**Lists**

A list is simply the way in which we store information in Python.

* Lists begin and end with square brackets [ ].
* Each item [must, be, separated, with, a, comma].
* It is good practice to insert a space, after, every, comma.

heights = [61, 70, 67, 64]

And here is a list.

Lists can contain numbers and strings.

ints\_and\_strings = [1, 2, 3, "four", "five"]

This string contains integers and strings.

Strings can even contain Booleans:

mixed\_list\_common = ["Mia", 27, False, 0.5]

**Empty Lists**

Lists do not have to contain anything, some maybe empty.

This maybe because we plan to fill it later with another input.

empty\_list = []

**List Methods**

In Python, for a specific data-type there is built-in functionality that we can use to create, manipulate and 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.

One example is the .append() method 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)

When we use the .append() method, a new element is always added to the end of the list.

garden = ["Tomatoes", "Grapes", "Cauliflower"]

# Append a new element

garden.append("Green Beans")

print(garden)

In the above example, Green Beans is added to the end of our list.

**Growing a List: Plus (+)**

We can add multiple items to a list. In this instance we can use the + sign, also known as concatenation.

items\_sold\_new = items\_sold + ["biscuit", "tart"]

print(items\_sold\_new)

Remember that if we want to add a single element using +, we have to put it into a list with brackets.

my\_list + [4]

**Accessing List Elements**

In Python, we call the location of an element in a list its index. Python lists are zero-index. This means that the first element in a list has index 0.

This list:

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

A picture containing text, screenshot, font, number

Description automatically generatedHas the following index positions:

It is possible to 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])

This would output the name Amare who is at index position 2.

**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.

**Accessing List Elements: Negative Index**

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

pancake\_recipe = ["eggs", "flour", "butter", "milk", "sugar", "love"]

print(pancake\_recipe[-1])

Look at this list. In this instance our print command would print love to the terminal.

Here are the following index numbers for our list:A picture containing text, screenshot, font, number

Description automatically generated

**Modifying List Elements**

Sometimes we may want to replace elements within a list.

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

garden[2] = "Strawberries"

Notice here that we would be replacing Cauliflower which as index position 2 (list starts from base 0) with Strawberries.

You can also use negative indices!

garden[-1] = "Raspberries"

print(garden)

This would replace the final element in the list.

**Shrinking a list: Remove**

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

Perhaps we have a shopping list:

shopping\_line = ["Cole", "Kip", "Chris", "Sylvana"]

We can remove Chris with the .remove() method.

shopping\_line.remove("Chris")

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)

Duplicate Elements

IF we 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)

This would output: [‘Cole’, ‘Kip’, ‘Sylvana’, ‘Chris’]

**Two-Dimensional (2D) Lists**

Lists can contain other lists!

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

Sometimes we can find that a two-dimensional list is a 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"]

]

**Accessing 2D Lists**

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.

In this example we want to access ‘Noelle’s’ height.

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

#Access the sublist at index 0, and then access the 1st index of that sublist.

noelles\_height = heights[0][1]

print(noelles\_height)

A picture containing text, screenshot, font, number

Description automatically generatedThis is what the index numbers next to sublist numbers would look like.

Here are some more examples;

#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)

**Modifying 2d Lists**

Now we know how to access two-dimensional lists, modifying the elements should not be so difficult.

To change a value in a two-dimensional list, reassign the value using the specific index.

# The list of Jenny is at index 0. The hobby is at index 1.

class\_name\_hobbies[0][1] = "Meditation"

print(class\_name\_hobbies)

Negative indices also work.

**Working With Lists**

**Python List Methods**

.count() – A list method to count the number of occurrences of an element in a list.

.insert() – A list method to insert an element into a specific index of a list.

.pop() – A list method to remove an element from a specific index or from the end of a list.

.range() – A built-in Python function to create a sequence of integers.

len() – A built-in Python function to get the length of a list.

.sort() / sorted() – A method and a built in function to sort a list.

**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 two inputs:

* The index you want to insert into.
* 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.

store\_line = ["Karla", "Maxium", "Martim", "Isabella"]

store\_line.insert(2, "Vikor")

print(store\_line)

Here is the method in action.

Remember:

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 be shift.

**Removing by Index: Pop**

The .pop() method allows us to remove elements at a specific index position. It takes an optional single input.

cs\_topics = ["Python", "Data Structures", "Balloon Making", "Algorithms", "Clowns 101"]

removed\_element = cs\_topics.pop()

print(cs\_topics)

print(removed\_element)

1. .pop() can be used without an index. It will remove the LAST element of the list. In this case it would be Clowns 101.
2. .pop() will return the value that was removed. If you wanted to know which element was removed, you can assign a variable to the call of the .pop() method.

Range

**Consecutive Lists: Range**

Python has a way of creating lists. For example; instead of us typing out all the numbers between 1 and a 100 we can use the range() function.

The function range() takes a single input, and generates numbers starting at 0 and ending at the number before the input. If we wanted to use the numbers from 0 to 9 we use range(10) because 10 is 1 greater than 9.

my\_range = range(10)

print(my\_range)

OUTPUT:

range(0, 10)

Remember: the range() function produces a range object. In this case; range(0, 10).

If we want to use the range object then we have to convert it to a list using another built in function; list().

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

In this instance if we wanted the numbers between 0 and 9 we would write;

print(list(my\_range))

Here are some more examples;

number\_list = range(9)

print(list(number\_list))

zero\_to\_seven = range(8)

print(list(zero\_to\_seven))

The first example would print [0, 1, 2, 3, 4, 5, 6, 7, 8].

The second example would print [0, 1, 2, 3, 4, 5, 6, 7].

**Range between numbers**

So far we have looked at a range between 0 and x. But what if we wanted the range of numbers between 2 and 9?

my\_list = range(2, 9)

print(list(my\_list))

This would print; [2, 3, 4, 5, 6, 7, 8].

**Skip numbers**

If we used a third input, we can create a rule that skips certain numbers;

my\_range2 = range(2, 9, 2)

print(list(my\_range2))

Here we are looking for numbers between 2 and 9. This would return a list where each number is 2 greater than the previous number. This would print;

[2, 4, 6, 8].

You can skip in as many numbers as you want. You could start at 1 and skip in increments of 10.

my\_range3 = range(1, 100, 10)

print(list(my\_range3))

Which would output [1, 11, 21, 31, 41….91]

The list stops at 91. The next increment would have been 101, which exceeds our range.

**Length**

Sometimes we need to know the number of items in a list. To do this we use len().

my\_list = [1, 2, 3, 4, 5]

print(len(my\_list))

This would output 5 as there are 5 elements in this list.

**Slicing Lists 1.**

In Python sometimes we want to extract only a portion of a list. Dividing a list is known as slicing.

We have some letters;

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

We want to select from ‘b’ through to ‘f’.

We can do this using the following syntax: letters[start:end].

* start: is the index of the first element that we want to include in our selection. In this instance we want to start at b and it has an index position of 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.

Here is an example:

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

beginning = suitcase[0:2]

print(beginning)

# Your code below:

middle = suitcase[2:4]

print(middle)

Beginning will print [‘shirt’, ‘shirt’].

Middle will print [‘pant’, ‘pants’].

**Slicing Lists 2**

The first n elements

Slicing syntax in Python is flexible. Previously we sliced elements from between certain points. But what if we wanted the first n elements or the last n elements?

This is the syntax:

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

fruits[:n]

By replacing n with the number 3 we would receive;

[‘apple’, ‘cherry’, ‘pineapple’].

We can inverse it by using -n. If we used fruits[-2:] we would get;

[‘orange’, ‘mango’]. This is because mango is at position -1 and orange is at position -2. It starts at the largest negative position and works towards the lowest.

All but n last elements

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

fruits[:-n]

Notice the difference? The -n is on the opposite side of the colon.

print(fruits[:-1])

From our fruits example, this starts counting from 0 up to position -1.

In this instance we would get: apple, cherry, pineapple, orange but NOT mango.

Here is a good way to remember the concept:

list[first n elements : all but n elements]

Here is the code in action:

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)

**Counting in a List**

Simple List

In Python we might want to count occurrences of an item in a list. Here is a list:

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

If we wanted to count the number of ‘i’s in the list we would use the .count() method.

num\_i = letters.count("i")

print(num\_i)

This would output 4.

2d List

Here is a 2d list:

number\_collection = [[100, 200], [100, 200], [475, 29], [34, 34]]

Lets say we want to count the instances that [100, 200] appear.

To do this we would simply use:

num\_pairs = number\_collection.count([100, 200])

print(num\_pairs)

This would output 2. Notice that we have saved this to a variable?

**Sorting Lists 1**

Sometimes we may need to sort in alphabetical or numerical order.

For this we use the .sort() method.

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

names.sort()

print(names)

Let empty this would sort in alphabetical order.

If we wanted to reverse the alphabetical order we could use:

names.sort(*reverse*=True)

print(names)

This would print a list in reverse alphabetical order; [‘Xander’, ‘Willow’…’Angel’].

NOTE: The .sort() method does not return a value. It modifies the list directly. This means we can not assign it to a variable.

**Sorting Lists 2**

There is a built in function; sorted().

The sorted() function is different from the .sort() method.

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

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

sorted\_names = sorted(names)

print(sorted\_names)

Again, this would sort into alphabetical order; [‘Angel’, ‘Buffy’, ‘Giles’, ‘Willow’, ‘Xander’].

If we were to print(names), the original list would be unaffected. This is because it does not change the original list. It is returning a new one.

**Review**

* Add elements to a list by index using the .insert() method.
* Remove elements from a list by index using the .pop().
* 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.