INTRODUCTION TO LISTS

What is a List?

In programming, it is common to want to work with collections of data. In Python, a *list* is one of the many built-in data structures that allows us to work with a collection of data in sequential order.

Suppose we want to make a list of the heights of students in a class:

- Noelle is 61 inches tall
- Ava is 70 inches tall
- Sam is 67 inches tall
- Mia is 64 inches tall

In Python, we can create a variable called `heights` to store these integers into a *list*:

```
heights = [61, 70, 67, 64]
```

Notice that:

1. A list begins and ends with square brackets (`[` and `]`).
2. Each item (i.e., `67` or `70`) is separated by a comma (`,`)
3. It's considered good practice to insert a space () after each comma, but your code will run just fine if you forget the space.

Let's write our own list!

Instructions

1.

Examine the existing list `heights` in your code editor.

A new student just joined the class!

- Chloe is 65 inches tall

Add Chloe's height to the end of the list `heights`.

Hint

When adding a new element to a list, don't forget the comma to separate the values!

2.

Remove the `#` in front of the definition of the list `broken_heights`. If you run this code, you'll get an error in your terminal:

```
SyntaxError: invalid syntax
```

Add commas (`,`) to `broken_heights` so that it runs without errors.

Hint

Did you add commas (,) between each element?

script.py

```
heights = [61, 70, 67, 64, 65]


broken_heights = [65, 71, 59, 62]
```

What can a List contain?

Lists can contain more than just numbers.

Let's revisit our classroom example with heights:

- Noelle is 61 inches tall
- Ava is 70 inches tall
- Sam is 67 inches tall
- Mia is 64 inches tall

Instead of storing each student's height, we can make a list that contains their names:

```
names = ["Noelle", "Ava", "Sam", "Mia"]
```

We can even combine multiple data types in one list. For example, this list contains both a string and an integer:

```
mixed_list_string_number = ["Noelle", 61]
```

Lists can contain any data type in Python! For example, this list contains a string, integer, boolean, and float.

```
mixed_list_common = ["Mia", 27, False, 0.5]
```

Let's experiment with different data types in our own lists!

Instructions

1.

Add any additional string to the end of the list `ints_and_strings`.

Hint

Remember that a string has quotes (`'`) or (`"`) at the beginning and the end.

2.

Create a new list called `sam_height_and_testscore` that contains:

1. The string `"Sam"` (to represent Sam's name)
2. The number `67` (to represent Sam's height)
3. The float `85.5` (to represent Sam's score)
4. The boolean `True` (to represent Sam passing the test)

Make sure to write the elements in exact order.

Hint

Remember the key components of a list:

1. A list begins and ends with square brackets (`[` and `]`).
2. Each item (i.e., `67` or `70`) is separated by a comma (`,`)

```
practice_list = ['String', 6, 0.5, True]
```

script.py

```
ints_and_strings = [1, 2, 3, "four", "five", "six"]

sam_height_and_testscore = ["Sam", 67, 85.5, True]
```

Empty Lists

A list doesn't have to contain anything. You can create an empty list like this:

```
empty_list = []
```

Why would we create an empty list?

Usually, it's because we're planning on filling it up later based on some other input. We'll talk about two ways of filling up a list in the next exercise.

Let's practice writing an empty list!

Instructions

1.

Create an empty list and call it `my_empty_list`. Don't put anything in the list just yet.

Hint

Remember that a list begins and ends with square brackets (`[` and `]`) and does not have to contain any elements.

script.py

```
my_empty_list = []
```

List Methods

As we start exploring lists further in the next exercises, we will encounter the concept of a *method*.

In Python, for any specific data-type (strings, booleans, lists, etc.) there is built-in functionality that we can use to create, manipulate, and even delete our data. We call this built-in functionality a method.

For lists, methods will follow the form of `list_name.method()`. Some methods will require an input value that will go between the parenthesis of the method `( )`.

An example of a popular list method is `.append()`, which allows us to add an element to the end of a list.

```
append_example = [ 'This', 'is', 'an', 'example']
append_example.append('list')

print(append_example)
```

Will output:

```
['This', 'is', 'an', 'example', 'list']
```

Instructions

We will be exploring `.append()` and many other methods in the upcoming exercises but for now take a second to examine and play around with the code for two common list methods.

script.py

```
example_list = [1, 2, 3, 4]


#Using Append
example_list.append(5)
print(example_list)


#Using Remove
example_list.remove(5)
print(example_list)
```

Growing a List: Append

We can add a single element to a list using the `.append()` Python method.

Suppose we have an empty list called `garden`:

```
garden = []
```

We can add the element `"Tomatoes"` by using the `.append()` method:

```
garden.append("Tomatoes")

print(garden)
```

Will output:

```
['Tomatoes']
```

We see that `garden` now contains `"Tomatoes"`!

When we use `.append()` on a list that already has elements, our new element is added to the **end** of the list:

```
# Create a list
garden = ["Tomatoes", "Grapes", "Cauliflower"]

# Append a new element
garden.append("Green Beans")
print(garden)
```

Will output:

```
['Tomatoes', 'Grapes', 'Cauliflower', 'Green Beans']
```

Let's use the `.append()` method to manipulate a list.

Instructions

1.

Jiho works for a gardening store called Petal Power. Jiho keeps a record of orders in a list called `orders`.

Use `print` to inspect the orders he has received today.

Hint

You should see:

```
['daisies', 'periwinkle']
```

2.

Jiho just received a new order for `"tulips"`. Use `append` to add this string to `orders`.

Hint
To use the `.append()` method on a list, attach it to the end of your list and pass an element you want to add in between the parenthesis `( )`

```
example_list = [1, 2, 3]
example_list.append(4)

print(example_list)
```

Will output:

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

3.
Another order has come in! Use `append` to add `"roses"` to `orders`.

Hint
Double-check your spelling and capitalization of `roses`.

You can add an element to `orders` like so:

```
orders.append( #YOUR ELEMENT HERE )
```

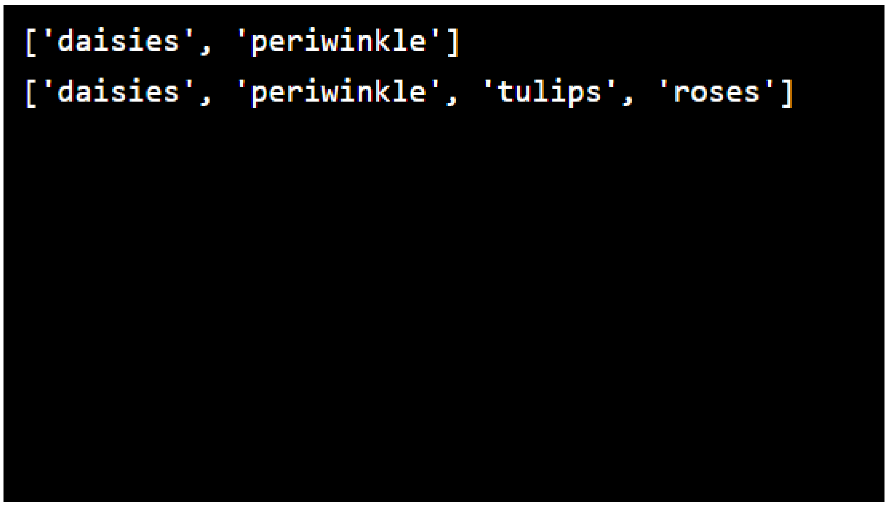
4.
Use `print` to inspect the `orders` Jiho has received today.

Hint
You should see:

```
['daisies', 'periwinkle', 'tulips', 'roses']
```

script.py

```
orders = ["daisies", "periwinkle"]
print(orders)

orders.append("tulips")
orders.append("roses")

print(orders)
```



```
['daisies', 'periwinkle']
['daisies', 'periwinkle', 'tulips', 'roses']
```

Growing a List: Plus (+)

When we want to add multiple items to a list, we can use `+` to combine two lists (this is also known as concatenation).

Below, we have a list of items sold at a bakery called `items_sold`:

```
items_sold = ["cake", "cookie", "bread"]
```

Suppose the bakery wants to start selling `"biscuit"` and `"tart"`:

```
items_sold_new = items_sold + ["biscuit", "tart"]
print(items_sold_new)
```

Would output:

```
['cake', 'cookie', 'bread', 'biscuit', 'tart']
```

In this example, we created a new variable, `items_sold_new`, which contained both the original items sold, and the new items. We can inspect the original `items_sold` and see that it did not change:

```
print(items_sold)
```

Would output:

```
['cake', 'cookie', 'bread']
```

We can only use `+` with other lists. If we type in this code:

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

we will get the following error:

```
TypeError: can only concatenate list (not "int") to list
```

If we want to add a single element using `+`, we have to put it into a list with brackets (`[]`):

```
my_list + [4]
```

Let’s use `+` to practice combining two lists!

Instructions

1.

Jiho is updating a list of `orders`. He just received orders for `"lilac"` and `"iris"`.

Create a list called `new_orders` that contains our new orders.

Hint

Remember the key components of a list:

1. A list begins and ends with square brackets (`[` and `]`).
2. Each item is separated by a comma (`,`)

```
practice_list = ["String", 6, 0.5, True]
```

Make sure to double-check your spelling, capitalization, and element order. Check to make sure it matches the values provided from the checkpoint.

2.

Use `+` to create a new list called `orders_combined` that combines `orders` with `new_orders`.

Hint

To combine two lists using `+`, define a new variable and set it to the two lists we want to combine with `+` in between.

```
list_one = [1, 2, 3]
list_two = [4, 5, 6]

combo_using_plus = list_one + list_two
print(combo_using_plus)
```

Will output:

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

3.

Remove the `#` and whitespace in front of the list `broken_prices`. If you run this code, you’ll get an error:

```
TypeError: can only concatenate list (not "int") to list
```

Fix the command by inserting brackets (`[` and `]`) so that it will run without errors.

Hint

Getting an `IndentationError`? Remember to remove the whitespace in front of `broken_prices`

script.py

```
orders = ["daisy", "buttercup", "snapdragon", "gardenia", "lily"]

# Create new orders here:
new_orders = ["lilac", "iris"]

orders_combined = orders + new_orders



broken_prices = [5, 3, 4, 5, 4] + [4]
```

Accessing List Elements

We are interviewing candidates for a job. We will call each candidate in order, represented by a Python list:

```
calls = ["Juan", "Zofia", "Amare", "Ezio", "Ananya"]
```

First, we'll call `"Juan"`, then `"Zofia"`, etc.

In Python, we call the location of an element in a list its *index*.

Python lists are *zero-indexed*. This means that the first element in a list has index `0`, rather than `1`.

Here are the index numbers for the list `calls`:

Element	Index
`"Juan"`	0
`"Zofia"`	1
`"Amare"`	2
`"Ezio"`	3
`"Ananya"`	4

In this example, the element with *index* `2` is `"Amare"`.

We can select a single element from a list by using square brackets (`[]`) and the index of the list item. If we wanted to select the third element from the list, we'd use `calls[2]`:

```
print(calls[2])
```

Will output:

```
Amare
```

Note: When accessing elements of a list, you *must* use an `int` as the index. If you use a `float`, you will get an error. This can be especially tricky when using division. For example `print(calls[4/2])` will result in an error, because `4/2` gets evaluated to the `float` `2.0`.

To solve this problem, you can force the result of your division to be an `int` by using the `int()` function. `int()` takes a number and cuts off the decimal point. For example, `int(5.9)` and `int(5.0)` will both become `5`.
Therefore, `calls[int(4/2)]` will result in the same value as `calls[2]`, whereas `calls[4/2]` will result in an error.

Instructions

1.
Use square brackets (`[` and `]`) to select the 4th employee from the list `employees`. Save it to the variable `employee_four`.

Hint
Remember lists are zero-indexed. The first element of a list will start at zero. If we are trying to access the *3rd* element, we are really trying to access the 2nd index.

```
calls = ["Juan", "Zofia", "Amare", "Ezio", "Ananya"]

print(calls[2]) #Will access the third element (second index)
```
Will output:

```
Amare
```
2.
Paste the following code into **script.py**:

```
print(employees[8])
```
What happens? Why?

Hint
When we try to access an element that is outside of the range of the list indexes, Python will return an `IndexError`.

```
IndexError: list index out of range
```
3.
Selecting an element that does not exist produces an `IndexError`.

In the line of code that you pasted, change `8` to an index that exists so that you don't get an `IndexError`.

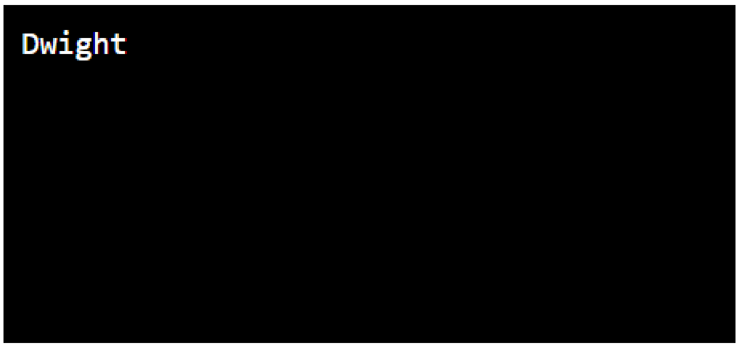
Run your code again!

Hint

Make sure you are selecting an element that exists in the list.

script.py

```
employees = ["Michael", "Dwight", "Jim", "Pam", "Ryan", "Andy", "Robert"]

employee_four = employees[3]

print(employees[1])
```



```
Dwight
```

Accessing List Elements: Negative Index

What if we want to select the last element of a list?

We can use the index `-1` to select the last item of a list, even when we don’t know how many elements are in a list.

Consider the following list with 6 elements:

```
pancake_recipe = ["eggs", "flour", "butter", "milk", "sugar", "love"]
```

If we select the `-1` index, we get the final element, `"love"`.

```
print(pancake_recipe[-1])
```

Would output:

```
love
```

This is equivalent to selecting the element with index `5`:

```
print(pancake_recipe[5])
```

Would output:

```
love
```

Here are the negative index numbers for our list:

Element	Index
`"eggs"`	`-6`
`"flour"`	`-5`
`"butter"`	`-4`
`"milk"`	`-3`
`"sugar"`	`-2`
`"love"`	`-1`

Instructions

1.

Create a variable called `last_element`.

Assign the last element in `shopping_list` to the variable `last_element` using a negative index.

Hint

The last element of a list can be accessed using the index `-1`

2.

Now select the element with index `5` and save it to the variable `index5_element`.

Hint

We can select a single element from a list by using square brackets (`[]`) and the index of the list item.

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

print(example_list[1])
```

Would output:

```
2
```

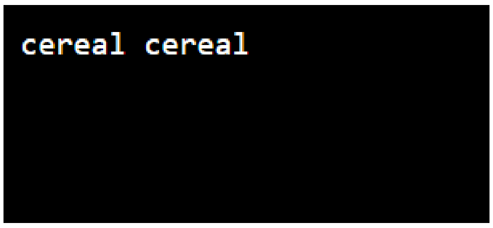
3.

Use `print` to display both `index5_element` and `last_element`.

Note that they are equal to `"cereal"`!

script.py

```
shopping_list = ["eggs", "butter", "milk", "cucumbers", "juice", "cereal"]


last_element = shopping_list[-1]
index5_element = shopping_list[5]


print(last_element, index5_element)
```



```
cereal cereal
```

Modifying List Elements

Let’s return to our garden.

```
garden = ["Tomatoes", "Green Beans", "Cauliflower", "Grapes"]
```

Unfortunately, we forgot to water our cauliflower and we don’t think it is going to recover.

Thankfully our friend Jiho from Petal Power came to the rescue. Jiho gifted us some strawberry seeds. We will replace the cauliflower with our new seeds.

We will need to modify the list to accommodate the change to our `garden` list. To change a value in a list, reassign the value using the specific index.

```
garden[2] = "Strawberries"

print(garden)
```

Will output:

```
["Tomatoes", "Green Beans", "Strawberries", "Grapes"]
```

Negative indices will work as well.

```
garden[-1] = "Raspberries"

print(garden)
```

Will output:

```
["Tomatoes", "Green Beans", "Strawberries", "Raspberries"]
```

Instructions

1.

We have decided to start selling some of our garden produce. Word around our town has spread and people are interested in getting some of our delicious vegetables and fruit.

We decided to create a waitlist to make sure we can sell to all of our new customers!

Define a list called `garden_waitlist` and set the value to contain our customers (in order): `"Jiho"`, `"Adam"`, `"Sonny"`, and `"Alisha"`.

Hint

Remember the key components of a list:

1. A list begins and ends with square brackets (`[` and `]`).
2. Each item is separated by a comma (`,`)

```
practice_list = ["String", 6, 0.5, True]
```

Make sure to double-check your spelling, capitalization, and element order. Check to make sure it matches the values provided from the checkpoint.

2.

`"Adam"` decided his fridge is too full at the moment and asked us to remove him from the waitlist and make space for one of our other townsfolk.

Replace `"Adam"` with our other interested customer `"Calla"` using the index method we used in the narrative.

Print `garden_waitlist` to see the change!

Hint

To change a value in a list, reassign the value using the specific index inside brackets `[ ]`.

```
example_list = [1, 2, 3, 4]
example_list[2] = -1

print(example_list)
```

Would output:

```
[1, 2, -1, 4]
```

3.

`Alisha` realized she was already stocked with all the items we are selling. She asked us to replace her with her friend `Alex` who just ran out.

Replace `Alisha` with `Alex` ***using a negative index***.

Print `garden_waitlist` again to see the change!

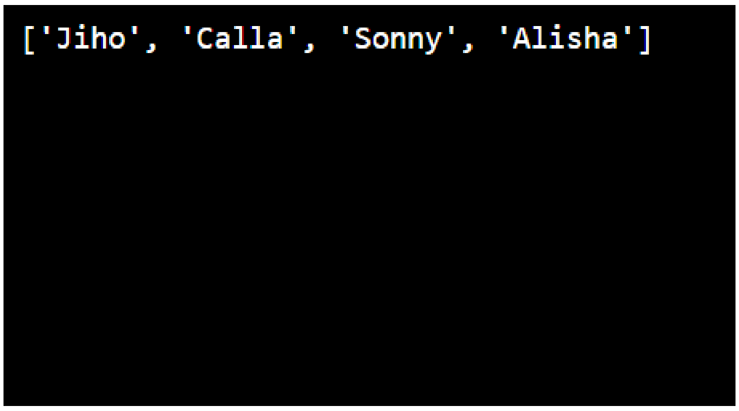
Hint

Your list output should look like this:

```
['Jiho', 'Calla', 'Sonny', 'Alex']
```

script.py

```
# Your code below:
garden_waitlist = ["Jiho", "Adam", "Sonny", "Alisha"]

garden_waitlist[1] = "Calla"

print(garden_waitlist)

garden_waitlist[-1] = "Alex"
```


```
['Jiho', 'Calla', 'Sonny', 'Alisha']
```

Shrinking a List: Remove

We can remove elements in a list using the `.remove()` Python method.

Suppose we have a filled list called `shopping_line` that represents a line at a grocery store:

```
shopping_line = ["Cole", "Kip", "Chris", "Sylvana"]
```

We could remove `"Chris"` by using the `.remove()` method:

```
shopping_line.remove("Chris")

print(shopping_line)
```

If we examine `shopping_line`, we can see that it now doesn’t contain `"Chris"`:

```
["Cole", "Kip", "Sylvana"]
```

We can also use `.remove()` on a list that has duplicate elements.

Only the first instance of the matching element is removed:

```
# Create a list
shopping_line = ["Cole", "Kip", "Chris", "Sylvana", "Chris"]

# Remove a element
shopping_line.remove("Chris")
print(shopping_line)
```

Will output:

```
["Cole", "Kip", "Sylvana", "Chris"]
```

Let's practice using the `.remove()` method to remove elements from a list.

Instructions

1.
We have decided to get into the grocery store business. Our manager Calla has decided to store all the inventory purchases in a list to help track what products need to be ordered.

Let's create a list called `order_list` with the following values (in this particular order):

`"Celery"`, `"Orange Juice"`, `"Orange"`, `"Flatbread"`

Print `order_list` to see the current list.

Hint
Remember the key components of a list:

1. A list begins and ends with square brackets (`[` and `]`).
2. Each item is separated by a comma (`,`)

```
practice_list = ['String', 6, 0.5, True]
```

Make sure to double-check your spelling and capitalization. Check to make sure it matches the values provided from the checkpoint in the provided order.

2.
We are in luck! We actually found a spare case of `"Flatbread"` in our back storage. We won't need to order it anymore. Let's remove it from `order_list` using the `.remove()` method.

Print `order_list` to see the current list.

Hint
To use the `.remove()` method, call it on the list you are modifying and pass the value you want to remove in between the parenthesis `( )`.

```
practice_list = ["a", "b", "c"]
practice_list.remove("b")

print(practice_list)
```

Would output:

```
["a", "c"]
```

3.
Our store has grown to be a huge success! We decided to open a second store and require a new order list. Calla has done us the favor of putting one together.

Create a new list called `new_store_order_list` and assign it the following values (in order):

`"Orange"`, `"Apple"`, `"Mango"`, `"Broccoli"`, `"Mango"`

Note: Our second store is going to need two orders of mangos so the value is duplicated.

Print `new_store_order_list` to see the current list.

Hint
Remember the key components of a list:

1. A list begins and ends with square brackets (`[` and `]`).
2. Each item is separated by a comma (`,`)

```
practice_list = ['String', 6, 0.5, True]
```

Make sure to double-check your spelling and capitalization. Check to make sure it matches the values provided from the checkpoint.

4.
We are in luck again! We actually found a spare case of `"Mango"` in our back storage.

We won't be needing to place two orders anymore.

Let's remove it from `new_store_order_list` using the `.remove()` method.

Print `new_store_order_list` to see the current list.

Hint
The `.remove()` method will work on duplicate values in a list. `.remove()` will delete the first instance of a match for the provided element you want to delete.

```
practice_list = ["a", "b", "c", "b"]
practice_list.remove("b")

print(practice_list)
```

Would output:

```
["a", "c", "b"]
```

5.

Calla ran to tell us some important news! She asked us to remove `"Onions"` from our new `new_store_order_list`. If we double-check our list, we will notice we don't have `"Onions"` on our list.

Let's see what happens when we try to remove an item that does not exist.

Call the `.remove()` method with the value of `"Onions"` on our `new_store_order_list` list.

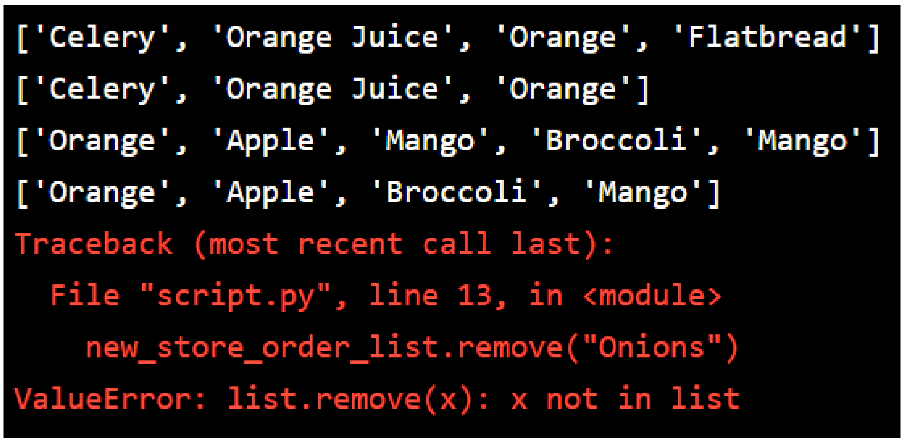
Hint

When we call `.remove()` on a list with a value that does not exist, we will receive a `ValueError`.

```
Traceback (most recent call last):
  File "script.py", line 18, in <module>
    new_store_order_list.remove("Onions")
ValueError: list.remove(x): x not in list
```

script.py

```python
# Your code below:
order_list = ["Celery", "Orange Juice", "Orange", "Flatbread"]
print(order_list)

order_list.remove("Flatbread")
print(order_list)

new_store_order_list = ["Orange", "Apple", "Mango", "Broccoli", "Mango"]
print(new_store_order_list)

new_store_order_list.remove("Mango")
print(new_store_order_list)
new_store_order_list.remove("Onions")
```



```
['Celery', 'Orange Juice', 'Orange', 'Flatbread']
['Celery', 'Orange Juice', 'Orange']
['Orange', 'Apple', 'Mango', 'Broccoli', 'Mango']
['Orange', 'Apple', 'Broccoli', 'Mango']
Traceback (most recent call last):
  File "script.py", line 13, in <module>
    new_store_order_list.remove("Onions")
ValueError: list.remove(x): x not in list
```

Two-Dimensional (2D) Lists

We've seen that the items in a list can be numbers or strings. Lists can contain other lists! We will commonly refer to these as *two-dimensional (2D)* lists.

Once more, let's look at a class height example:

- Noelle is 61 inches tall
- Ava is 70 inches tall
- Sam is 67 inches tall
- Mia is 64 inches tall

Previously, we saw that we could create a list representing both Noelle's name and height:

```
noelle = ["Noelle", 61]
```

We can put several of these lists into one list, such that each entry in the list represents a student and their height:

```
heights = [["Noelle", 61], ["Ava", 70], ["Sam", 67], ["Mia", 64]]
```

We will often find that a two-dimensional list is a very good structure for representing grids such as games like tic-tac-toe.

```
#A 2d list with three lists in each of the indices.
tic_tac_toe = [
            ["X","O","X"],
            ["O","X","O"],
            ["O","O","X"]
]
```

Let's practice creating our own 2D list!

Instructions

1.

A new student named `"Vik"` has joined our class. Vik is `68` inches tall. Add a sublist to the end of the `heights` list that represents Vik and his height.

Hint

Your sublist should be `["Vik", 68]`.

2.

Create a two-dimensional list called `ages` where each sublist contains a student's name and their age. Use the following data:

- `"Aaron"` is `15`
- `"Dhruti"` is `16`

Hint

Remember the key components of a two-dimensional list:

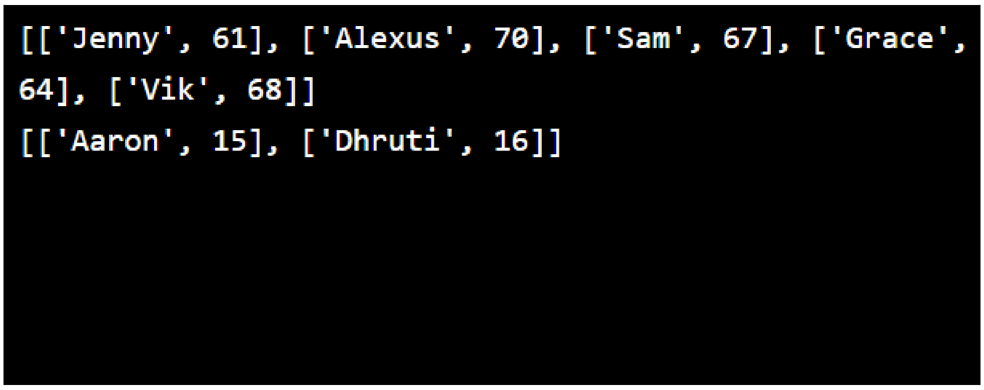
1. A two-dimensional list begins and ends with square brackets (`[` and `]`). This is our “container” list that wraps all of our inner sublists.
2. Any number of sublists within the “container” list. These are our inner lists.
3. Each item is separated by a comma (`,`) both in the inner and outer lists.

```
#Outermost "container" list
example_2d_list = [
  #Innermost sublists
  ["First Sublist", "Second Item"],
  ["Second Sublist", "Second Item"],
  ["Third Sublist", "Second Item"]
]
```

Make sure to double-check your spelling, capitalization, and element order. Check to make sure it matches the values provided from the checkpoint.

script.py

```
heights = [["Jenny", 61], ["Alexus", 70], ["Sam", 67], ["Grace", 64]]

heights.append(["Vik", 68])

ages = [["Aaron", 15], ["Dhruti", 16]]

print(heights)
print(ages)
```



```
[['Jenny', 61], ['Alexus', 70], ['Sam', 67], ['Grace',
64], ['Vik', 68]]
[['Aaron', 15], ['Dhruti', 16]]
```

Accessing 2D Lists

Let's return to our classroom heights example:

```
heights = [["Noelle", 61], ["Ali", 70], ["Sam", 67]]
```

Two-dimensional lists can be accessed similar to their one-dimensional counterpart. Instead of providing a single pair of brackets `[ ]` we will use an additional set for each dimension past the first.

If we wanted to access `"Noelle"`'s height:

```
#Access the sublist at index 0, and then access the 1st index of that sublist.
noelles_height = heights[0][1]
print(noelles_height)
```

Would output:

```
61
```

Here are the index numbers to access data for the list `heights`:

Element	Index
`"Noelle"`	`heights[0][0]`
`61`	`heights[0][1]`
`"Ali"`	`heights[1][0]`
`70`	`heights[1][1]`
`"Sam"`	`heights[2][0]`
`67`	`heights[2][1]`

Let's practice accessing data in a two-dimensional list.

Instructions

1.

We want to have a way to store all of our classroom test score data.

Using the provided table, create a two-dimensional list called `class_name_test` to represent the data. Each sublist in `class_name_test` should have one student's name and their associated score.

Name	Test Score
"Jenny"	90
"Alexus"	85.5
"Sam"	83
"Ellie"	101.5

Print `class_name_test` to see the result.

Hint

The first sublist in `class_name_test` should be `["Jenny", 90]`. See if you can fill in the rest!

2.

Use double square brackets (`[][]`) to select `Sam`'s test score from the list `class_name_test`.

Save it to the variable `sams_score`.

Print the variable `sams_score` to see the result.

Hint

When accessing a two-dimensional list, determine the index for both the inner and outer list.

Apply the indices inside a pair of double brackets `[][]`.

```
example_2d_list = [
  ["First Sublist"],
  ["Second Sublist"],
  ["Third Sublist"]
]
```

Element	Access Index
`"First Sublist"`	`example_2d_list[0][0]`
`"First Sublist"`	`example_2d_list[1][0]`
`"First Sublist""`	`example_2d_list[2][0]`

3.

Use double square brackets (`[][]`) to select `Ellie`s test score from the list `class_name_test`. ***This time only use negative indices!***

Save it to the variable `ellies_score`.

Print the variable `ellies_score` to see the result.

Hint

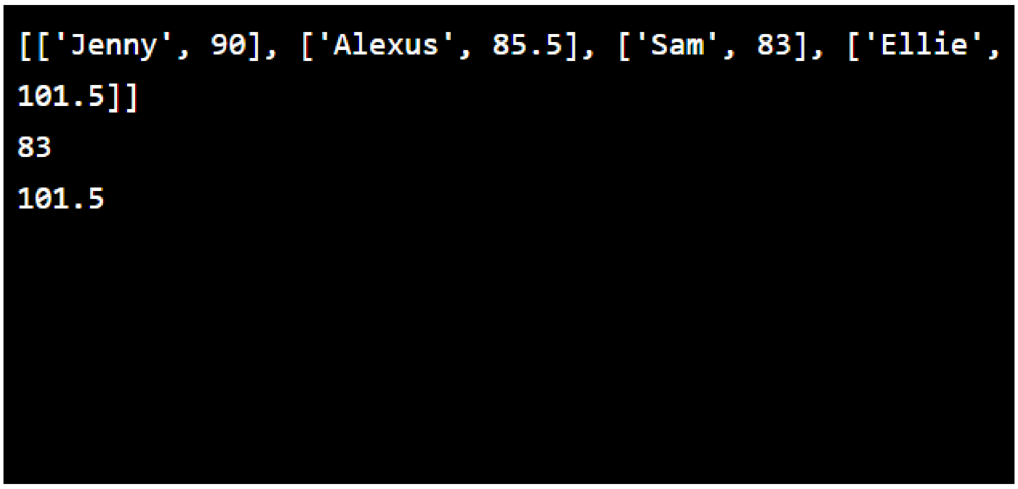
Negative indices in two-dimensional lists work the same as their one-dimensional counterpart.

```
example_2d_list = [
  ["First Sublist", "Second Item"],
  ["Second Sublist", "Second Item Two"],
  ["Third Sublist", "Second Item Three"]
]
```

Element	Access Index
`"Second Item"`	`example_2d_list[-3][-1]`
`"Second Item Two"`	`example_2d_list[-2][-1]`
`"Second Item Three"`	`example_2d_list[-1][-1]`

script.py

```python
#Your code below:
class_name_test = [["Jenny", 90], ["Alexus", 85.5], ["Sam", 83], ["Ellie", 101.5]
]
print(class_name_test)

sams_score = class_name_test[2][1]
print(sams_score)

ellies_score = class_name_test[-1][-1]
print(ellies_score)
```



```
[['Jenny', 90], ['Alexus', 85.5], ['Sam', 83], ['Ellie',
101.5]]
83
101.5
```

Modifying 2D Lists

Now that we know how to access two-dimensional lists, modifying the elements should come naturally.

Let’s return to a classroom example, but now instead of heights or test scores, our list stores the student’s favorite hobby!

```python
class_name_hobbies = [["Jenny", "Breakdancing"], ["Alexus",
"Photography"], ["Grace", "Soccer"]]
```

`"Jenny"` changed their mind and is now more interested in `"Meditation"`.

We will need to modify the list to accommodate the change to our `class_name_hobbies` list. To change a value in a two-dimensional list, reassign the value using the specific index.

```python
# The list of Jenny is at index 0. The hobby is at index 1.
class_name_hobbies[0][1] = "Meditation"
print(class_name_hobbies)
```

Would output:

```
[["Jenny", "Meditation"], ["Alexus", "Photography"], ["Grace",
"Soccer"]]
```

Negative indices will work as well.

```
# The list of Grace is the last entry. The hobby is the last element.
class_name_hobbies[-1][-1] = "Football"
print(class_name_hobbies)
```

Would output:

```
[["Jenny", "Meditation"], ["Alexus", "Photography"], ["Grace",
"Football"]]
```

Instructions

1.

Our school is expanding! We are welcoming a new set of students today from all over the world.

Using the provided table, create a two-dimensional list called `incoming_class` to represent the data. Each sublist in `incoming_class` should contain the name, nationality, and grade for a single student.

Name	Nationality	Grade Level
`"Kenny"`	`"American"`	9
`"Tanya"`	`"Ukrainian"`	9
`"Madison"`	`"Indian"`	7

Print `incoming_class` to see our list.

Hint

Remember the key components of a two-dimensional list:

1. A two-dimensional list begins and ends with square brackets (`[` and `]`). This is our “container” list that wraps all of our inner sublists.
2. Any number of sublists within the “container” list. These are our inner lists.
3. Each item is separated by a comma (`,`) both in the inner and outer lists.

For our example, the first sublist would look like this:

```
[["Kenny", "American", 9]]
```

Make sure to double-check your spelling, capitalization, and element order. Check to make sure it matches the values provided from the checkpoint.

2.

`"Madison"` passed an exam to advance a grade. She will be pushed into 8th grade rather than her current 7th in our list.

Modify the list using double brackets `[][]` to make the change. ***Use positive indices***.

Print `incoming_class` to see our change.

Hint

Given the list

```
my_list = [["a","b"],["c","d"]],
```

we could change the letter `"b"` to `"z"` using

```
my_list[0][1] = "z"
print(my_list)
```

Would output:

```
[["a","z"],["c","d"]]
```

3.

`"Kenny"` likes to be called by his nickname `"Ken"`. Modify the list using double brackets `[][]` to accommodate the change but ***only using negative indices***.

Print `incoming_class` to see our change.

Hint

Negative indices in two-dimensional lists work the same as their one-dimensional counterpart.

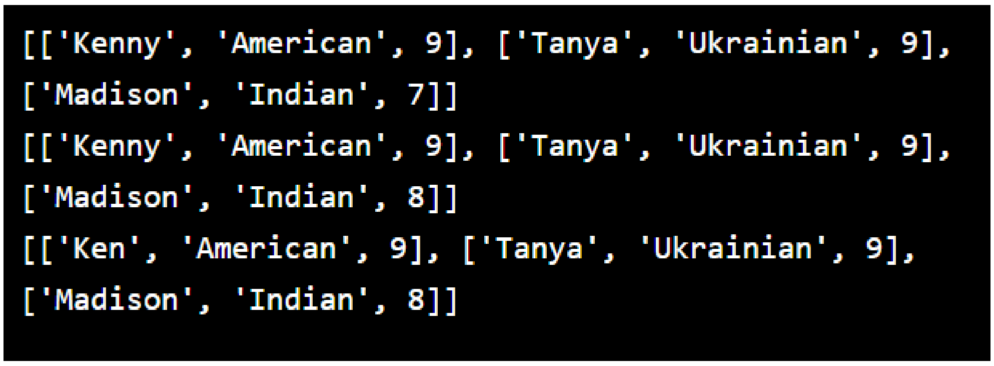
```
example_2d_list = [
  ["First Sublist", "Second Item"],
  ["Second Sublist", "Second Item Two"],
  ["Third Sublist", "Second Item Three"]
]
```

Element	Access Index
"Second Item"	example_2d_list[-3][-1]
"Second Item Two"	example_2d_list[-2][-1]
"Second Item Three"	example_2d_list[-1][-1]

script.py

```
#Your code below:
incoming_class = [["Kenny", "American", 9], ["Tanya", "Ukrainian", 9], ["Madison"
, "Indian", 7]]
print(incoming_class)

incoming_class[2][2] = 8
print(incoming_class)
```

```
incoming_class[-3][-3] = "Ken"
print(incoming_class)
```



```
[['Kenny', 'American', 9], ['Tanya', 'Ukrainian', 9],
['Madison', 'Indian', 7]]
[['Kenny', 'American', 9], ['Tanya', 'Ukrainian', 9],
['Madison', 'Indian', 8]]
[['Ken', 'American', 9], ['Tanya', 'Ukrainian', 9],
['Madison', 'Indian', 8]]
```

Review
So far, we have learned:

- How to create a list
- How to access, add, remove, and modify list elements
- How to create a two-dimensional list
- How to access and modify two-dimensional list elements

Let's practice these skills.

Instructions

1.
Maria is entering customer data for her web store business. We're going to help her organize her data.

Start by turning this list of customer first names into a list called `first_names`. Make sure to enter the names in this order:

- Ainsley
- Ben
- Chani
- Depak

Hint
Remember the key components of a list:

1. A list begins and ends with square brackets (`[` and `]`).

2. Each item is separated by a comma (,)

```
practice_list = ["String", 6, 0.5, True]
```

Make sure to double-check your spelling, capitalization, and element order. Check to make sure it matches the values provided from the checkpoint.

2.

Maria wants to track all customer's preferred sizes for her clothing. Create a list called `preferred_size`.

Fill our new list `preferred_size` with the following data, containing the preferred sizes for Ainsley, Ben, and Chani:

```
["Small", "Large", "Medium"]
```

Hint

Remember the key components of a list:

1. A list begins and ends with square brackets (`[` and `]`).
2. Each item is separated by a comma (`,`)

```
practice_list = ['String', 6, 0.5, True]
```

Make sure to double-check your spelling and capitalization. Check to make sure it matches the values provided from the checkpoint in the provided order.

3.

Oh no! We forgot to add Depak's size.

Depak's size is `"Medium"`. Use `.append()` to add `"Medium"` to the `preferred_size` list.

Print `preferred_size` to see our change.

Hint

To use the `.append()` method on a list, attach it to the end of your list and pass an element you want to add in between the parenthesis `( )`

```
example_list = [1, 2, 3]
example_list.append(4)

print(example_list)
```

Will output:

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

4.

Maria is having a hard time visualizing which customer is associated with each size. Let's restructure our two lists into a two-dimensional list to help Maria.

In addition to our already available data, Maria is adding a third value for each customer that reflects if they want expedited shipping on their orders.

This will be reflected using a boolean value (`True` for expedited, `False` for regular)

Create a two-dimensional list called `customer_data` using the following table as a reference for the data. Each sublist should contain a name, size, and expedited shipping option for a single person.

Name	Size	Expedited Shipping
`"Ainsley"`	`"Small"`	True
`"Ben"`	`"Large"`	False
`"Chani"`	`"Medium"`	True
`"Depak"`	`"Medium"`	False

Print `customer_data` to see the data.

Hint

Here is what our list would look like if it only included `"Ainsley"`

```
[["Ainsley","Small", True]]
```

Make sure to double-check your spelling, capitalization, and element order. Check to make sure it matches the values provided from the checkpoint.

5.

`"Chani"` reached out to Maria. She requested to switch to regular shipping to save some money.

Change the data value for `"Chani"`'s shipping preference to `False` in our two-dimensional list to reflect the change.

Hint

`"Chani's"` information should be listed as the third element in `customer_data` (index `2`).

Shipping information is the third item in each sublist (index `2`). Because Chani does not want expedited shipping, the value at this location should be changed to `False`."

6.

`"Ben"` reached out to Maria asking to remove his shipping option because he is not sure what type he wants.

Use the `.remove()` method to delete the shipping value from the sublist that contains ben's data.

Note: We never explicitly went over how to use the `.remove()` method on a 2d list together. If you feel like you are struggling, take a look at the hint for some guidance.

Hint

To use the `.remove()` method on a two-dimensional list, call it on the sublist you are modifying and pass the value you want to remove in between the parenthesis `( )`.

```
practice_list = [["a"], ["b"], ["c"]]
practice_list[1].remove("b")

print(practice_list)
```

Would output:

```
[["a"], [], ["c"]]
```

7.

Great job making it this far! One last thing, Maria received new customers, `"Amit"` and `"Karim"`, the following 2d list contains their data:

```
[["Amit", "Large", True], ["Karim", "X-Large", False]]
```

Create a new variable `customer_data_final`. Combine our existing list `customer_data` with our new customer 2d list using `+` by adding it to the end of `customer_data`.

Print `customer_data_final` to see our final result.

Hint

To combine two lists using `+`, define a new variable and set it to the two lists we want to combine with `+` in between.

Here is an example:

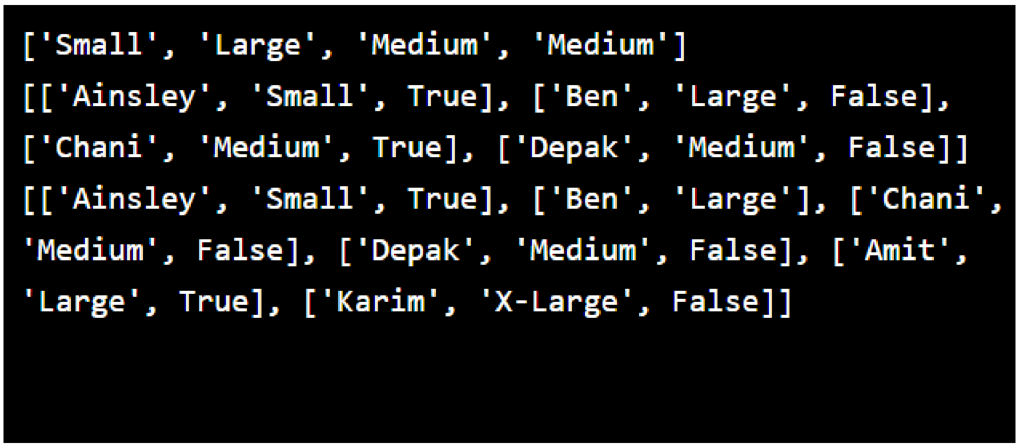
```
group_1 = [1, 2, 3]

group_2 = group_1 + [4, 5]
print(group_2)
```

Would output:

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

script.py

```
# Your code below:
first_names = ["Ainsley", "Ben", "Chani", "Depak"]

preferred_size = ["Small", "Large", "Medium"]

preferred_size.append("Medium")
print(preferred_size)

customer_data = [["Ainsley", "Small", True], ["Ben", "Large", False], ["Chani", "
Medium", True], ["Depak", "Medium", False]]
print(customer_data)

customer_data[2][2] = False

customer_data[1].remove(False)

customer_data_final = customer_data + [["Amit", "Large", True], ["Karim", "X-
Large", False]]
print(customer_data_final)
```



```
['Small', 'Large', 'Medium', 'Medium']
[['Ainsley', 'Small', True], ['Ben', 'Large', False],
['Chani', 'Medium', True], ['Depak', 'Medium', False]]
[['Ainsley', 'Small', True], ['Ben', 'Large'], ['Chani',
'Medium', False], ['Depak', 'Medium', False], ['Amit',
'Large', True], ['Karim', 'X-Large', False]]
```