Python Survival Guide

Week 1

Every week, you will receive a link to videos that cover the topics that you will need to know for the week. The videos will come from the O'Reilly website, which is a free supplemental resource for those who attend the school. Follow the instructions below to register for O'Reilly.

1. Click here: `http://byui.idm.oclc.org/login?url=https://www.safaribooksonline.com/library/view/temporary-access/`

2. Enter your BYU-Idaho email address on the Welcome! screen to access the database. You will receive an email to set up an individual account - this is optional. You will be able to access content without an account. (See 5a before deciding if you want an account or not.)

3. Click this link to get to the book: `https://learning.oreilly.com/videos/beginning-python/9781786468994`

4. When you need to get back to the book, you can click on the link in step 3 again (or bookmark it). It should recognize that you already have done steps 1 and 2.

If for some reason it doesn't let you have access again, click this link and it will let you in. (It's steps 1 and 2 combined, but you can only use it after doing steps 1 and 2 the first time.) `https://learning.oreilly.com/accounts/login/?next=/library/view/temporary-access/`

This week's video assignment: Chapters 1 and 2 in the video text book (the link in step 3)

1 Introduction

So, you've been asked to learn Python. You may feel overwhelmed, frightened, or ready to quit before you've even started. You are 100 percent correct to feel that way.

Learning a new programming language is tough, especially if you've never learned one before, but it's not impossible. Fortunately, Python is intuitive. By the end of the semester, when you are asked to code a formula, you may spit the code out without even realizing it.

This step-by-step guide offers the necessary knowledge to get through the semester and prepare for a future career. The task may seem daunting at first, but with a little hard work and dedication, you will be coding like a master in no time.

Python is a powerful programming language that is simple to use. Many tedious job responsibilities can be fixed through programming. Say you work for an engineering company and the manager asks you to research heat temperatures in different cities. Instead of manually inputting data into an excel sheet from hundreds of websites, you can program Python to do it for you. Within an hour you can have results on your manager's desk. You can mine Twitter data, track BitCoin prices, and create your own calculator. You can create graphs, design websites, solve complex math problems, and even program artificial intelligence. The possibilities are literally endless. Overall, Python will reduce human error in your work, save time, and offer a peace of mind.

Don't believe me? Guess you'll just have to try it yourself.

2 Helpful Tips

2.1 Visualization

If you don't understand a piece of code, or if you're unsure why the code is not working, follow the link below, paste the offending code, and press "Visualize Execution." The program will take you line-by-line to see how the code functions.

`http://www.pythontutor.com/visualize.html#mode=edit`

2.2 Google Everything

Throughout this guide, you will find website links and tutorials to help you code. They may not be required for every homework assignment, however these tools will support your work throughout the semester.

I encourage you to not only follow those links, but look up other questions that you have on the internet. Programming is 5 percent previous knowledge, 15 percent crying, and 80 percent googling. Almost all of your questions can be answered with this guide and Google.

Googling is an underestimated skill in the workplace. Many people lack the fundamental skill of searching for information. In an era when research is gold, the ability to use a search engine effectively is a useful skill to possess. Master this skill, and your work will improve.

3 Getting Started

3.1 Installation

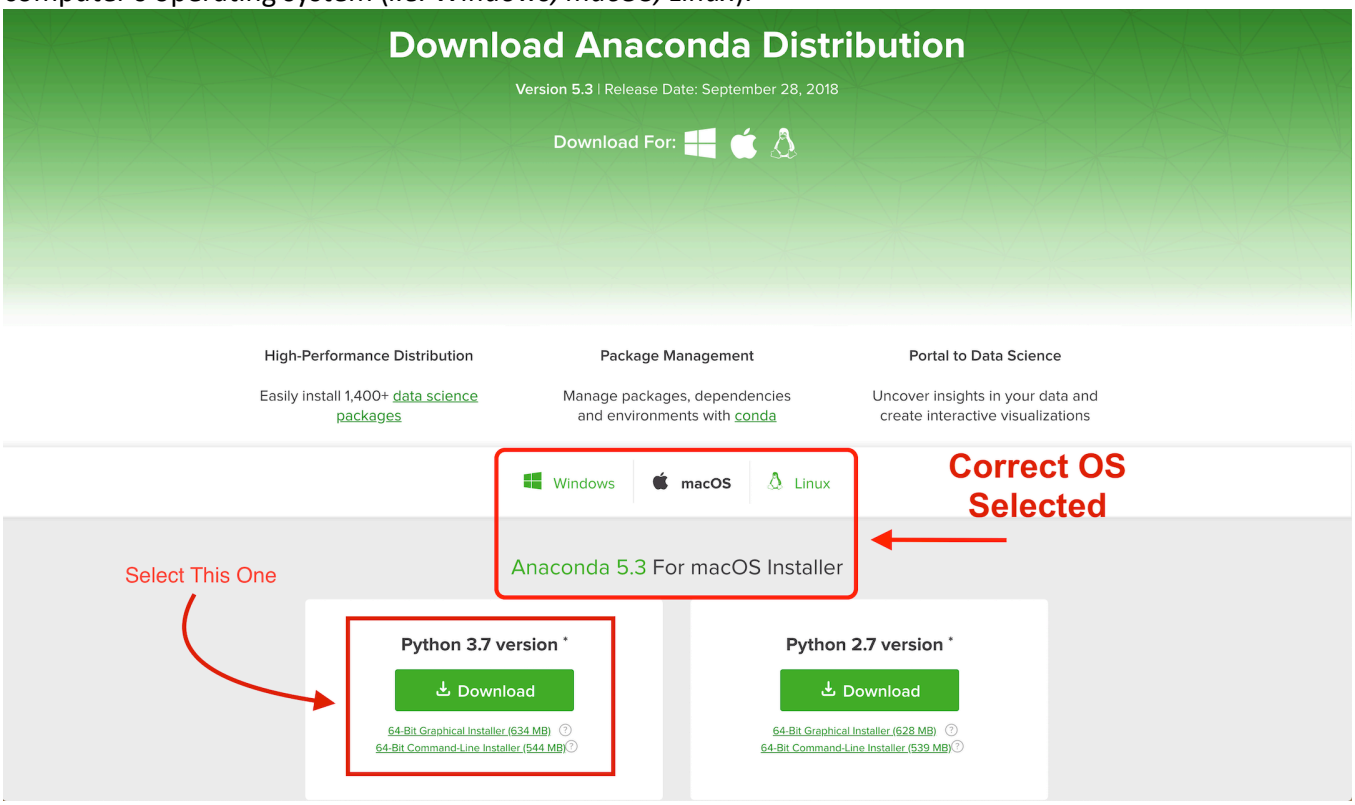
First, follow the installation guide in the pages below.

Python Installation Guide
ECON 381 – Intermediate Macroeconomics
Winter 2019

Python Installation

1. Copy & Paste this link into your browser: https://www.anaconda.com/download.

2. Click Download under '*Python 3.7 version*'. Ensure you select the appropriate download for your computer's operating system (*i.e. Windows, macOS, Linux*).

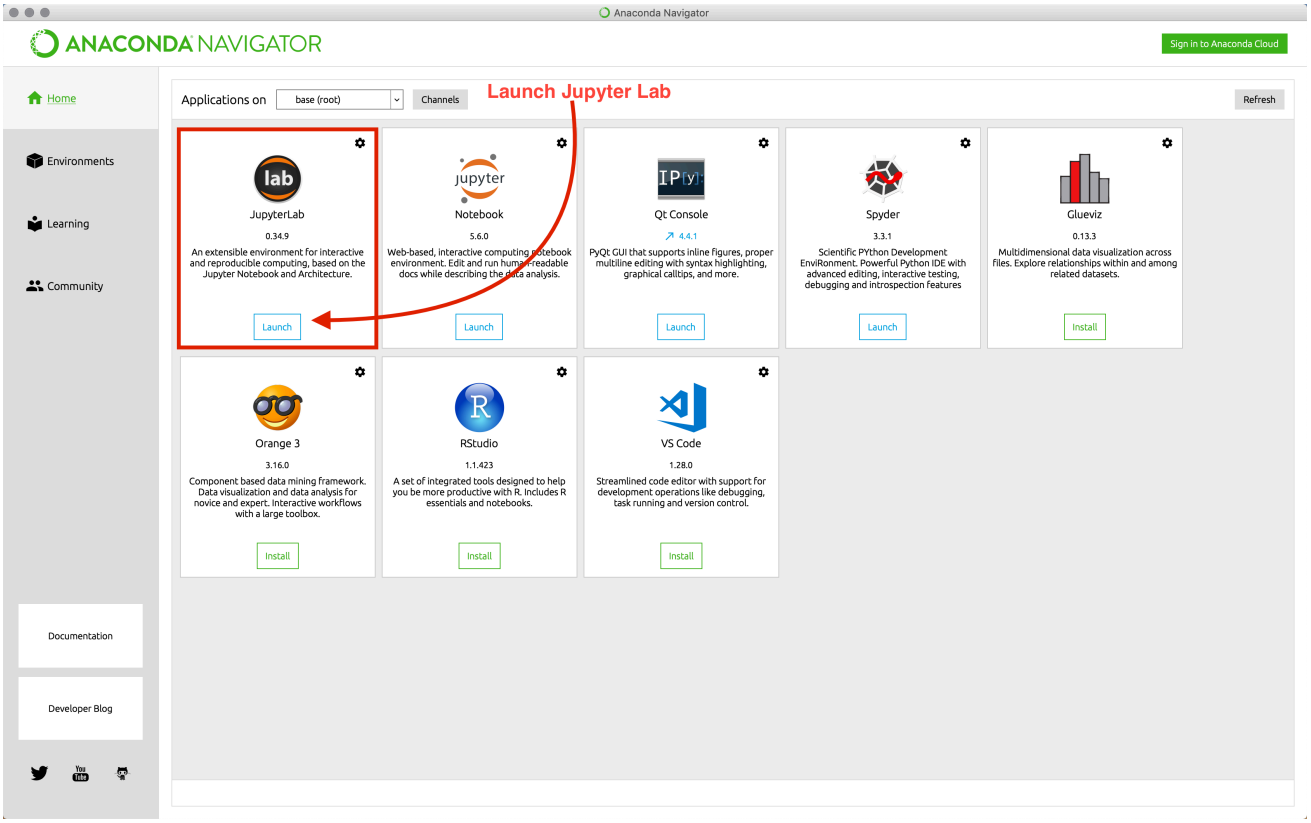


3. After downloading, an install helper will open on your computer. Complete the installation without changing any default settings.

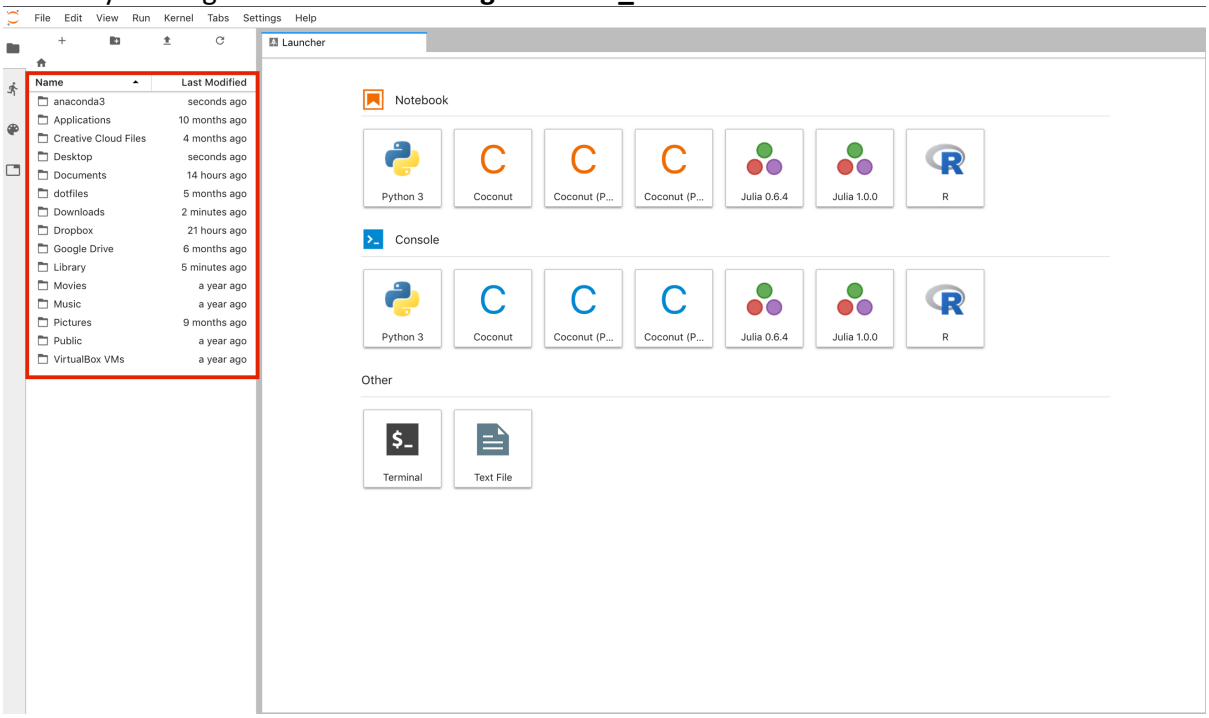
4. After the installation is complete, find and open up the newly installed application called, "**Anaconda Navigator**" this will be the application you will use to interface with Python and Jupyter Lab Notebooks for the semester.

Anaconda Navigator & Jupyter Lab

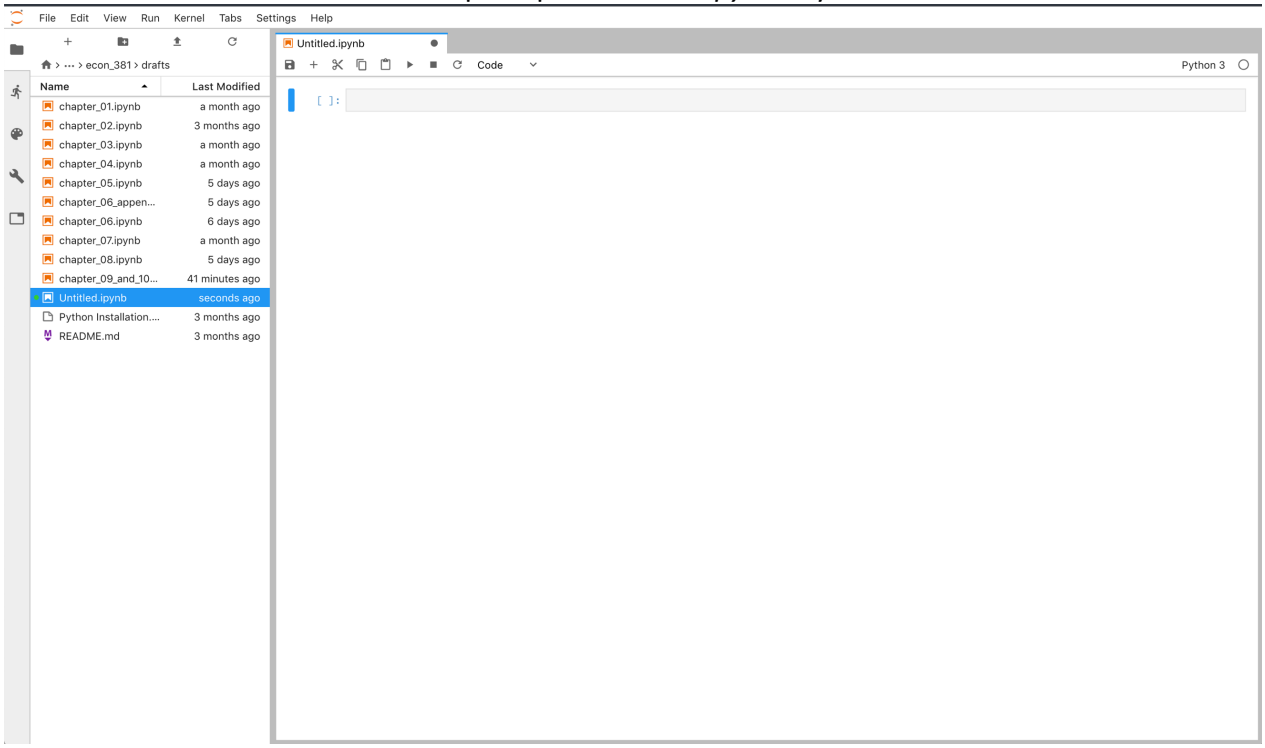
1. Upon opening Anaconda Navigator, it will take you to a home screen which will display several apps you can use to interface with Python and other programming languages. For this class, we will be using **Jupyter Lab** (notice the distinction from Jupyter Notebook). Double click on Jupyter Lab to launch a new session.



2. Now use the file explorer on the sidebar to navigate to your ‘econ_381’ folder on your computer. Where this folder is created and stored will be different for everyone. For example, I can access my folder by clicking **Documents >> College >> econ_381**

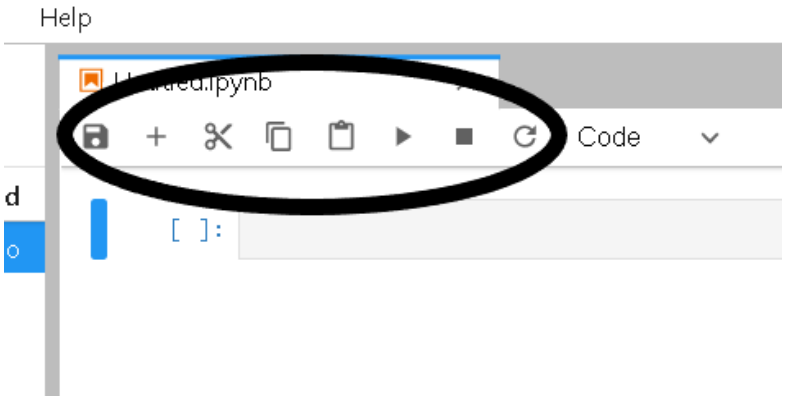


3. Once you have navigated to the proper folder, create a new Python Notebook by selecting **Python 3** under the **Launcher Tab**. This should open up an *Untitled.ipynb* in your browser.



3.2 The Interface

Now that you have JupyterLab open, it's time to play around with it. First, hover over and click each of these buttons as shown. What do they do?



Now, click the first cell. A cell is a gray box where you type in the Python code. Right now, you should only have one or two cells in your file. Click on the first one, but don't type anything yet.

Click the drop-down button that says "code," and change it to markdown.

You'll notice the cell you had clicked on before no longer has square brackets next to it. This means that you can now type in plain text or LaTeX code. To understand this concept, just type "Hello World" into the markdown cell and press control + enter. What do you notice?

3.3 Print Function

Before we go any further, it is important to know what the print function is.

Learning to code is similar to a theater performance. You are the stage manager responsible for the total production and ambiance of the show. You make sure the lighting, props, sound, and stage setup is perfect. The input screen is the code you see behind-the-scenes, whereas the output screen is what the audience notices on stage. They don't see the hustle and bustle, but an organized and entertaining production.

The *print()* function allows you to display your answers on the output screen. Try typing this in JupyterLab:

```
1 print('Hello World')
2
3 Out: Hello World
```

3.4 Commenting

As you code, you will want to leave notes for yourself. Sometimes, you might want to leave a note that explains what a piece of code does. Other times, you might want to take note of a piece of code that you may want to improve later. In python, you can do this directly on the interface using the pound symbol (#). When you put this symbol in front of some words in python, you are telling python to skip over those words because they are only for your reference. Throughout the guide, I will be putting comments on the code that I write to clarify how the code works. See the code below for a few examples of a comment.

```
#This is a comment
#This will not show on the output screen
print('Python reads this') #Python skips this

#Python reads the print function and
#the words inside it, but not the ones
# after the pound symbol

Out: Python reads this
```

3.5 Variables

Variables are like a storage container. They label and store information in the data memory which can used throughout the program. Variables prevent you from retyping equations, numbers, or words that you've already inputted, which saves you time and energy. You can name your inputs almost anything you want, but try your best to name a variable with an accurate description. Note that variables are case sensitive.

To create a variable, type the name you want to give your variable, an equals sign, and the value of the variable. To access a variable, simply type in its name.

```
#Creating variables called "Length" and "Width"
Length= 20
Width= 30
#Accessing the variables
Length
#using the width variable inside the print function
print(Width)

```

```
Out: 20
Out: 30
```

Since "Length" is equal to 20, when you type the variable "Length," Python prints out 20. The same is true for "Width."

You can program operations using variables. We'll go over how to do math in Python later, but for now, just focus on how Python reads the equations that are given to it. How does Python use the following equations to get an output?

```
Length = 20
Width = 30
(Length + Width) * 2 #creating an equation using
    variables

Out: 100

#You can also store an equation in a variable.
Area= Length * Width #create equation within the
    variable "Area"
print(Area)

Out: 600
```

Let's say after doing all that math, you realize "Length" was supposed to be 25, not 20. If you created an equation without variables, you would have to manually change the values from 20 to 25:

```
(20 + 30) * 2
Area= 20 * 30
```

This doesn't seem like too much of a hassle since it's such a small equation; but imagine it became more complex. Instead of one equation, you now have seven. Each equation has the number 20 that needs to change to 25. Instead of changing the equation seven different times, you only have to change it once using a variable. Python updates each equation automatically.

```
Length = 25
Width = 30
(Length + Width) * 2

Out: 110

```

```
7  Area= Length * Width
8  print(Area)
9
10 Out: 750
```

3.6 Types of inputs

What is the difference between A and 1? It's an obvious answer—A is simply a letter and 1 is a number. Just as you have labels for letters and numbers, Python has labels for the inputs it receives. These labels are called data types.

A data type is nothing but a categorization of data. In Python, like in all programming languages, data types are used to classify one particular type of data. This is important because the specific data type you use will determine what values you can assign to it and what you can do to it (including what operations you can perform on it). For example, in the real world, you can do this:

$$8 + \pi$$

Even though π is a symbol, and it stands for an irrational number, we are still able to interpret the expression and complete the equation. However, how would you interpret this?

$$sky + 11$$

Just like you wouldn't know what to do with an equation like that, python has difficulties combining certain data types. You need to know what each data type is, how to use it, and when it is and isn't OK to combine them in order to be a successful programmer. There are several types of data in Python, but the only ones you will need to use this semester are integer (int), float, string (str), and boolean (bool).

Integers:
Integers are whole numbers such as 3, 8, 10, or 70.

Floats
Floats are numbers that have decimals in them, such as 4.2, 78.1, and 8.976.

Strings:
Strings are words. When typing words in Python, you must wrap them in single or double quotes. For Example:

```
print('This is a string')
#Or
print("This is a string")
```

Booleans

Booleans are essentially true or false indicators. You can create your own booleans, or you can ask Python to state whether items fit certain conditions.

```
#creating your own booleans and testing them
happy = True
knowit = True
print('happy and knowit      :', happy and knowit)
print('happy or knowit       :', happy or knowit)
print('not happy             :', not happy)
print('not(happy and knowit):', not (happy and knowit))
print('not happy and knowit :', not happy and knowit)
print('not happy or knowit  :', not happy or knowit)
print('not (happy or knowit):', not (happy or knowit))

Out: happy and knowit        : True
Out: happy or knowit         : True
Out: not happy               : False
Out: not (happy and knowit)  : False
Out: not happy and knowit    : False
Out: not happy or knowit     : True
Our: not (happy or knowit)   : False


# Comparing floats or integers
gpa = 3.45

# is gpa equal, greater than, or less than 4.0?
print('gpa == 4.0 ->', gpa == 4.0)
print('gpa > 3.0  ->', gpa > 4.0)
print('gpa < 3.0  ->', gpa < 4.0)

Out: gpa == 4.0 -> False
Out: gpa > 3.0  -> False
Out: gpa < 3.0  -> True
```

Note the double equal signs "==" are used when asking if GPA is equal

to 4.0. You must always use double equal signs when comparing floats and integers.

A Note on Data Types

Python automatically assigns a data type to every input that it receives. When you enter data into an equation, you have to make sure all of the data types are the same. You cannot combine strings and floats, or integers and strings. The only exceptions to this rule are integers and floats. Essentially, make sure that you only place numbers with numbers and words with words.

Sometimes, Python misreads what data type a certain input is. For example, you may have a number, but python reads it as a string. When writing code, you may get an error that looks like this:

```
a = 10 #storing the value 10 in the variable a
b = 20 #storing the value 20 in the variable b

a * b  #multiplying 10 * 20

Out: 20202020202020202020 #repeats 20 10 times instead
   of multiplying
```

If your output looks blatantly wrong like this one does, then you might want to check what kind of data type each variable is.

```
print(type(a))
print(type(b))

Out: int
Out: str
```

Because the variable b was a string, python interpreted $a * b$ as repeating b 10 times. To convert b into an integer, simply do this:

```
b = int(b) #creating a new b variable that replaces the
    old b
print(b)
print(type(b))

Out: 20
Out: int
```

To convert b to a float, do this:

```
b = float(b)
print(b)
print(type(b))

Out: 20.0
Out: float
```

You can also convert b to a string. To do this, follow the code below:

```
b = str(b)
print(b)
print(type(b))

Out: "20"
Out: str
```

If you want to learn more about data type conversion, go to this website: `https://www.geeksforgeeks.org/type-conversion-python/`

To learn other data types in Python, go to this website: `https://realpython.com/python-data-types/`

3.7 Basic Commands

3.7.1 Simple Math

You can do simple math using Python such as addition, subtraction, multiplication, division, square roots, and raising to a power. When using these operations, always leave a space between the number and the symbol for easy reading.

Addition:

```
2 + 3

Out: 5
```

Subtraction:

```
5 - 3

Out: 2
```

Multiplication:

```
2 * 3

Out: 6
```

Division:

```
6 / 3

Out: 2
```

Square Roots:

```
import numpy as np #see section 5.1 for importing
np.sqrt(9)

Out: 3
```

Raising to a Power:

```
3**2 #Do not use a caret symbol

#Or
pow(3, 2)

Out: 9
```

Absolute Value:

```
value = -123.456
print(value)
print(abs(value))

Out: -123.456
Out: 123.456
```

Range:

While there is no function to find the range of a list or array (more on those later), you can combine the minimum and maximum functions to create your own range function.

```
my_list = [1, 2, 3, 4, 5]
print(max(my_list) - min(my_list))

Out: 4
```

3.7.2 Formatting

You can use the *.format()* command to format the outputs of your *print()* inputs. Note that when you use the *.format()* command, all numbers are automatically converted to strings.

The main formatting type you will use is the delimiter. The delimiter essentially rounds your numbers in an equation. You can use the delimiter input in the *.format()* command by adding a separator (comma) and the delimiter input between the placeholder brackets: *:.2f*. This command limits your output to two decimal figures (hence 2f after the period). You can limit it to as many decimal places simply by changing the number.

Important: Do not put a space before the colon in the delimiter command. It will give you an error.

```
number= 28.9374
print('{:.2f}'.format(number))
print('{:.3f}'.format(number))

Out: 28.93
Out: 28.937
```

You will also use the *.format()* command to format numbers to appear as percentages. Just like the delimiter command, you can (and must) specify the number of decimal places. Instead of an f, place a percent sign after you specify the number of decimals.

```
number= .25
print('{:.2%}'.format(number))
print('{:.3%}'.format(number))

Out: 25.00%
Out: 25.000%
```

You can also use the *.format()* command to use a comma to separate thousands. Just put a comma after the colon, and Python will enter the commas where they are supposed to go.

```
number= 2500
print('{:,}'.format(number))

Out: 2,500
```

Lastly, you can combine the delimiter with the comma separator and percentage formatter.

```
number1= 2500.37382
print('{:,.2f}'.format(number1))

number2 = 25.0037382
print('{:,.2%}'.format(number2))

Out: 2,500.37
Out:2,500.37%
```