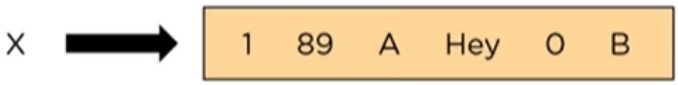
Set 7: Introduction to lists

Skill 7.1: Describe how data is stored in lists

Skill 7.1 Concepts

A list is a collection of data. In our day-to-day lives, we often encounter the need to make lists. For example, if we are packing for a trip we can make a packing list; if we are going to the grocery store, we may bring a grocery list. In Python, lists can hold values of multiple data types and simplify the data storage process by grouping similar data together.

The example below illustrates the different types of values that can be stored in a list called *x*.



One way we can visualize a list is to imagine a street of houses. Notice on our street below, each house has an address and that the addresses start at "0". Each address in our example is also referred to as an *index*. The contents inside each house are referred to as the *value* associated with the index.

index refers to the location of an element in a list.
value refers to the contents stored at the index.

Skill 7.1 Exercise 1

Skill 7.2: Create a list in Python

Skill 7.2 Concepts

Lists have many features which make them different from variables, but **most of what you've learned about variables also applies to lists**. For example, just like a variable:

- Lists should be given a **descriptive and meaningful** name.
- Lists can be initialized/set using the equals sign, =

The code below illustrates how to create a list of numbers in Python

```
My_first_list = [100, 250, 500]
```

The list above contains 3 values: 100, 250, 500. Notice that the values are separated with commas (,) and that the entire list is enclosed in brackets, [ ]. We can use *print* to display the contents of a list just like we would a variable.

Code	Output
`myFirstList = [100, 250, 500]` `print(myFirstList)`	`[100, 250, 500]`

A list of non-numeric values can be created by enclosing each value in the list in quotes,

Code	Output
`myPets = ["Wiggles", "Sir Toby", "Nemo"]` `print(myPets)`	`['Wiggles', 'Sir Toby', 'Nemo']`

Lists can hold a mixture of data types as well,

Code	Output
`items = ["eggs", 4.50, "bread", 3.25]` `print(items)`	`['eggs', 4.5, 'bread', 3.25]`

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

Code	Output
`items = []` `print(items)`	`[]`

Why would we create an empty list?

Usually, it's because we're planning on filling it up later based on some other input.

Skill 7.2 Exercises 1 thru 3

Skill 7.3: Grow a list with the *append()* method

Skill 7.3 Concepts

As we start exploring lists further, 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 parentheses 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.

Code	Output
append_example = ['This', 'is', 'an', 'example'] append_example.append('list') print(append_example)	['This', 'is', 'an', 'example', 'list']

Skill 7.3 Exercise 1

Skill 7.4: Grow a list with (+) or (*)

Skill 7.4 Concepts

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

Below, we've added new bakery items to the items_sold list using the + operator.

Code
items_sold = ["cake", "cookie", "bread"] items_sold_new = items_sold + ["biscuit", "tart"] print(items_sold_new)
Output
['cake', 'cookie', 'bread', 'biscuit', 'tart']

In the example above, 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,

Code	Output
print(items_sold)	['cake', 'cookie', 'bread']

We can only use the + operator with other lists. The code below, for example, will produce an error,

Code	Output
my_list = [1, 2, 3] my_new_list = my_list + 4	TypeError: can only concatenate list (not "int") to list

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

Code	Output
my_list = [1, 2, 3] my_new_list = my_list + [4] print(my_new_list)	[1, 2, 3, 4]

Another useful way to grow a list is with the (*) sign. For example what if we wanted a list that contained 10 x's or even 200 x's?

Code
`x_list = ["x"] * 10` `print(x_list)`
Output
['x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x']

Skill 7.4 Exercise 1

Skill 7.5: Access list elements

Skill 7.5 Concepts

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 below,

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

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. Below we've accessed each element in the list *calls.*

Code	Output

`calls = ["Juan", "Zofia", "Amare", "Ezio", "Ananya"]`
`print(calls[0])`
`print(calls[1])`
`print(calls[2])`
`print(calls[3])`
`print(calls[4])`
	Juan
	Zofia
	Amare
	Ezio
	Ananya

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.

What if we want to select the last element of a list? We can use the negative index to select items starting at the last index,

Code	Output

`calls = ["Juan", "Zofia", "Amare", "Ezio", "Ananya"]`
`print(calls[-1])`
`print(calls[-2])`
`print(calls[-3])`
`print(calls[-4])`
`print(calls[-5])`
	Ananya
	Ezio
	Amare
	Zofia
	Juan

If you try to access an index that is not in the list, you will get an *IndexError*.

Code	Output

`myPets = ["Wiggles", "Sir Toby", "Nemo"]`
`print(myPets[0])`
`print(myPets[1])`
`print(myPets[2])`
`print(myPets[3])`
	`Wiggles`
	`Sir Toby`
	`Nemo`
	`    print(myPets[3])`
	`IndexError: list index out of range`

`myPets = ["Wiggles", "Sir Toby", "Nemo"]`
`print(myPets[-1])`
`print(myPets[-2])`
`print(myPets[-3])`
`print(myPets[-4])`
	`Nemo`
	`Sir Toby`
	`Wiggles`
	`    print(myPets[-4])`
	`IndexError: list index out of range`

Skill 7.5 Exercises 1 and 2

Skill 7.6: Modify list elements

Skill 7.6 Concepts

Each location in a list can be treated like its own variable. We've already seen how we can use bracket notation to reference values stored at specific locations in a list. Just like with other variables, we can assign or reassign the value of a specific location in a list using the assignment (=) operator.

Code

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

garden[2] = "Strawberries"
print(garden)
```

Output

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

Negative indices will work as well.

Code

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

garden[-1] = "Raspberries"

print(garden)
```

Output

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

Skill 7.6 Exercise 1

Skill 7.7: Remove list elements with the *remove()* method

Skill 7.7 Concepts

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

Suppose we have a filled list called *shopping_line* that represents a line at a grocery store. We could remove "Chris" by using the *remove()* method,

Code

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

shopping_line.remove("Chris")


print(shopping_line)
```

Output

```
['Cole', 'Kip', 'Sylvana']
```

We can also use the *.remove()* method on a list that has duplicate elements. In the example below, notice that only the first instance of the matching element is removed,

Code

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

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

Output

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

Removing an item that does not exist in a list results in an error,

Code

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

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

Output

```
ValueError: list.remove(x): x not in list
```

Skill 7.7 Exercise 1

Skill 7.8: Check if an element is in a list

Skill 7.8 Concepts

The easiest way to check if an element is a list is to use the *in* operator.

The *in* operator checks for membership and returns True if the element is found in the list, and False otherwise.

Code

```
inventory = ["knife", "cheese"]
if "coin" in inventory:
    inventory.remove("coin")
    print("The ferryman takes the coin and invites you to board.")
else:
    print("The ferryman refuses your passage")
```

Output

```
The ferry man refuses your passage
```

You can also use the *not in* operator to check if an element is not present in a list.

Code

```
inventory = ["knife", "cheese"]
if "coin" not in inventory:
    print("The ferryman refuses your passage")
else:
    inventory.remove("coin")
    print("The ferryman takes the coin and invites you to board.")
```

Output

```
The ferry man refuses your passage
```

In the previous skill we learned about the remove method and saw that if we tried to remove something that didn't exist, we got an error. Using in or not in to check whether an element is in a list is a good way to avoid this error.

Skill 7.8 Exercise 1

Skill 7.9: Find the length of a list

Skill 7.9 Concepts

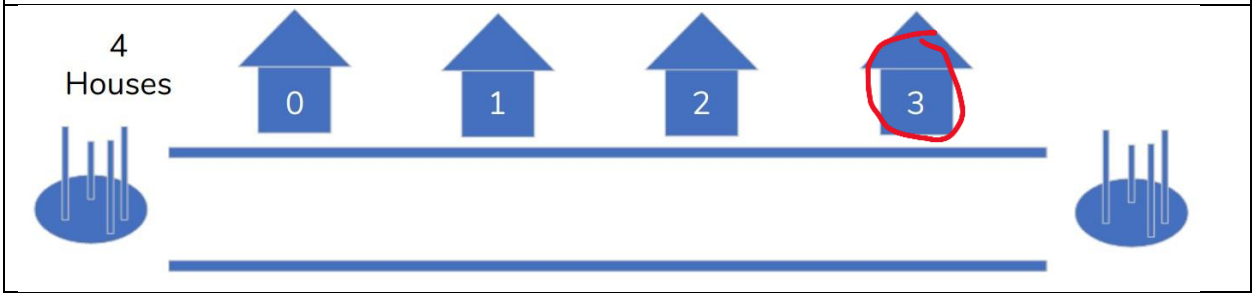
To find the length of a list in Python, the built-in len() function is used. This function returns the number of elements within the list.

Code

`my_list = [10, 20, 30, 40, 50]`
`list_length = len(my_list)`
`print(f"The length of the list is: {list_length}")`

Output
The length of the list is 5

In the illustration below, we see that the address of the last house in our array is one less than the length.

The list below contains 4 houses. Notices the address of the last house is 1 less than the number of total houses.


Consider the following code snippet,

Code	Output

`my_list = [10, 20, 30, 40, 50]`
`length = len(my_list)`
`last_element = my_list[length]`
	`IndexError: list index out of range`

`my_list = [10, 20, 30, 40, 50]`
`length = len(my_list)`
`last_element = my_list[length - 1]`
`print(last_element)`
	5

Skill 7.9 Exercise 1