# Week 3

# \*Update for Week 3 (Unit 2: Code Organization and Reuse) ~\*

# How's everyone doing? :smile: Today starts \*Week 3\* in the course, and we'll be wrapping up \*Unit 2\*. There will be \*1 code challenge\*, \*1 project\*, and \*1 \_optional\_ project\*. In addition, you'll get a chance to get Python3 set up on your own computer!

# ----

# This article will guide you through how to install Python3, install Python packages, and run Python code: https://www.codecademy.com/articles/install-python3

# It may also be helpful to complete the short optional prework course on the command-line if you haven't already: https://www.codecademy.com/courses/learn-the-command-line/lessons/navigation/exercises/your-first-command

# Or, if you're a Windows user, this might be helpful! https://www.computerhope.com/issues/chusedos.htm

# The second article from this week will introduce you to using Jupyter Notebook (which is used to complete the optional project): https://www.codecademy.com/articles/how-to-use-jupyter-notebooks-py3

# In addition, you could check out this video for some great tips and information: https://www.youtube.com/watch?v=jZ952vChhuI

# Finally, this week may also be a good time to download a text editor to use to write your code on your own computer. Take a look here for some recommendations! https://www.codecademy.com/articles/text-editors-sublime-atom

# ----

# A few notes on the topic of Control Flow for this week about the difference between `if`s and `elif`s.

# When we have an `if`/`elif`/(`else`) flow, exactly \*1\* of the blocks will execute:

# ```a = 5

# if a == 5:

# print('equals five') # prints

# elif a < 10:

# print('less than ten') # does not print even though 5 < 10```

# Compared to if we only use `if`, each `if` will be independent of what comes before it:

# ```a = 5

# if a == 5:

# print('equals five') # prints

# if a < 10:

# print('less than ten') # prints as well```

# A quick example of when we would want to use `elif`:

# ```a = 'y'

# if a == 'y':

# print('You said yes') # prints

# elif a == 'n':

# print('You said no')

# else:

# print('Not a valid answer - please enter y or n')```

# If we only used `if`s and no `elif`, this would not work as expected:

# ```a = 'y'

# if a == 'y':

# print('You said yes') # prints

# if a == 'n':

# print('You said no')

# else:

# print('Not a valid answer - please enter y or n') # also prints!```

# ----

# If you have any questions on this week's material, #unit\_2\_week\_3 is the place to post! Since this week also covers getting Python3 set up on your own machines, feel free to post any issues you run into in the channel so we could troubleshoot. Or if you have any tips for others during this process, please share as well! :slightly\_smiling\_face:

# Control Flow Project

## Project

## Sal's Shipping

In this project, you will help create some functions to optimize the cost of deliveries from Sal's Shipping using your knowledge of control flow.

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/sals_shipping.py)

Python: Control Flow

# Sal's Shipping

Sal runs the biggest shipping company in the tri-county area, Sal's Shippers. Sal wants to make sure that every single one of his customers has the best, and most affordable experience shipping their packages. In this project, you'll build a program that will take the weight of a package and determine the cheapest way to ship that package using Sal's Shippers.

Sal's Shippers has several different options for a customer to ship their package. They have ground shipping, which is a small flat charge plus a rate based on the weight of your package. Premium ground shipping, which is a much higher flat charge, but you aren't charged for weight. They recently also implemented drone shipping, which has no flat charge, but the rate based on weight is triple the rate of ground shipping.

Here are the prices:

**Ground Shipping**

| **Weight of Package** | **Price per Pound** | **Flat Charge** |
| --- | --- | --- |
| 2 pounds or less | $1.50 | $20.00 |
| Over 2 pounds but less than or equal to 6 pounds | $3.00 | $20.00 |
| Over 6 pounds but less than or equal to 10 pounds | $4.00 | $20.00 |
| Over 10 pounds | $4.75 | $20.00 |

**Drone Shipping**

| **Weight of Package** | **Price per Pound** | **Flat Charge** |
| --- | --- | --- |
| 2 pounds or less | $4.50 | $00.00 |
| Over 2 pounds but less than or equal to 6 pounds | $9.00 | $00.00 |
| Over 6 pounds but less than or equal to 10 pounds | $12.00 | $00.00 |
| Over 10 pounds | $14.25 | $00.00 |

**Premium Ground Shipping**

Flat charge: $125.00

Write a program that asks the user for the weight of their package and then tells them which method of shipping is cheapest and how much it will cost to ship their package using Sal's Shippers.

### Finding the Cheapest Shipping Method

1.

First off, we need to know how much it costs to ship a package of given weight by normal ground shipping.

Write a function for the cost of ground shipping. This function should take one parameter, weight, and return the cost of shipping a package of that weight.

2.

A package that weighs 8.4 pounds should cost $53.60 to ship with normal ground shipping.

Test that your ground shipping function gets the same value.

3.

We'll also need to make sure we include the price of premium ground shipping in our code.

Create a variable for the cost of premium ground shipping.

Note: this does not need to be a function because the price of premium ground shipping does not change with the weight of the package.

4.

Write a function for the cost of drone shipping. This function should take one parameter, weight, and return the cost of shipping a package of that weight.

5.

A package that weighs 1.5 pounds should cost $6.75 to ship by drone.

Test that your drone shipping function gets the same value.

6.

Using those two functions for ground shipping cost and drone shipping cost, as well as the cost of premium ground shipping, write a function that takes one parameter, weight and prints a statement that tells the user

* The shipping method that is cheapest.
* How much it would cost to ship a package of said weight using this method.

7.

Great job! Now, test your function!

What is the cheapest method of shipping a 4.8 pound package and how much would it cost?

What is the cheapest method of shipping a 41.5 pound package and how much would it cost?

(See hint for answers)

The cheapest way to ship 4.8 pound package is using ground shipping and it will cost $34.40.

The cheapest way to ship a 41.5 pound package is using premium ground shipping and it will cost $125.00.

Report a Bug

Solution:

# Calculates the cost of normal ground shipping given weight

def ground\_shipping(weight):

if weight <= 2:

cost = weight \* 1.5 + 20

return cost

elif weight <= 6:

cost = weight \* 3.00 + 20

return cost

elif weight <= 10:

cost = weight \* 4.00 + 20

return cost

else:

cost = weight \* 4.75 + 20

return cost

# Cost of premium ground shipping is independent of weight

prem\_ground\_shipping = 125.00

# Calculates the cost of drone shipping given weight

def drone\_shipping (weight):

if weight <= 2:

cost = weight \* 4.5

return cost

elif weight <= 6:

cost = weight \* 9.00

return cost

elif weight <= 10:

cost = weight \* 12.00

return cost

else:

cost = weight \* 14.25

return cost

# Compares the cost of each shipping method and prints the cheapest method and how much it costs

def cheapest\_shipping(weight):

if (drone\_shipping(weight) < prem\_ground\_shipping) and (drone\_shipping(weight) < ground\_shipping(weight)):

print("You should ship using drone shipping, it will cost $" + str(drone\_shipping(weight)))

elif ground\_shipping(weight) < prem\_ground\_shipping:

print("You should ship using ground shipping, it will cost $" + str(ground\_shipping(weight)))

else:

print("You should ship using premium ground shipping, it will cost $" + str(prem\_ground\_shipping))

# Lesson

# Control Flow - Code Challenge

About 1 minute

Now you have learned about the basics of Python syntax, control flow, and how to create functions that run repeatable blocks of code.

Let's reinforce these concepts with a series of practice problems.

Code Challenge: Control Flow

# Introduction

This lesson will help you review Python functions by providing some challenge exercises involving control flow.

As a refresher, function syntax looks like this:

def some\_function(some\_input1, some\_input2):

… do something with the inputs …

return output

For example, a function that takes a number and checks to see if it is greater than ten would look like this

def greaterThanTen(number):

if number > 10:

return True

else:

return False

And this would produce output like:

>>> greaterThanTen(20)

True

>>> greaterThanTen(-10)

False

>>> greaterThanTen(10)

False

When you're ready to do this series of short function challenges, continue on to the rest of the lesson!

Task 1 Solution:

# Write your in\_range function here:

def in\_range(num, lower, upper):

if(num >= lower and num <= upper):

return True

return False

# Uncomment these function calls to test your in\_range function:

print(in\_range(10, 10, 10))

# should print True

print(in\_range(5, 10, 20))

# should print False

# Movie Review

movie\_review(rating)

**1.**

Create a function named movie\_review that has one parameter named rating.

If rating is less than or equal to 5, return "Avoid at all costs!". If rating is between 5 and 9, return "This one was fun.". If rating is 9 or above, return "Outstanding!"

Task 2 Solution:

# Write your movie\_review function here:

def movie\_review(rating):

if rating <= 5:

return "Avoid at all costs!"

elif rating < 9:

return "This one was fun."

else:

rating >=9

return "Outstanding!"

# Uncomment these function calls to test your movie\_review function:

print(movie\_review(9))

# should print "Outstanding!"

print(movie\_review(4))

# should print "Avoid at all costs!"

print(movie\_review(6))

# should print "This one was fun."

Code Challenge: Control Flow

# Twice as Large

twice\_as\_large(num1, num2)

**1.**

Create a function named twice\_as\_large that has two parameters named num1 and num2.

Return True if num1 is more than double num2. Return False otherwise.

**Solution:**

# Write your twice\_as\_large function here:

def twice\_as\_large(num1, num2):

if num1 > 2 \* num2:

return True

else:

return False

# Uncomment these function calls to test your twice\_as\_large function:

print(twice\_as\_large(10, 5))

# should print False

print(twice\_as\_large(11, 5))

# should print True

Code Challenge: Control Flow

# Power

large\_power(base, exponent)

1. Create a function named large\_power that takes two parameters named base and exponent.

If base raised to the exponent is greater than 5000, return True, otherwise return False

# Write your large\_power function here:

def large\_power (base, exponent):

if base \*\* exponent > 5000:

return True

else:

return False

# Uncomment these function calls to test your large\_power function:

print(large\_power(2, 13))

# should print True

print(large\_power(2, 12))

# should print False

# Divisible By Ten

divisible\_by\_ten(num)

**1.**

Create a function called divisible\_by\_ten that has one parameter named num.

The function should return True if num is divisible by 10, and False otherwise. Consider how to use modulo (%) to check for divisibility.

Hint:

If a % b is 0, then a is divisible by b. Use that in combination with an if statement to return the correct response.

Solution:

# Write your divisible\_by\_ten function here:

def divisible\_by\_ten (num):

if num %10:

return False

else:

return True

# Uncomment these function calls to test your divisible\_by\_ten function:

print(divisible\_by\_ten(20))

# should print True

print(divisible\_by\_ten(25))

# should print False

# Max Number

max\_num(num1, num2, num3)

**1.**

Create a function called max\_num that has three parameters named num1, num2, and num3.

The function should return the largest of these three numbers. If any of two numbers tie as the largest, you should return "It's a tie!".

Hint:

Use the following control flow. If none of the if or else if statements were true, then you know there must have been a tie for the largest number.

if num1 > num2 and \_\_\_\_\_:

return num1

elif \_\_\_\_\_ and \_\_\_\_\_:

return num2

elif \_\_\_\_\_ and \_\_\_\_\_:

return num3

else:

return "It's a tie!"

Solution:

# Write your max\_num function here:

def max\_num(num1, num2, num3):

if num1 > num2 and num1 > num3:

return num1

elif num2 > num1 and num2 > num3:

return num2

elif num3 > num1 and num3 > num2:

return num3

else:

return "It's a tie!"

# Uncomment these function calls to test your max\_num function:

print(max\_num(-10, 0, 10))

# should print 10

print(max\_num(-10, 5, -30))

# should print 5

print(max\_num(-5, -10, -10))

# should print -5

print(max\_num(2, 3, 3))

# should print "It's a tie"

# Over Budget

over\_budget(budget, food\_bill, electricity\_bill, internet\_bill, rent)

**1.**

Create a function called over\_budget that has five parameters named budget, food\_bill, electricity\_bill, internet\_bill, and rent.

The function should return True if budget is less than the sum of the other four parameters. You've gone over budget! Return False otherwise.

Hint:

You can do all of the math inside the if statement like this

if a < b + c + d:

return True

# Write your over\_budget function here:

def over\_budget (budget, food\_bill, electricity\_bill, internet\_bill, rent):

if budget < food\_bill + electricity\_bill + internet\_bill + rent:

return True

else:

return False

# Uncomment these function calls to test your over\_budget function:

print(over\_budget(100, 20, 30, 10, 40))

# should print False

print(over\_budget(80, 20, 30, 10, 30))

# should print True

# Always False

always\_false(num)

**1.**

Create a function named always\_false that has one parameter named num.

Using an if statement, your variable num, and the operators >, and <, make it so your function will return False no matter what number is stored in num.

An if statement that is always false is called a contradiction. You will rarely want to do this while programming, but it is important to realize it is possible to do this.

Hint:

Can num ever be less than 0 and greater than 0 at the same time? Use this fact to ensure you will always return the same thing.

Solution:

# Write your always\_false function here:

def always\_false(num):

if (num > 0 and num < 0):

return True

else:

return False

# Uncomment these function calls to test your always\_false function:

print(always\_false(0))

# should print False

print(always\_false(-1))

# should print False

print(always\_false(1))

# should print False

# Not Equal

not\_sum\_to\_ten(num1, num2)

Instructions

**1.**

Create a function named not\_sum\_to\_ten that has two parameters named num1 and num2.

Return True if num1 and num2 do not sum to 10. Return False otherwise.

Community Forums

Hint:

Use != to check to make sure num1 + num2 doesn't equal ten.

Solution:

# Write your not\_sum\_to\_ten function here:

def not\_sum\_to\_ten(num1, num2):

if num1 + num2 != 10:

return True

else:

return False

# Uncomment these function calls to test your not\_sum\_to\_ten function:

print(not\_sum\_to\_ten(9, -1))

# should print True

print(not\_sum\_to\_ten(9, 1))

# should print False

print(not\_sum\_to\_ten(5,5))

# should print False

# Same Name

same\_name(your\_name, my\_name)

Instructions

**1.**

Create a function named same\_name that has two parameters named your\_name and my\_name.

If our names are identical, return True. Otherwise, return False.

Hint:

You can compare strings using == in the same way you compare numbers.

Solution:

# Write your same\_name function here:

def same\_name(your\_name, my\_name):

if your\_name == my\_name:

return True

else:

return False

# Uncomment these function calls to test your same\_name function:

print(same\_name("Colby", "Colby"))

# should print True

print(same\_name("Tina", "Amber"))

# should print False

# Installing Python 3 and Jupyter

## Background

Codecademy's learning environment allows you to enter Python code and receive feedback on whether or not the code you entered is correct for a given exercise. In this article, we'll walk you through how to install Python so that you can write and run Python code outside of Codecademy and on your computer!

## Why build outside of Codecademy?

The programming world is massive, and it’s impossible to teach everything in one place. Although Codecademy excels at teaching you how to code via interactive lessons, we'd also like for you to learn how to code on your computer so that you can create personal projects (and perhaps share them with the world)!

## How long does installing Python take?

Creating a development environment for yourself on your own computer is important so that you can continue to build upon your Python knowledge outside of the Codecademy platform. Unfortunately, every computer is different and some take a long time to troubleshoot. You should budget no less than **2 hours** in order to go through this installation process. If you happen to get through it faster, that's great. But, since this article can only offer general instructions, you may need to do some investigation and research about how to get Python to run on your computer. Be prepared, and after you get through this you'll be ready to take on the world.

## Contents

In this article, we'll cover the following topics:

* What is Anaconda?
* What is Miniconda?
* Should I Install Anaconda, Miniconda, or just Python?
* Installation: Python
* Running Python Code
* Installing and Managing Python Packages Using pip3
* Installation: Miniconda
* Was the Installation Successful?
* Managing Packages in Anaconda / Miniconda
* pip3 vs. conda

## What is Anaconda?

Anaconda is an open-source Python distribution for large-scale data analytics (provided by Continuum Analytics, Inc.). It additionally provides you with many of the tools you need to analyze large sets of data. When installed, Anaconda includes:

* The core Python language (you can use which version)
* Over 1000 packages, many for data science
* Package management with conda
* IPython
* Much more

## What is Miniconda?

Miniconda is a slimmed-down version of Anaconda. The Anaconda download is large (a few gigabytes) and can take quite some time to download and install. Miniconda, on the other hand, is a smaller alternative. It includes only the basic requirements and allows you to install packages as-needed, thereby decreasing the size and time of the download.

## Should I Install Anaconda, Miniconda, or just Python?

Both Anaconda and Miniconda also install Python, so you can install Anaconda, Miniconda, **or** Python.

Anaconda and Miniconda make it easier to install Jupyter notebooks, which we make extensive use of at Codecademy. It's possible to install Jupyter without any additional software on top of your Python installation. Installing Python vs. Anaconda vs. Miniconda is ultimately your choice. We recommend installing Miniconda to decrease the amount of time required to set up everything.

## Installation: Python

To install Python, follow these steps:

1. Navigate to the Python downloads page: [Python downloads](https://www.python.org/downloads/).
2. Click on the link/button to download Python 3.6.x.
3. Follow the installation instructions (leave all defaults as-is).
4. Open your terminal again and type the command cd. Next, type the command python. The Python interpreter should respond with the version number. If you're on a Windows machine, you will likely have to navigate to the folder where Python is installed (for example, Python36, which is the default) for the python3 command to function.

## Running Python Code

Congrats! You should have Python 3.6 installed now. Let's run some Python code!

**Mac OS**:

1. Open your terminal and type cd (if you're using Windows, navigate to the Python36 folder instead).
2. Create a file called mycode.py (make sure it has a .py extension).
3. Open mycode.py using your favorite [text editor](https://www.codecademy.com/articles/text-editors-sublime-atom).
4. Add the following code to the file and save it:

print("I'm running Python code on my own!")

5. In your terminal, type the following command and press "Enter" (or "Return") on your keyboard to run your code (again, if you're on Windows you will need to navigate to the folder you installed Python 3.6 in):

python3 mycode.py

6. Your terminal should output the following message:

I'm running Python code on my own!

Congratulations! You just ran Python code on your computer!

## Installing and Managing Python Packages Using pip3

With Python, you can build just about anything, from simple scripts to full applications. The Python language, however, doesn't come pre-installed with all of the fancy features you might want (or require). When you need particular functionality, you can look toward Python packages. A package structures Python modules, which contain pre-written code that other developers have created for you. Modules are handy when you are looking for specific functionality.

You can use pip3, Python's package manager, to install and manage Python packages. pip3 gets installed along with Python.

You can use pip3 to install packages, like so:

pip3 install jupyter

In the example above, pip3 will install the [Jupyter package](https://jupyter.readthedocs.io/en/latest/index.html), a popular package (among many) used for creating "notebooks" for running Python.

There are a multitude of Python packages, which you can find on [PyPI](https://pypi.org/) — the Python Package Index — the official repository for third-party software for the Python programming language. PyPI is where pip3 grabs Python packages from when you use pip3 to install a new package on your computer.

You can use pip3 for a variety of other things as well, which you can learn about through a quick search on the web.

## Installation: Miniconda

To install Miniconda, follow these steps:

1. Navigate to the Miniconda download page: [Miniconda](https://conda.io/miniconda.html)
2. Select the Python 3.6 installer for your computer's operating system.
3. Locate the installer that you downloaded using Explorer (Windows) or Finder (Mac OS).
4. Run the installer. Use the following instructions based on your computer's operating system:

**Mac OS**:

1. You may receive a notification about XCode requiring additional component. Click "Install" and enter your password to proceed.
2. Open your terminal and navigate to the folder where you downloaded the installer. Type the following command in the terminal and press "Return" on your keyboard:

bash miniconda-filename.sh

miniconda-filename.sh is a fictional file name in the example above. Your file name will look something like Miniconda3-latest-MacOSX-x86\_64.sh.

3. Follow all instructions in the terminal (you can press Enter as-needed and type yes when necessary).

**Windows**:

1. Follow the installation instructions provided by the installer.

## Was the Installation Successful?

To test whether your installation was successful (regardless of your computer's operating system), type the following command into your terminal:

conda list

You should see a list of all the packages that Miniconda installed. If you're on a computer that uses Windows, you may have to first navigate to the folder where you installed Miniconda for the conda list command to function properly.

If it works out, you can install Jupyter notebooks by running the following command:

conda install jupyter

Congrats! You now have Miniconda (with Python 3.6) installed on your computer as well as Jupyter notebooks. You're ready to code!

## Managing Packages in Anaconda / Miniconda

With Python, you can build just about anything, from simple scripts to full applications. The Python language, however, doesn't come pre-installed with all of the fancy features you might want (or require), even when installed using Anaconda or Miniconda. When you need particular functionality, you can look toward Python packages. A package structures Python modules, which contain pre-written code that other developers have created for you. Modules are handy when you are looking for specific functionality.

Usually, pip3 is used to install and manage Python packages. It is the package manager for the official Python distribution. If you installed Python with Anaconda or Miniconda, however, the package manager is not pip3, the package manager is conda.

To learn more about conda, visit the Conda documentation at the following link:

* [Getting Started with Conda](https://conda.io/docs/user-guide/getting-started.html).

## pip vs. conda

Although conda is the package manager for Anaconda (and Miniconda), pip3 is also included with Anaconda (and Miniconda). Certain packages will not be available from conda or Anaconda.org. When this happens, you can use pip3 to install packages.

Be careful when using pip3, though. Using pip3 to install packages available to conda can result in installation errors.

## Conclusion

So far, you've been writing Python code on Codecademy. Your learning journey, however, is not complete unless you can also write Python code outside of Codecademy, on your computer. If you want to run your own Python programs, we recommend installing Miniconda (with Python 3.6), and then using conda to install certain packages. Have fun coding!

# How To Use Jupyter Notebooks

Reading Time: About 6 minutes

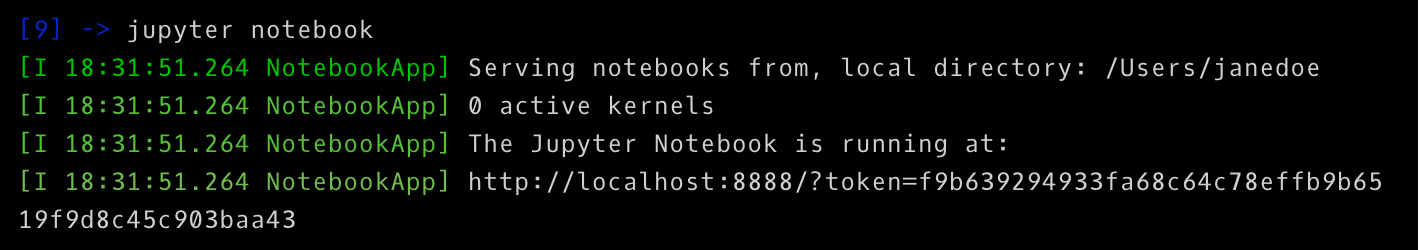
## INTRODUCTION

Jupyter Notebooks are a powerful way to write and iterate on your Python code for data analysis. Rather than writing and re-writing an entire program, Jupyter Notebooks allow you to write code in separate blocks (or “cells”) and run each block of code individually. Then, if you need to make a change, you can go back and make your edit and rerun the program again, all in the same window.

Jupyter Notebook is built off of IPython, an interactive way of running Python code in the terminal using the REPL model (Read-Eval-Print-Loop). The IPython Kernel runs the computations and communicates with the Jupyter Notebook front-end interface. It also allows Jupyter Notebook to support multiple languages. Jupyter Notebooks extend IPython through additional features, like storing your code and output and allowing you to keep markdown notes.

#### Launch a Notebook

To launch a Jupyter notebook, open your terminal and navigate to the directory where you would like to save your notebook. Then type the command jupyter notebook and the program will instantiate a local server at localhost:8888 (or another specified port).



A browser window should immediately pop up with the Jupyter Notebook interface, otherwise, you can use the address it gives you. The notebooks have a unique token since the software uses pre-built Docker containers to put notebooks on their own unique path. To stop the server and shutdown the kernel from the terminal, hit control-C twice.

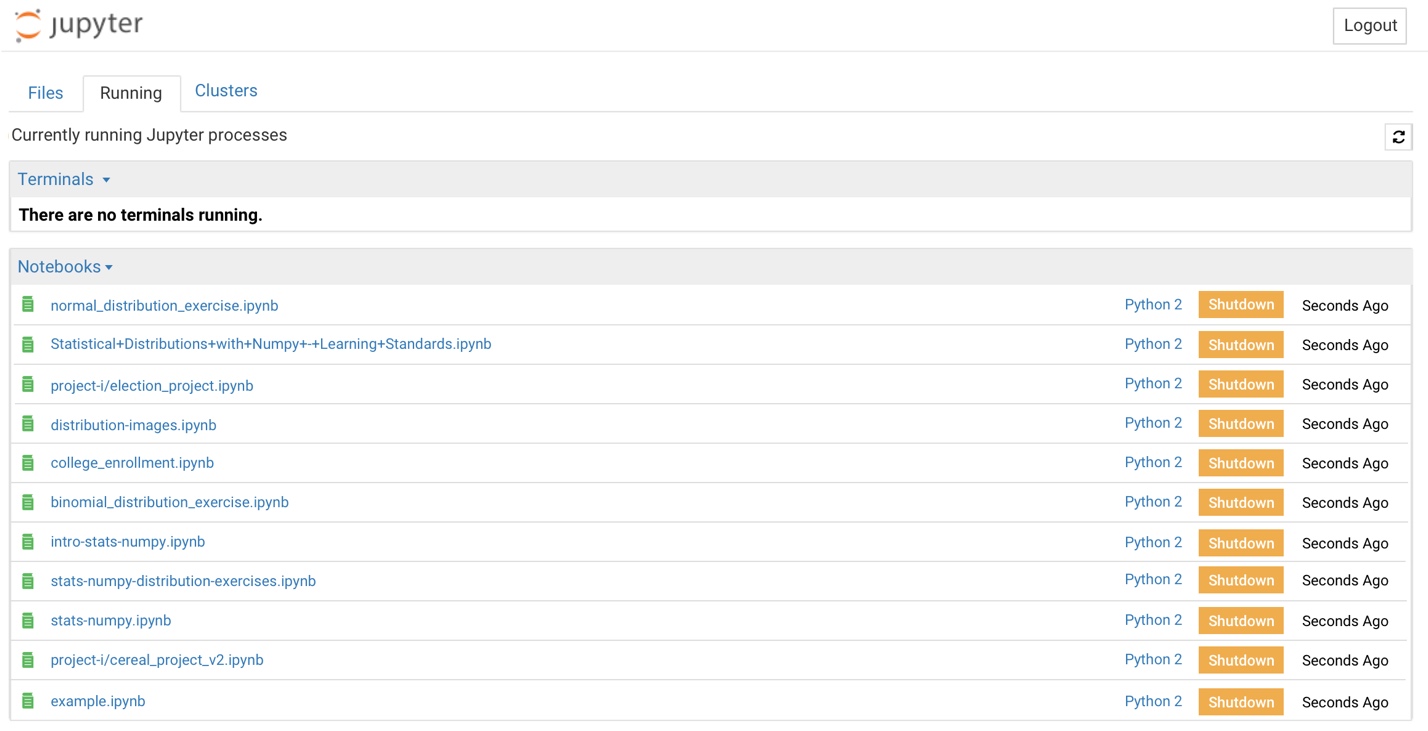
#### Jupyter Interface

Now you're in the Jupyter Notebook interface, and you can see all of the files in your current directory. All Jupyter Notebooks are identifiable by the **notebook icon** next to their name. If you already have a Jupyter Notebook in your current directory that you want to view, find it in your files list and click it to open.



To create a new notebook, go to **New** and select **Notebook - Python 3**. If you have other Jupyter Notebooks on your system that you want to use, you can click **Upload** and navigate to that particular file.

Notebooks currently running will have a green icon, while non-running ones will be grey. To find all currently running notebooks, click on the **Running** tab to see a list.

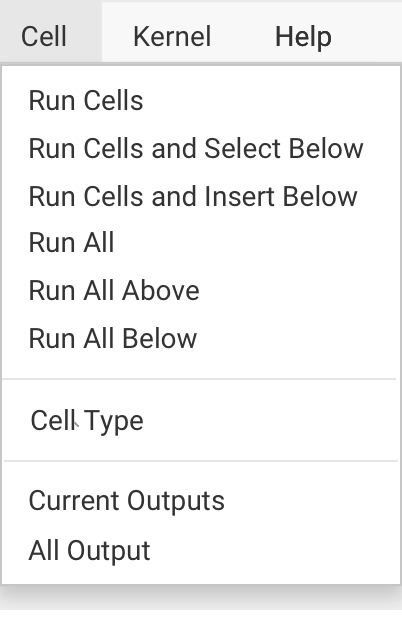


#### Inside the Notebook

When you open a new Jupyter notebook, you'll notice that it contains a cell.



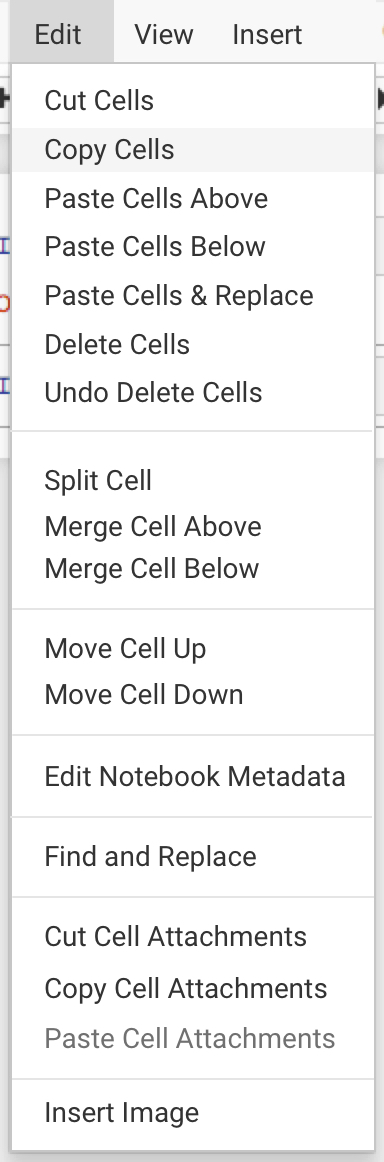
Cells are how notebooks are structured and are the areas where you write your code. To run a piece of code, click on the cell to select it, then press SHIFT+ENTER or press the play button in the toolbar above. Additionally, the **Cell** dropdown menu has several options to run cells, including running one cell at a time or running all cells at once.



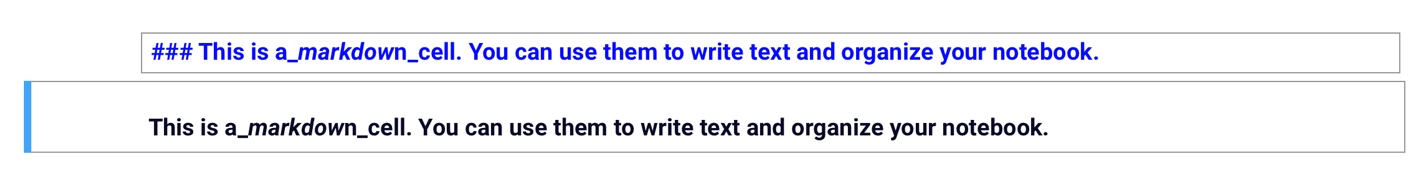
After you run a cell, the output of the cell's code will appear in the space below. To stop running a piece of code, press the stop button.



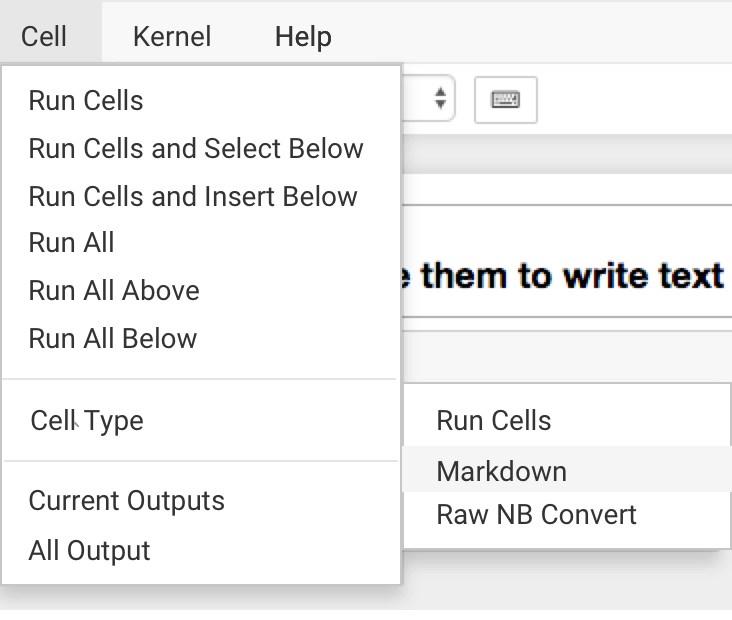
To create new cells, use the plus (+) button in the toolbar or hit SHIFT+ENTER on the last cell in the Notebook. To cut, copy, delete or just generally edit cells - select the cell you want to modify and go to the **Edit** button in the navigation bar to see your options.



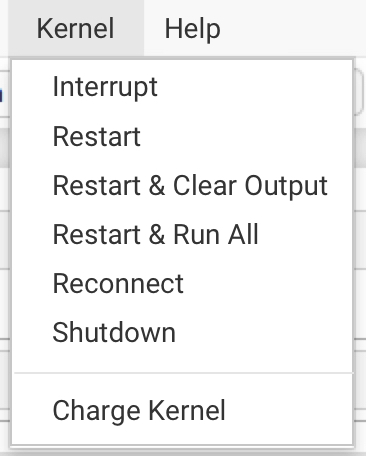
In addition to running lines of code, you can also include **text-only** cells that use Markdown to format and organize your notebooks.



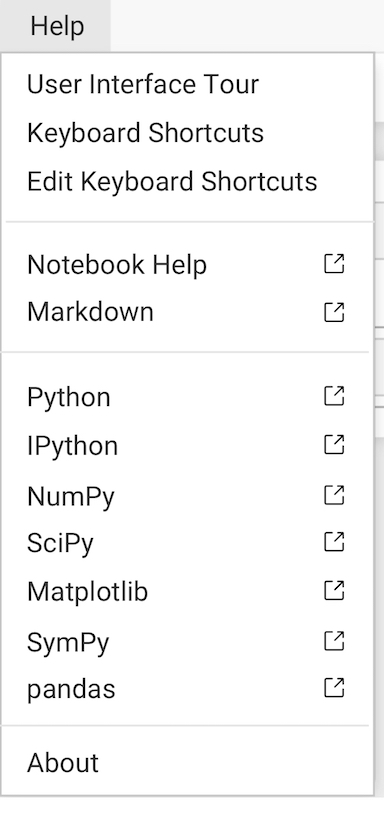
When you create a new cell, it will default to being a **Code** cell. To create a cell that uses markdown, click on the **Cell** menu from the navigation bar, scroll down to **Cell Type** and choose Markdown.



Occasionally, you might need to restart the kernel. Head to the **Kernel** dropdown and hit **Restart**. To shut down a kernel, you can click **Shutdown**, which will have a dialogue process asking if that's what you would like to do. To force an immediate shutdown, go to the **File** dropdown and click **Close and Halt** and the browser window will close itself. Restarting and shutting down kernels will affect your variables, so be careful.



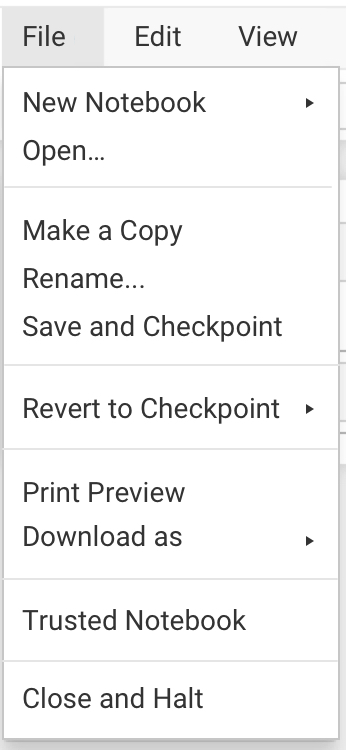
In the **Help** dropdown, you'll find useful information such as keyboard shortcuts as well as links to different documentation for modules such as Numpy, SciPy, and Matplotlib.



The **toolbar** has several shortcut buttons for popular actions. From left to right: save, add a new cell, cut selected cells, copy selected cells, paste cells below, move selected cells up, move selected cells down, run, interrupt the kernel, restart the kernel, a dropdown that allows you to change the cell type, and a shortcut to open the command palette.

toolbar

Jupyter Notebook files are saved as you go. They will exist in your directory as a JSON file with the extension .ipynb. You can also export Jupyter Notebooks in other formats, such as HTML. To do so, go to the File menu, scroll down to Download as and select the type of file you're looking for. A popup will appear asking where you would like this new file to download. Once you've navigated to the appropriate directory, click Save and Checkpoint.



#### SUMMARY

As we've seen, Jupyter Notebook files are very useful. Its interface allows you to navigate using your mouse with dropdown menus and buttons, or by keyboard shortcuts. They allow you to run small segments of code at a time, save them in their current state, or restart and have them return to their original state.In addition to running code, we can also use markdown to neatly organize our notebooks so they are presentable to others.

If you're interested in learning more about Jupyter Notebooks, [read their documentation](http://jupyter.readthedocs.io/en/latest/). To try out a notebook in your browser, go to <https://try.jupyter.org/>.

# Customer Service Bot

Reading Time: About 2 minutes

Welcome to your first cumulative project! In this project, you'll be working on your own computer and using your Python function and control flow knowledge to build a Customer Service Bot for a fictional internet service provider called the Definitely Not Sinister Cable Company, or DNS Cable Company for short.

**This project is totally optional; if you’re busy this week, you can skip it!**

The DNS Cable Company is a wonderful internet service provider that cares about its customers! They care so much about their customers, they want to try to solve their problems using a simple text bot. They've provided their very specific and detailed instructions on how they want the bot to function in a Juypter Notebook, that they have sent over to you to work in. Follow the steps below to get started with your project!

## Working on Your Computer

1. If you’ve never used the command line, we recommend taking the [Learn the Command Line course](https://www.codecademy.com/learn/learn-the-command-line).
2. Install Python by following the directions in this article on [Installing Python](https://www.codecademy.com/articles/install-python3).
3. Learn about [Jupyter Notebooks](https://www.codecademy.com/articles/how-to-use-jupyter-notebooks-py3), a cool way of combining Python code with explanations or instruction in a web terminal.
4. Download the [Customer Service Bot project](https://s3.amazonaws.com/codecademy-content/programs/programming-with-python/customer_service_bot.zip).
5. Unzip it by double-clicking on it.
6. In the terminal, navigate to the directory containing the project, and type:

jupyter notebook

This should open a browser tab.

1. Click on customer\_service\_bot.ipynb in the browser tab. This will open up your Jupyter Notebook.
2. Follow the steps in the Jupyter Notebook. If you get stuck, you can look at customer\_service\_bot\_solutions.ipynb for the answer.