index from the list. Everything behind slides down by 1 index.

Returns: The value of the removed element.

```
my_list = [1, 2, 3, 4, 5]
print(my_list)
print(my_list.pop(2))
print(my_list)
```

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