# Python Lists

About 35 minutes

Welcome to your next course — Lists in Python!

This first introductory lesson will teach you how to create lists, a common data structure used to store sequential information. By the end of the lesson, you will know how to make different kinds of lists add elements to them in different ways. You will see many examples of lists that are similar to what you will encounter as you enter the world of coding.

Creating and Modifying a List in Python

# What is a list?

A list is an ordered set of objects in Python.

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

* Jenny is 61 inches tall
* Alexus is 70 inches tall
* Sam is 67 inches tall
* Grace is 64 inches tall

In Python, we can create a variable called heights to store these numbers:

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.

**1.**

A new student just joined the class:

* Cole is 65 inches tall

Add Cole's height to the end of the list heights.

**2.**

Remove the # in front of the definition of the list broken\_heights. If you run this code, you'll get an error:

SyntaxError: invalid syntax

Add commas (,) to broken\_heights so that it runs without errors.

Solution:

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

broken\_heights = [65, 71, 59, 62]

Creating and Modifying a List in Python

# Lists II

Lists can contain more than just numbers.

Let's revisit our height example:

* Jenny is 61 inches tall
* Alexus is 70 inches tall
* Sam is 67 inches tall
* Grace is 64 inches tall

We can make a list of strings that contain the students' names:

names = ['Jenny', 'Alexus', 'Sam', 'Grace']

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

mixed\_list = ['Jenny', 61]

Instructions

**1.**

Add any string to the list ints\_and\_strings.

**2.**

Create a new list called sam\_height that contains:

* The string 'Sam'
* The number 67

**Solution:**

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

sam\_height = ['Sam', 67]

Creating and Modifying a List in Python

# List of Lists

We've seen that the items in a list can be numbers or strings. They can also be other lists!

Once more, let's return to our class height example:

* Jenny is 61 inches tall
* Alexus is 70 inches tall
* Sam is 67 inches tall
* Grace is 64 inches tall

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

jenny = ['Jenny', 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 = [['Jenny', 61], ['Alexus', 70], ['Sam', 67], ['Grace', 64]]

Instructions

**1.**

A new student named 'Vik' has joined our class. Vik is 68 inches tall. Add a sublist to heights that represents Vik and his height.

**2.**

Create a list of lists called ages where each sublist contains a student's name and their age. Use the following data:

* 'Aaron' is 15
* 'Dhruti' is 16

Solution:

heights = [['Jenny', 61], ['Alexus', 70], ['Sam', 67], ['Grace', 64], ['Vik', 68]]

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

Creating and Modifying a List in Python

# Zip

Again, let's return to our class height example:

* Jenny is 61 inches tall
* Alexus is 70 inches tall
* Sam is 67 inches tall
* Grace is 64 inches tall

Suppose that we already had a list of names and a list of heights:

names = ['Jenny', 'Alexus', 'Sam', 'Grace']

heights = [61, 70, 67, 65]

If we wanted to create a list of lists that paired each name with a height, we could use the command zip. zip takes two (or more) lists as inputs and returns an object that contains a list of pairs. Each pair contains one element from each of the inputs. You won't be able to see much about this object from just printing it:

names\_and\_heights = zip(names, heights)

print(names\_and\_heights)

because it will return the location of this object in memory. Output would look something like this:

<zip object at 0x7f1631e86b48>

To see the nested lists, you can convert the zip object to a list first:

print(list(names\_and\_heights))

returns:

[('Jenny', 61), ('Alexus', 70), ('Sam', 67), ('Grace', 65)]

Instructions

**1.**

Use zip to create a new variable called names\_and\_dogs\_names that combines names and dogs\_names into a list of lists.

**2.**

Print the zip object as a list by calling list() on names\_and\_dogs\_names first.

Solution:

names = ['Jenny', 'Alexus', 'Sam', 'Grace']

dogs\_names = ['Elphonse', 'Dr. Doggy DDS', 'Carter', 'Ralph']

names\_and\_dogs\_names = zip (names, dogs\_names)

print(list(names\_and\_dogs\_names))

Creating and Modifying a List in Python

# 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 later based on some other input. We'll talk about two ways of filling up a list in the next exercise.

Instructions

**1.**

Create an empty list and call it my\_empty\_list.

Solution

my\_empty\_list = []

# Growing a List: Append

We can add a single element to a list using .append(). For example, suppose we have an empty list called empty\_list:

empty\_list = []

We can add the element 1 using the following commands:

empty\_list.append(1)

If we examine empty\_list, we see that it now contains 1:

>>> print(empty\_list)

[1]

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

my\_list = [1, 2, 3]

# Append a number

my\_list.append(5)

print(my\_list) # check the result

the output looks like:

[1, 2, 3, 5]

It's important to remember that .append() comes **after** the list. This is different from functions like print, which come **before**.

Instructions

**1.**

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

Use print to inspect the orders he has received today.

**2.**

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

**3.**

Another order has come in! Use append to add 'roses' to orders.

**4.**

Use print to inspect the orders Jiho has received today.

Solution:

orders = ['daisies', 'periwinkle']

print(orders)

orders.append('tulips')

orders.append('roses')

print(orders)

Creating and Modifying a List in Python

# Growing a List: Plus (+)

When we want to add multiple items to a list, we can use + to combine two lists.

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)

['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 ones. We can inspect the original items\_sold and see that it did not change:

>>> print(items\_sold)

['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]

Instructions

**1.**

Jiho is still updating his list of orders. He just received orders for 'lilac' and 'iris'.

Use + to create a new list called new\_orders that combines orders with the two new orders.

**2.**

Remove the # 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.

Solution:

orders = ['daisy', 'buttercup', 'snapdragon', 'gardenia', 'lily']

# Create new orders here:

new\_orders = orders + ['lilac', 'iris']

broken\_prices = [5, 3, 4, 5, 4] + [4]

Creating and Modifying a List in Python

# Range I

Often, we want to create a list of consecutive numbers. For example, suppose we want a list containing the numbers 0 through 9:

my\_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Typing out all of those numbers takes time and the more numbers we type, the more likely it is that we have a typo.

Python gives us an easy way of creating these lists using a function called range. The function range takes a single input, and **generates numbers starting at 0 and ending at the number before the input.** So, if we want the numbers from 0 through 9, we use range(10) because 10 is 1 greater than 9:

my\_range = range(10)

Just like with zip, the range function returns an object that we can convert into a list:

>>> print(my\_range)

range(0, 10)

>>> print(list(my\_range))

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Instructions

**1.**

Modify list1 so that it is a range containing numbers starting at 0 and up to, but not including, 9.

**2.**

Create a range called range1 with the numbers 0 through 7.

list1 = range(9)

range1 = range(8)

Creating and Modifying a List in Python

# Range II

We can use range to generate more interesting lists.

By default, range creates a list starting at 0. However, if we call range with two arguments, we can create a list that starts at a different number. For example, range(2, 9) would generate numbers starting at 2 and ending at 8 (just before 9):

>>> my\_list = range(2, 9)

>>> print(list(my\_list))

[2, 3, 4, 5, 6, 7, 8]

With one or two arguments, range will create a list of consecutive numbers (i.e., each number is one greater than the previous number). If we use a third argument, we can create a list that "skips" numbers. For example, range(2, 9, 2) will give us a list where each number is 2 greater than the previous number:

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

>>> print(list(my\_range2))

[2, 4, 6, 8]

We can skip as many numbers as we want! In this example, we'll start at 1 and skip 10 between each number until we get to 100:

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

>>> print(list(my\_range3))

[1, 11, 21, 31, 41, 51, 61, 71, 81, 91]

Our list stops at 91 because the next number in the sequence would be 101, which is greater than 100 (our stopping point).

Instructions

**1.**

Modify the range function that created list1 such that it:

* Starts at 5
* Has a difference of 3 between each item
* Ends **before** 15

**2.**

Create a range object called list2 that:

* Starts at 0
* Has a difference of 5 between each item
* Ends **before** 40

Solution:

list1 = range(5, 15, 3)

list2 = range(0, 40, 5)

Creating and Modifying a List in Python

# Review

So far, we have learned

* How to create a list
* How to create a list of lists using zip
* How to add elements to a list using either .append() or +
* How to use range to create lists of integers

Let's practice these skills.

Instructions

**1.**

Maria is entering customer data for her web design business. You're going to help her organize her data.

Start by turning this list of customer first names into a list called first\_names:

* Ainsley
* Ben
* Chani
* Depak

**2.**

Create an empty list called age.

**3.**

Use .append() to add the age 42 to age.

**4.**

Create a new list called all\_ages that combines age with the following list:

[32, 41, 29]

**5.**

Create a new variable called name\_and\_age that combines first\_names and all\_ages using zip.

**6.**

Create a range object called ids with an id number for each customer. There are 4 customers, so your range should be from 0 to 3.

first\_names = ['Ainsley', 'Ben', 'Chani', 'Depak']

age = []

age.append(42)

all\_ages = age + [32, 41, 29]

name\_and\_age = zip(first\_names, all\_ages)

ids = range(4)

Project

# Build a Gradebook

In this project, you will act as a student and create a gradebook to keep track of some of the subjects you've taken and grades you have received in them. To complete the project, you will need to understand how to create and combine lists, and how to add elements. On the way, you'll refresh your knowledge of basic Python syntax. If you get stuck or confused, remember that your Slack community is there to help!

This project is not graded, and you do not need to submit it anywhere. If you would like to check your results, the [solution code can be found here.](https://s3.amazonaws.com/codecademy-content/programs/programming-with-python/On-platform+solutions/gradebook.py)

Learn Python Lists

# Python Gradebook

You are a student and you are trying to organize your subjects and grades using Python. Let's explore what we've learned about lists to organize your subjects and scores.

### Create Some Lists:

1.

Create a list called subjects and fill it with the classes you are taking:

* "physics"
* "calculus"
* "poetry"
* "history"

Solution:

last\_semester\_gradebook = [("politics", 80), ("latin", 96), ("dance", 97), ("architecture", 65)]

subjects = ['physics', 'calculas', 'poetry', 'history']

grades = [98, 97, 85, 88]

subjects.append('Computer science')

grades.append(100)

gradebook = list(zip(grades, subjects))

gradebook.append(("visual arts", 93))

print(list(gradebook))

full\_gradebook = gradebook + last\_semester\_gradebook

print(full\_gradebook)

Lesson

# Python Lists II

About 35 minutes

This second lesson on lists in Python will teach you some more complex operations to use. By the end of the lesson, you will know how to sort lists and select certain ranges from them. You will get more practice with creating and modifying lists, so that the syntax you have learned gets solidified.

Working with Lists in Python

# Operations on Lists

Now that we know how to create a list, we can start working with existing lists of data.

In this lesson, you'll learn how to:

* Get the length of a list
* Select subsets of a list (called slicing)
* Count the number of times that an element appears in a list
* Sort a list of items

Instructions

When you're ready, continue to the next exercise.

Working with Lists in Python

# Length of a List

Often, we'll need to find the number of items in a list, usually called its length.

We can do this using the function len. When we apply len to a list, we get the number of elements in that list:

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

print(len(my\_list))

This would yield:

5

Instructions

**1.**

Calculate the length of list1 and save it to the variable list1\_len.

**2.**

Use print to examine list1\_len.

**3.**

Change the range command that generates list1 so that it skips 3 instead of 2 between items.

How does this change list1\_len?

Solution:

list1 = range(2, 20, 3)

print(len(list1))

list1\_len=len(list1)

Working with Lists in Python

# Selecting List Elements I

Chris is interviewing candidates for a job. He will call each candidate in order, represented by a Python list:

calls = ['Ali', 'Bob', 'Cam', 'Doug', 'Ellie']

First, he'll call 'Ali', then 'Bob', etc.

In Python, we call the order 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 that list:

| **Element** | **Index** |
| --- | --- |
| 'Ali' | 0 |
| 'Bob' | 1 |
| 'Cam' | 2 |
| 'Doug' | 3 |
| 'Ellie' | 4 |

In this example, the element with index 2 is 'Cam'.

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

>>> print(calls[2])

'Cam'

Instructions

**1.**

Use square brackets ([ and ]) to select the element with index 4 from the list employees. Save it to the variable index4.

**2.**

Use print and len to display how many items are in employees.

**3.**

Paste the following code into **script.py**:

print(employees[8])

What happens? Why?

**4.**

Selecting an element that does not exist produces an IndexError.

In the line of code that you pasted, change 8 to a different number so that you don't get an IndexError.

Solution:

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

print(employees[4])

index4=employees[4]

print(len(employees))

print(employees[6])

Working with Lists in Python

# Selecting List Elements II

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 5 elements:

list1 = ['a', 'b', 'c', 'd', 'e']

If we select the -1 element, we get the final element, 'e':

>>> print(list1[-1])

'e'

This is the same as selecting the element with index 4:

>>> print(list1[4])

'e'

Instructions

**1.**

Use print and len to display the length of shopping\_list.

**2.**

Get the last element of shopping\_list using the -1 index. Save this element to the variable last\_element.

**3.**

Now select the element with index 5 and save it to the variable element5.

**4.**

Use print to display both element5 and last\_element.

Solution:

shopping\_list = ['eggs', 'butter', 'milk', 'cucumbers', 'juice', 'cereal']

print(len(shopping\_list))

print(shopping\_list[-1])

last\_element=(shopping\_list[-1])

element5=(shopping\_list[5])

Working with Lists in Python

# Slicing Lists

Suppose we have a list of letters:

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

Suppose we want to select from b through f.

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

* start is the index of the first element that we want to include in our selection. In this case, we want to start at b, which has index 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.
* sublist = letters[1:6]

print(sublist)

This example would yield:

['b', 'c', 'd', 'e', 'f']

Notice that the element at index 6 (which is g) is not included in our selection.

Creating a selection from a list is called slicing.

Instructions

**1.**

Use print to examine the variable beginning.

How many elements does it contain?

**2.**

Modify beginning, so that it selects the first 4 elements of suitcase.

**3.**

Create a new list called middle that contains the middle two items from suitcase.

Hint:

There are 6 items in suitcase, which means its elements start at index 0 and end at index 5.

The middle two elements are index 2 and 3, so we want items starting at index 2 and up to, but not including, index 4.

Solution:

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

beginning = suitcase[0:4]

print(beginning)

middle = suitcase [2:4]

print(middle)

# Slicing Lists II

If we want to select the first 3 elements of a list, we could use the following code:

>>> fruits = ['apple', 'banana', 'cherry', 'date']

>>> print(fruits[0:3])

['apple', 'banana', 'cherry']

When **starting at the beginning** of the list, it is **also valid to omit the 0**:

>>> print(fruits[:3])

['apple', 'banana', 'cherry']

We can do something similar when selecting the last few items of a list:

>>> print(fruits[2:])

['cherry' , 'date']

We can omit the final index when selecting the final elements from a list.

If we want to select the last 3 elements of fruits, we can also use this syntax:

>>> print(fruits[-3:])

['banana', 'cherry', 'date']

We can use negative indexes to count backward from the last element.

Instructions

**1.**

Create a new list called start containing the first 3 elements of suitcase.

**2.**

Create a new list called end containing the final two elements of suitcase.

Solution:

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

start = suitcase [:3]

end = suitcase[-2:]

# Counting elements in a list

Suppose we have a list called letters that represents the letters in the word "Mississippi":

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

If we want to know how many times i appears in this word, we can use the function count:

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

print(num\_i)

This would print out:

4

**1.**

Mrs. WIlson's class is voting for class president. She has saved each student's vote into the list votes.

Use count to determine how many students voted for 'Jake'. Save your answer as jake\_votes.

**2.**

Use print to examine jake\_votes.

Solution:

votes = ['Jake', 'Jake', 'Laurie', 'Laurie', 'Laurie', 'Jake', 'Jake', 'Jake', 'Laurie', 'Cassie', 'Cassie', 'Jake', 'Jake', 'Cassie', 'Laurie', 'Cassie', 'Jake', 'Jake', 'Cassie', 'Laurie']

jake\_votes = votes.count('Jake')

print(jake\_votes)

# Sorting Lists I

Sometimes, we want to sort a list in either numerical (1, 2, 3, ...) or alphabetical (a, b, c, ...) order.

We can sort a list in place using .sort(). Suppose that we have a list of names:

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

print(names)

This would print:

['Xander', 'Buffy', 'Angel', 'Willow', 'Giles']

Now we apply .sort():

names.sort()

print(names)

And we get:

['Angel', 'Buffy', 'Giles', 'Willow', 'Xander']

Notice that sort goes after our list, names. If we try sort(names), we will get a NameError.

sort does not return anything. So, if we try to assign names.sort() to a variable, our new variable would be None:

sorted\_names = names.sort()

print(sorted\_names)

This prints:

None

Although sorted\_names is None, the line sorted\_names = names.sort() still edited names:

>>> print(names)

['Angel', 'Buffy', 'Giles', 'Willow', 'Xander']

Instructions

**1.**

Use sort to sort addresses.

**2.**

Use print to see how addresses changed.

**3.**

Remove the # in front of the line sort(names). Edit this line so that it runs without producing a NameError.

**4.**

Use print to examine sorted\_cities. Why is it not the sorted version of cities?

### Exercise 1 & 2 ###

addresses = ['221 B Baker St.', '42 Wallaby Way', '12 Grimmauld Place', '742 Evergreen Terrace', '1600 Pennsylvania Ave', '10 Downing St.']

# Sort addresses here:

addresses.sort()

print(addresses)

### Exercise 3 ###

names = ['Ron', 'Hermione', 'Harry', 'Albus', 'Sirius']

names.sort()

### Exercise 4 ###

cities = ['London', 'Paris', 'Rome', 'Los Angeles', 'New York']

sorted\_cities = cities.sort()

print(sorted\_cities)

# Sorting Lists II

A second way of sorting a list is to use sorted. sorted is different from .sort() in several ways:

1. It comes before a list, instead of after.
2. It generates a new list.

Let's return to our list of names:

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

Using sorted, we can create a new list, called sorted\_names:

sorted\_names = sorted(names)

print(sorted\_names)

This yields the list sorted alphabetically:

['Angel', 'Buffy', 'Giles', 'Willow', 'Xander']

Note that using sorted did not change names:

>>> print(names)

['Xander', 'Buffy', 'Angel', 'Willow', 'Giles']

**1.**

Use sorted to order games and create a new list called games\_sorted.

**2.**

Use print to inspect games and games\_sorted. How are they different?

Solution:

games = ['Portal', 'Minecraft', 'Pacman', 'Tetris', 'The Sims', 'Pokemon']

games\_sorted = sorted(games)

print(games)

print(games\_sorted)

# Review

In this lesson, we learned how to:

* Get the length of a list
* Select subsets of a list (called slicing)
* Count the number of times that an element appears in a list
* Sort a list of items

Instructions

**1.**

inventory is a list of items that are in the warehouse for Bob's Furniture. How many items are in the warehouse?

Save your answer to inventory\_len.

**2.**

Select the first element in inventory. Save it to the variable first.

**3.**

Select the last item from inventory and save it to the variable last.

**4.**

Select items from the inventory starting at index 2 and up to, but not including, index 6.

Save your answer to inventory\_2\_6.

**5.**

Select the first 3 items of inventory and save it to the variable first\_3.

**6.**

How many 'twin bed's are in inventory? Save your answer to twin\_beds.

**7.**

Sort inventory using .sort().

inventory = ['twin bed', 'twin bed', 'headboard', 'queen bed', 'king bed', 'dresser', 'dresser', 'table', 'table', 'nightstand', 'nightstand', 'king bed', 'king bed', 'twin bed', 'twin bed', 'sheets', 'sheets', 'pillow', 'pillow']

inventory\_len=(len(inventory))

first = inventory[0]

last = inventory[-1]

inventory\_2\_6 = inventory[2:6]

first\_3 = inventory [0:3]

twin\_beds = inventory.count('twin bed')

inventory.sort()

# Len's Slice

You work at Len's Slice, a new pizza joint in the neighborhood. You are going to use your knowledge of Python lists to organize some of your sales data.

Mark the tasks as complete by checking them off

### Make Some Pizzas

1.

To keep track of the kinds of pizzas you sell, create a list called toppings that holds the following:

* pepperoni
* pineapple
* cheese
* sausage
* olives
* anchovies
* mushrooms

It should look something like:

toppings = ["pepperoni", "pineapple", "cheese", "sausage", "olives", "anchovies", "mushrooms"]

2.

To keep track of how much each kind of pizza slice costs, create a list called prices that holds:

* 2
* 6
* 1
* 3
* 2
* 7
* 2

You don't need the quotes when you are dealing with integers:

prices = [2, 6, 1, 3, 2, 7, 2]

3.

Find the length of the toppings list and store it in a variable called num\_pizzas.

You can use len() to find the length of a list:

len(toppings)

4.

Print the string "We sell X different kinds of pizza!", with num\_pizzas where the X is.

To add a variable to a string, you can use the syntax:

"My age is " + str(my\_age)

**Note:** You have to cast the number as a string before you add it to another string!

The output should look like:

We sell 7 different kinds of pizza!

5.

Combine the two lists into a list called pizzas that has the structure:

[(price\_0, topping\_0), (price\_1, topping\_1), (price\_2, topping\_2), ...]

You can use zip() to combine two lists like this:

nested\_list = list(zip(list\_1, list\_2))

In this case:

pizzas = list(zip(prices, toppings))

6.

Print pizzas.

Does it look the way you expect?

The output should look like:

[(2, 'pepperoni'), (6, 'pineapple'), (1, 'cheese'), (3, 'sausage'), (2, 'olives'), (7, 'anchovies'), (2, 'mushrooms')]

### Sorting and Slicing Pizzas

7.

Sort pizzas so that the pizzas with the lowest prices are at the start of the list.

You can sort a list from low to high by using .sort():

my\_list.sort()

For a list of sub-lists, like pizzas, .sort() will compare against the first element in each sublist.

8.

Store the first element of pizzas in a variable called cheapest\_pizza.

To get an element of a list, use the syntax [n], where n is the index of the item you want to get. Remember that list indices start at zero!

second\_item = your\_list[1]

9.

A man in a business suit walks in and shouts "I will have your MOST EXPENSIVE pizza!"

Get the last item of the pizzas list and store it in a variable called priciest\_pizza.

To get the last element of a list, use the syntax [-1]

last\_item = your\_list[-1]

10.

Three mice walk into the store. They don't have much money (they're mice), but they do each want different pizzas.

Slice the pizzas list and store the 3 lowest cost pizzas in a list called three\_cheapest.

To get the first n items of a list, use [:n]. For example:

new\_list = my\_list[:2]

would store the first two items of my\_list in new\_list.

11.

Print the three\_cheapest list.

12.

Your boss wants you to do some research on $2 slices.

Count the number of occurrences of 2 in the prices list, and store the result in a variable called num\_two\_dollar\_slices. Print it out.

You can use .count to find the number of occurrences of a value in a list:

my\_list = ["a", "a", "b"]

number\_of\_as = my\_list.count("a")

# number\_of\_as is 2

Solution:

toppings = ['pepperoni', 'pineapple', 'cheese', 'sausage', 'olives', 'anchovies', 'mushrooms']

prices = [2, 6, 1, 3, 2, 7, 2]

num\_pizzas=(len(toppings))

print('We sell ' +str(num\_pizzas)+ " different kinds of pizzas!")

pizzas = list(zip(prices, toppings))

print(pizzas)

pizzas.sort()

cheapest\_pizza = pizzas[0]

priciest\_pizza = pizzas[-1]

three\_cheapest = pizzas[:3]

print(three\_cheapest)

num\_two\_dollar\_slices = prices.count(2)