MSDS 7330
File Organization and Database Management
Homework Introduction to Python

Due Week 3

Name: Zach Brown

What to Submit

Your final submission shall be a single pdf document that includes this document, screen captures of your exercises plus your answers to each of the written exercises. Note that you are expected to clearly label each section so as to make it clear to the instructor what files and data belong to which exercise/question.

Collaboration is expected and encouraged; however, each student must hand in their own homework assignment. To the greatest extent possible, answers should not be copied but, instead, should be written in your own words. Copying answers from anywhere is plagiarism, this includes copying text directly from the textbook. Do not copy answers. Always use your own words. For each question list all persons with whom you collaborated and list all resources used in arriving at your answer. Resources include but are not limited to the textbook used for this course, papers read on the topic, and Google search results. Note that 'Google' is not a resource. Don't forget to place your name on the document.

Exercise 1 : Installing Python

Install the Anaconda distribution of Python.

The Anaconda distribution may be downloaded from http://continuum.io. Follow the directions provided by continuum for your specific machine.

After installing Anaconda, at a command line prompt type condo install seaborn to install a number of useful packages.

Be sure to capture all of this to include in your submission.

Exercise 2 : Hello, world!

A program is just a set of instructions that the computer will execute. One of the most fundamental instructions is an instruction to print output to the screen. This is fundamental because without any way to observe the results of the instructions, it is not possible to reap the benefits of executing the instructions. Therefore, the most basic of commands is:

print x

which prints the value of the expression x followed by a new line.

Create a new program file called hello_world.py. Within this file you will write your very first "Hello, world!" program. You may use any text editor to create the program file; however, it is simple to use Spyder from the Anaconda distribution if you downloaded the graphical editors.

When you create your program file, be sure to save the file as hello_world.py. Do NOT skip the '.py' extension. Furthermore, every program that you create will begin with a bank of comments, with a comment line for your name, the course number and your section number, the file name and today's date. Recall that a comment line begins with a '#' (pound) symbol.

You are now ready to create your very own "Hello, world!" program in Python. "Hello, world!" is the first program that most programmers write in a new programming language. In Python, "Hello, world!" is very simple. It should be only one line.

Write your "Hello, world!" program now. Save your program and run it. Your program should look something like:

Daniel Engels <--- Your name # MSDS 7330 401
<--- Your section

hello_world.py <--- File name # 31 Aug 2015
<--- Date

prints "Hello, world!" to the screen <--- Proper comment print "Hello, world!"

And, your output on the iPython console (when used within Spyder) should look something like:

In [1]: runfile('/Users/dwe/hello_world.py', wdir='/Users/dwe') Hello, world!

Notice that the input to the Python command line (the command line begins In [1]:) is the Python command runfile with a full path to the file and a declaration of the working directory. The declared file is executed and the output, Hello, world!, is printed to the iPython console.

Be sure to submit your file and your output screen capture.

Exercise 3 : Tic-Tac-Toe

Now that you have said Hello, it's time to get better acquainted with your programming environment. The purpose of this exercise is to make sure you understand how to write programs using your computing environment. Many students struggle with Python because of issues with the programming environment, not because they have trouble with the material.

Write a program that, when run, prints out a tic-tac-toe board. Save this program in a file titled tic_tac_toe.py. Your output should look something like:

```
     |     |
-------------
     |     |
-------------
     |     |
```

Be sure to submit your file and your output screen capture.

Exercise 4 : Tic-Tac-Toe Step-by-Step

Now that we have a game board, it's time to print out the step-by-step state of that board during a particular game. That is, start by printing out the blank tic-tac-toe board. Then, print out the board after the first player, 'X', places their first 'X'. This board will have a single 'X' somewhere on it. Then, print out the board after the second player, 'O', places their first 'O'. This board will have a single 'X' and a single 'O' placed on it. Then repeat for all steps until the game is complete.

One approach to printing out the board is simply by hand designing and printing each new state of the board. You might notice that there is a lot of repetitive printing and typing. One approach to reduce the amount of typing is to use variables. Recall that a variable is a container for storing information. For example, the following program text:

a = "Hello, world!" print a

has the following output:

Hello, world!

The equal sign '=' is an assignment operator that tells the Python interpreter to assign the value "Hello, world!" to variable a.

Variables may have their stored value changed by simply reassigning them a value. For example, the following program text:

a = "Hello, world!" a = "And,
goodbye." print a

has the following output:

And, goodbye.

In this program text, after executing the first assignment operation, the variable a has the value "Hello, world!". But, after executing the second assignment operation, the variable a has the value "And, goodbye.". Since the value of a is printed only

after the second assignment operation is executed, the value of a is "And, goodbye.", which is the value printed to the screen. If you want to save the values of both strings, you should change the name of the second variable to something other a, such as b.

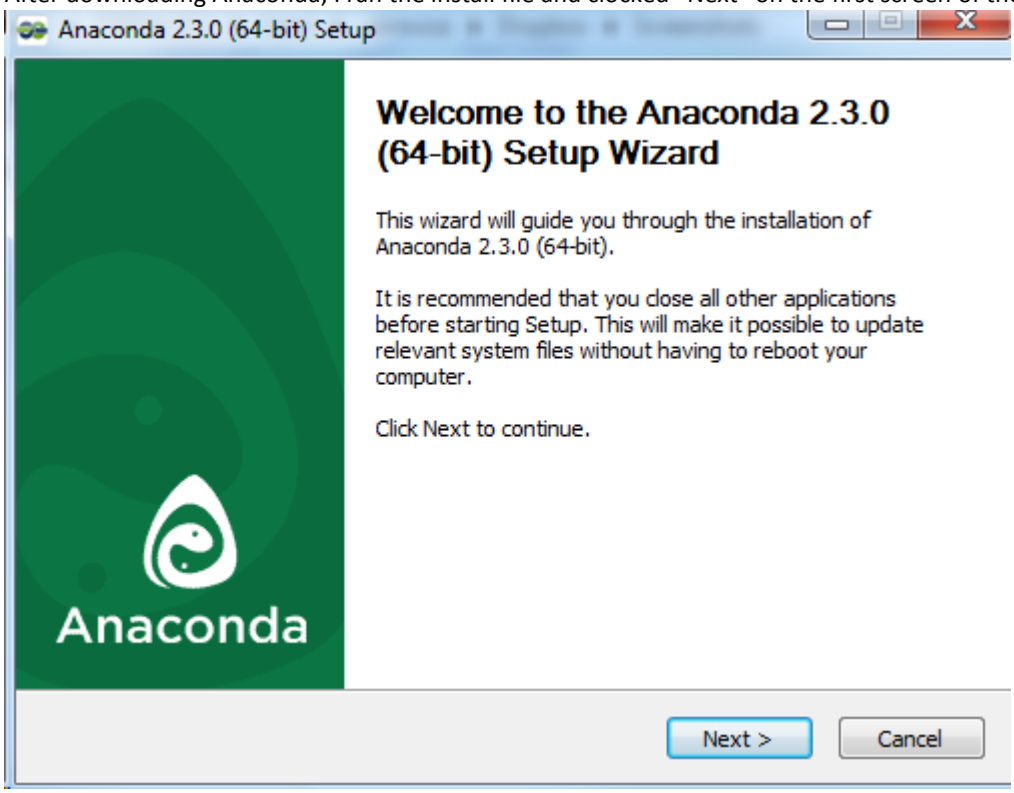
The input must be handled elegantly since the players cannot write their X's and O's on the computer screen and have that information captured (for a non-touch sensitive screen, anyway). It is also advisable to place a key as a reminder to the players on how they should enter their chosen square when it's their turn.

Write a program that prints out the tic-tac-toe board and the state of that board after every step of the game using variables to minimize the amount of code that is actually written.

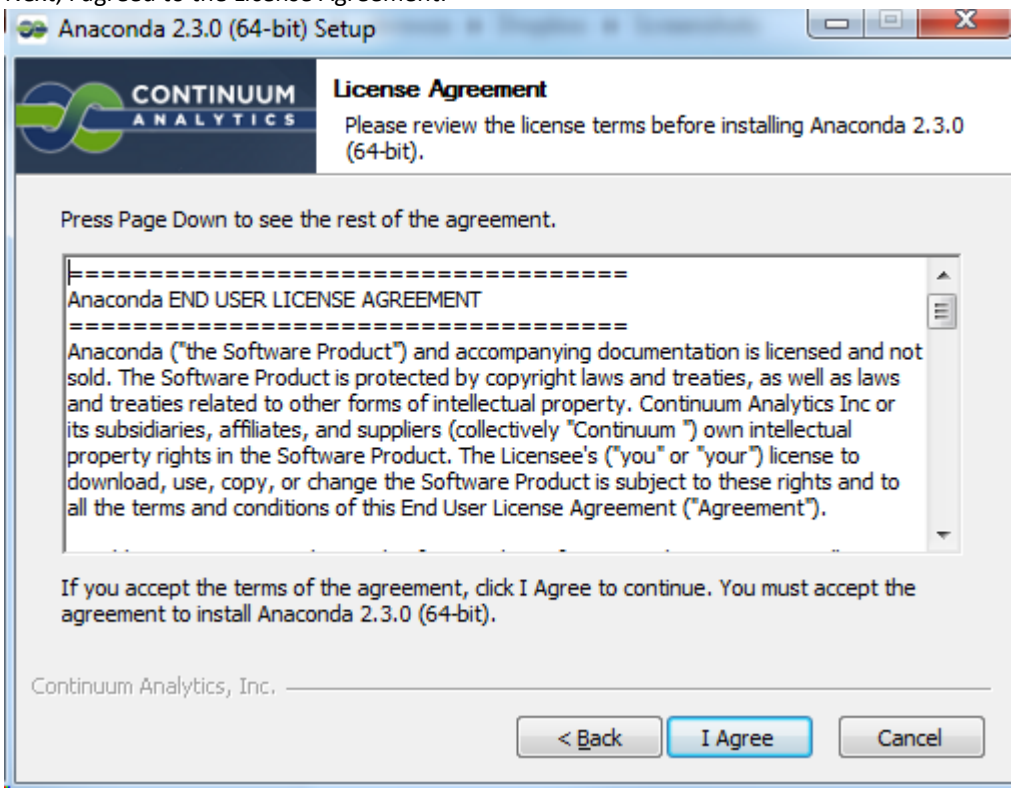
Be sure to capture the entire game, including prompts for the players and their inputs.

Exercise 1

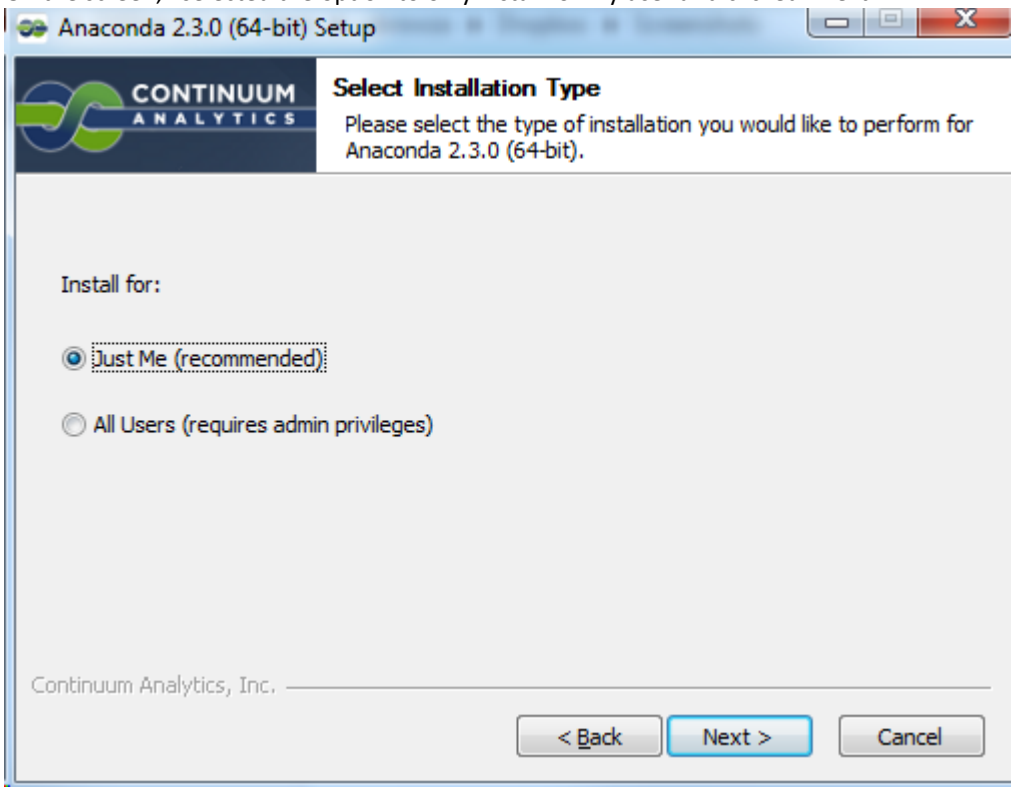
After downloading Anaconda, I ran the install file and clocked “Next” on the first screen of the Setup Wizard:



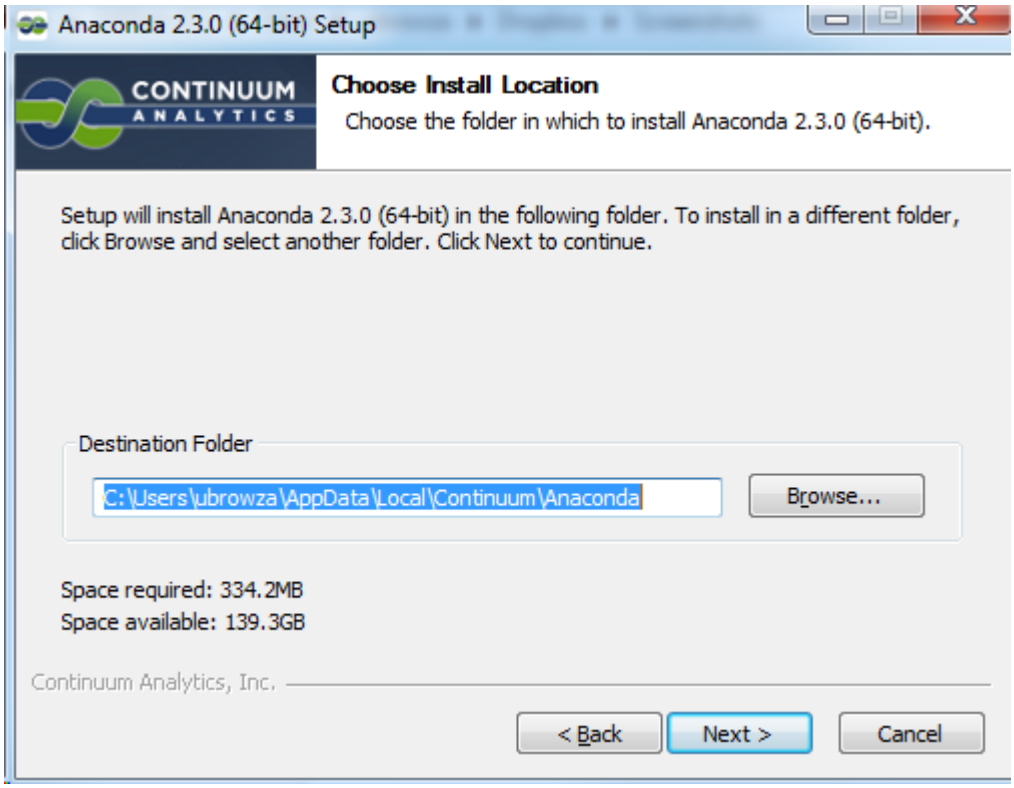
Next, I agreed to the License Agreement:



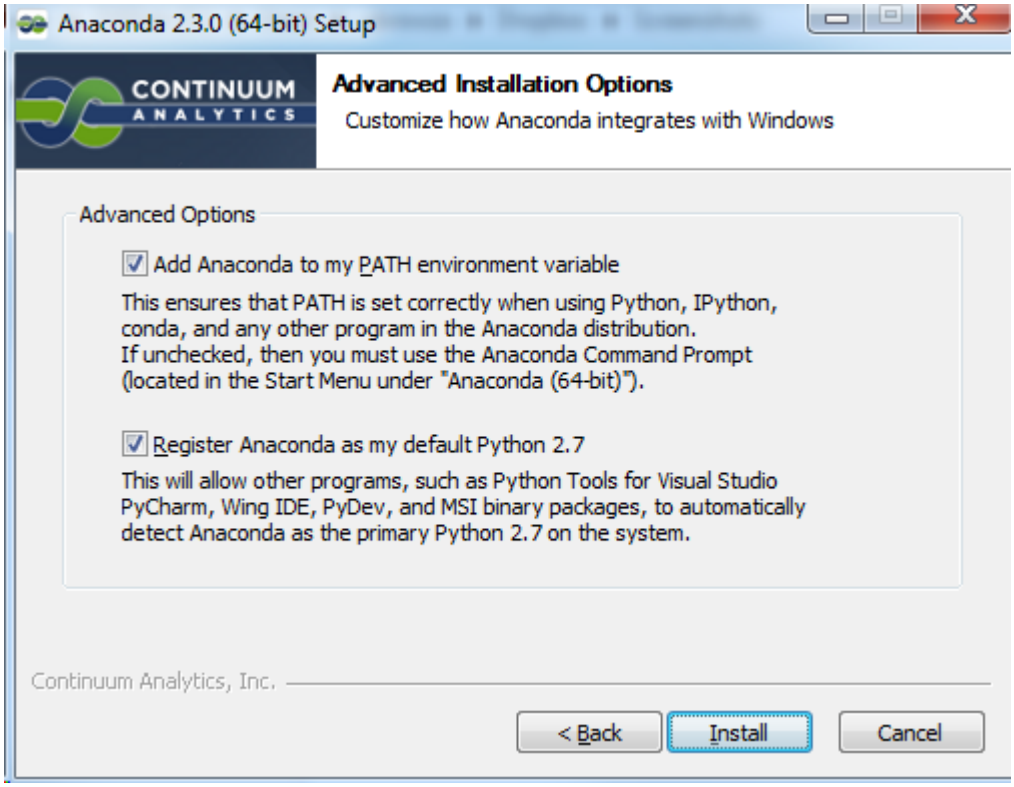
On the screen, I selected the option to only install for my user and clicked “Next”:



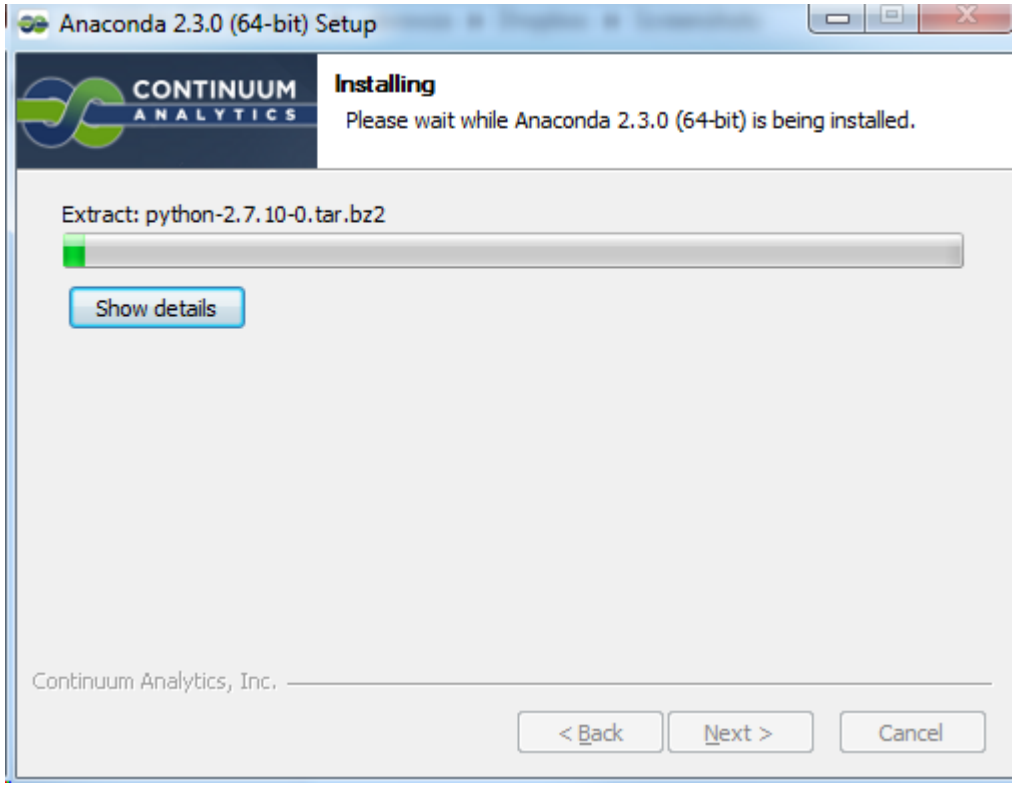
I then chose to install in the default destination folder and clicked “Next”:



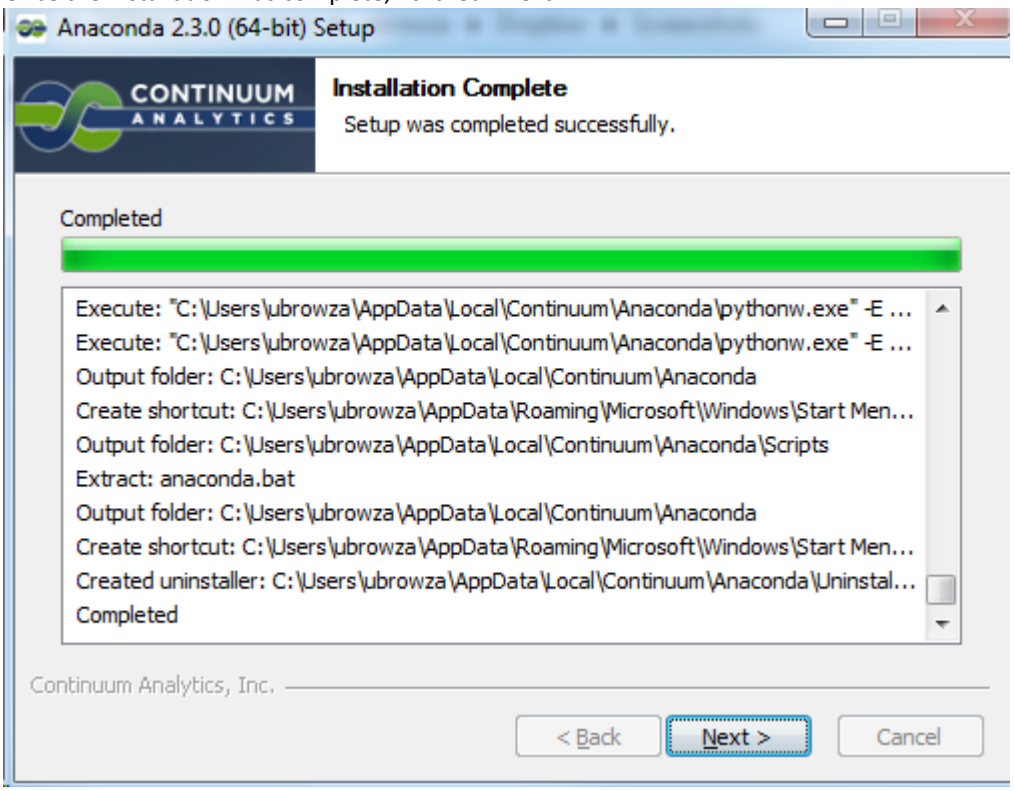
On the next screen, I left both boxes checked and clicked “Install”:



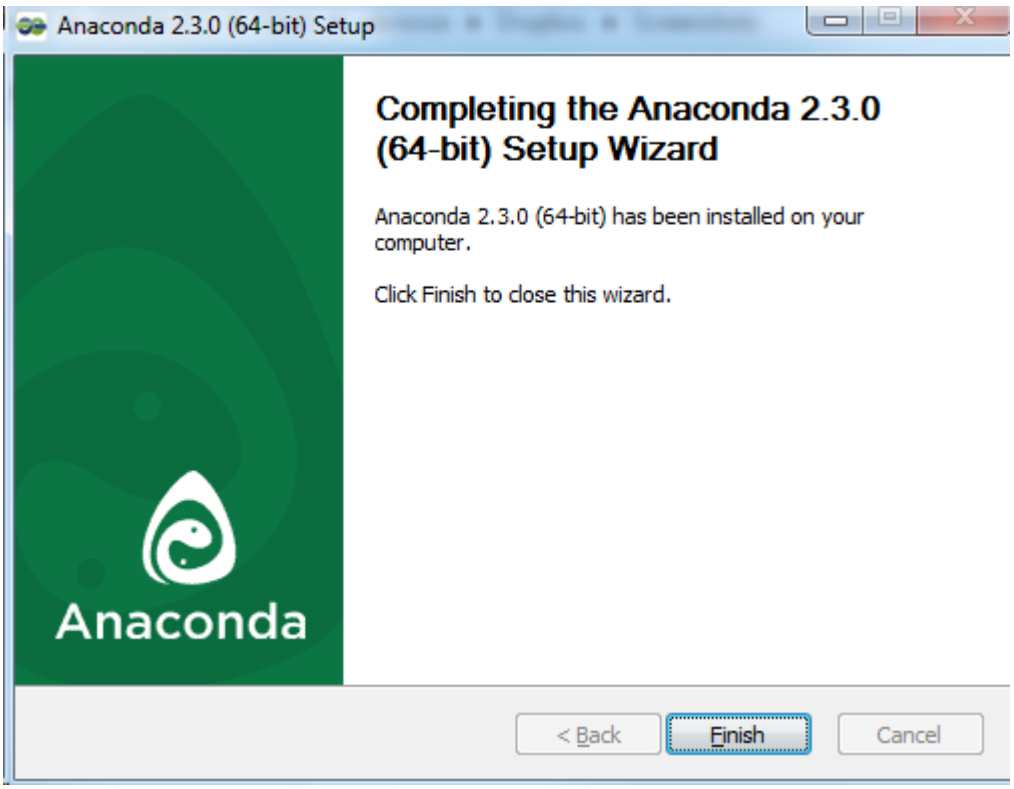
The installer then takes several minutes to run:



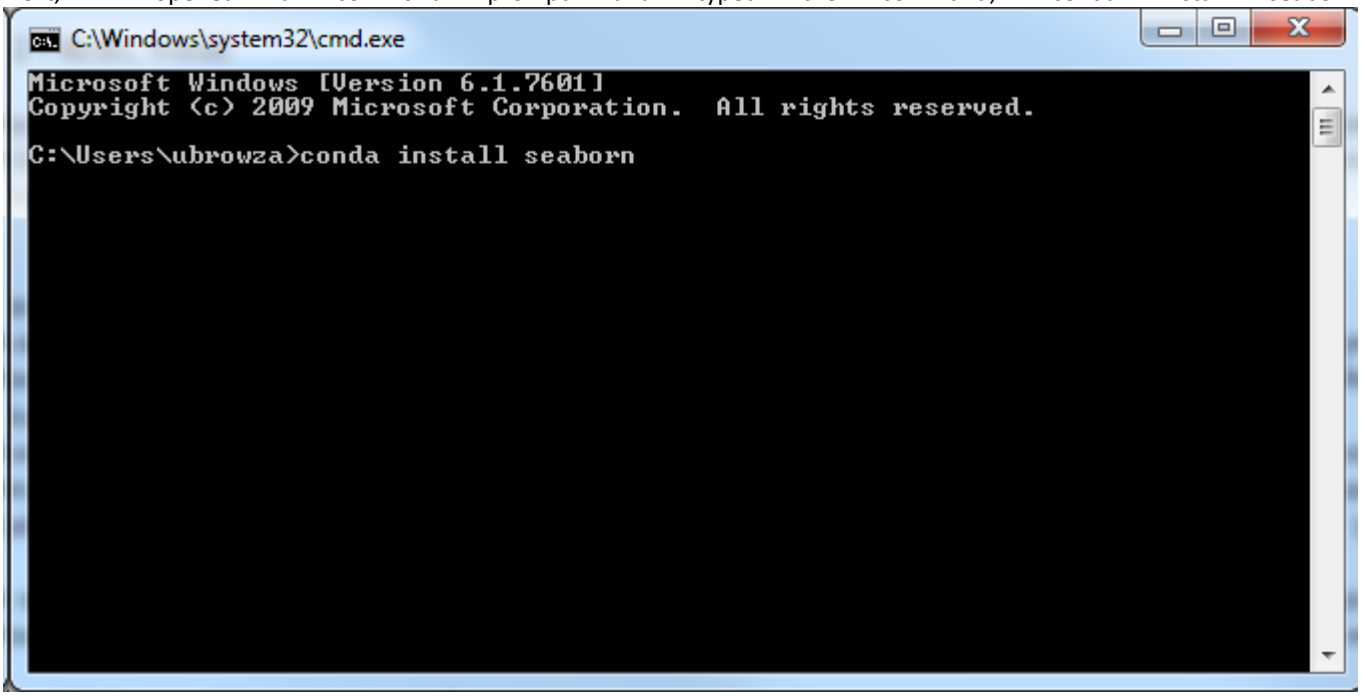
Once the installation was complete, I clicked “Next”:



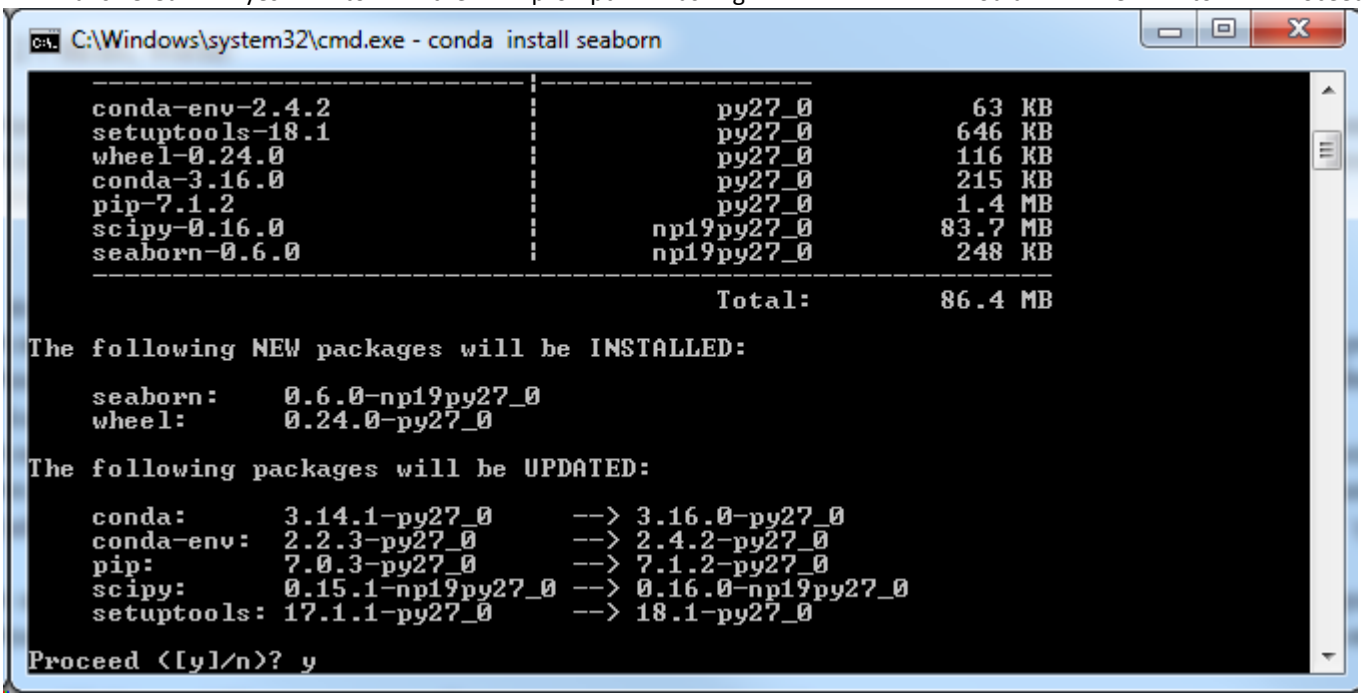
I then clicked “Finish” to close the wizard:



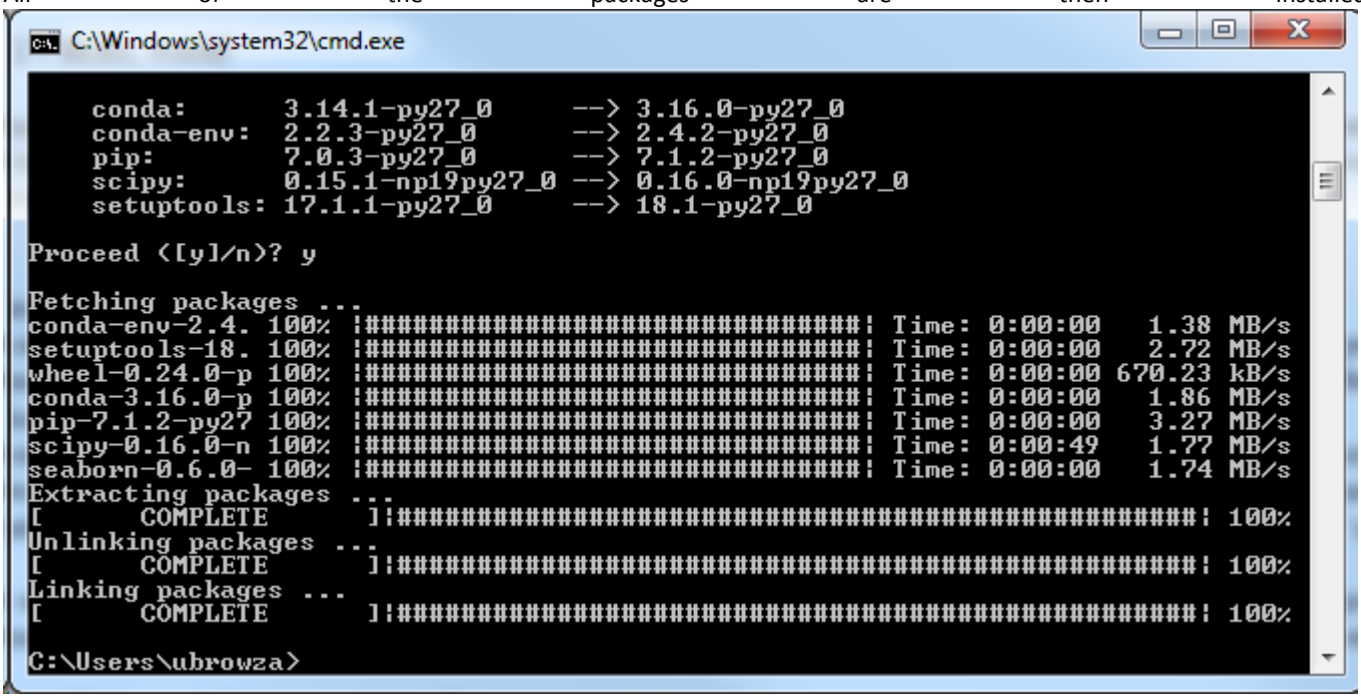
Next, I opened a command prompt and typed the command, “conda install seaborn”:



I answered yes to the prompt asking if I would like to Proceed:



All of the packages are then installed:



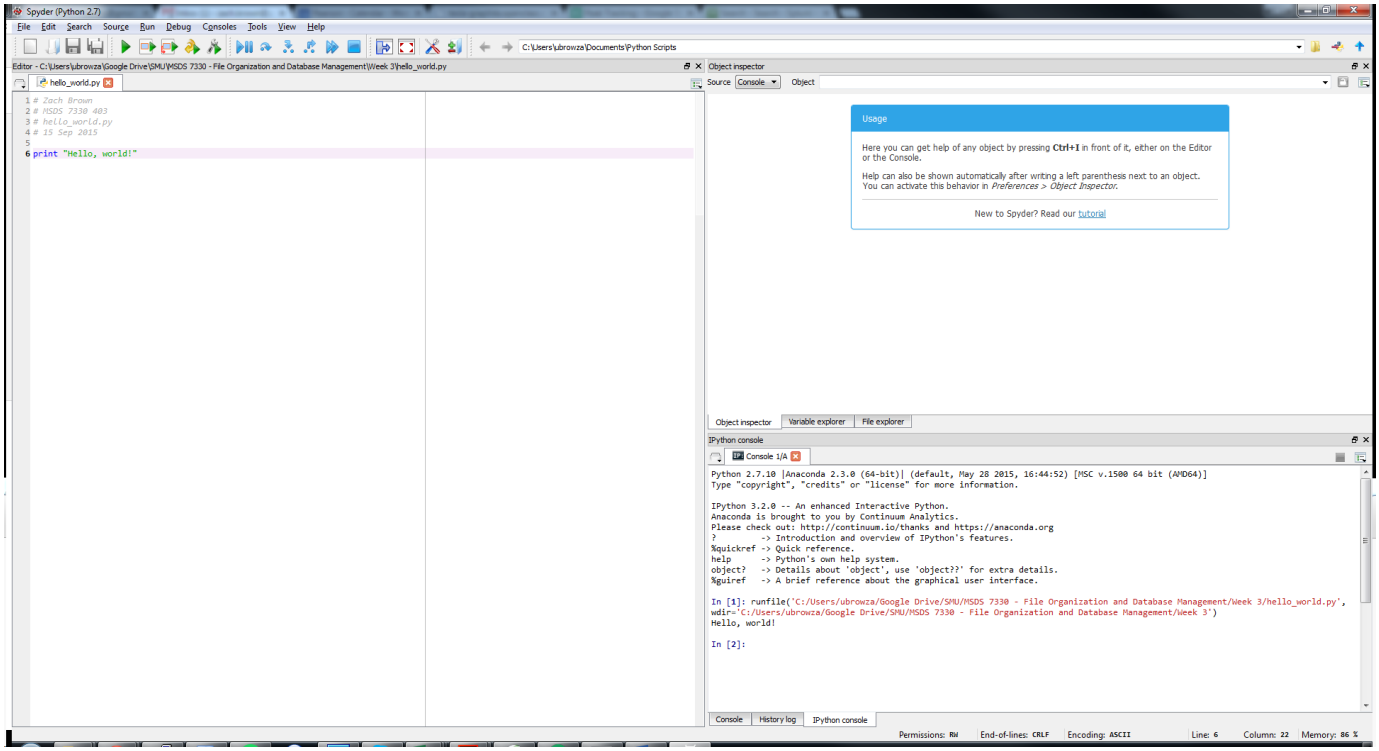
Exercise 2

The file named “hello_world.py” contains the following code:

```
# Zach Brown
# MSDS 7330 403
# hello_world.py
# 15 Sep 2015

print "Hello, world!"
```

This is a screenshot of the output of the script:



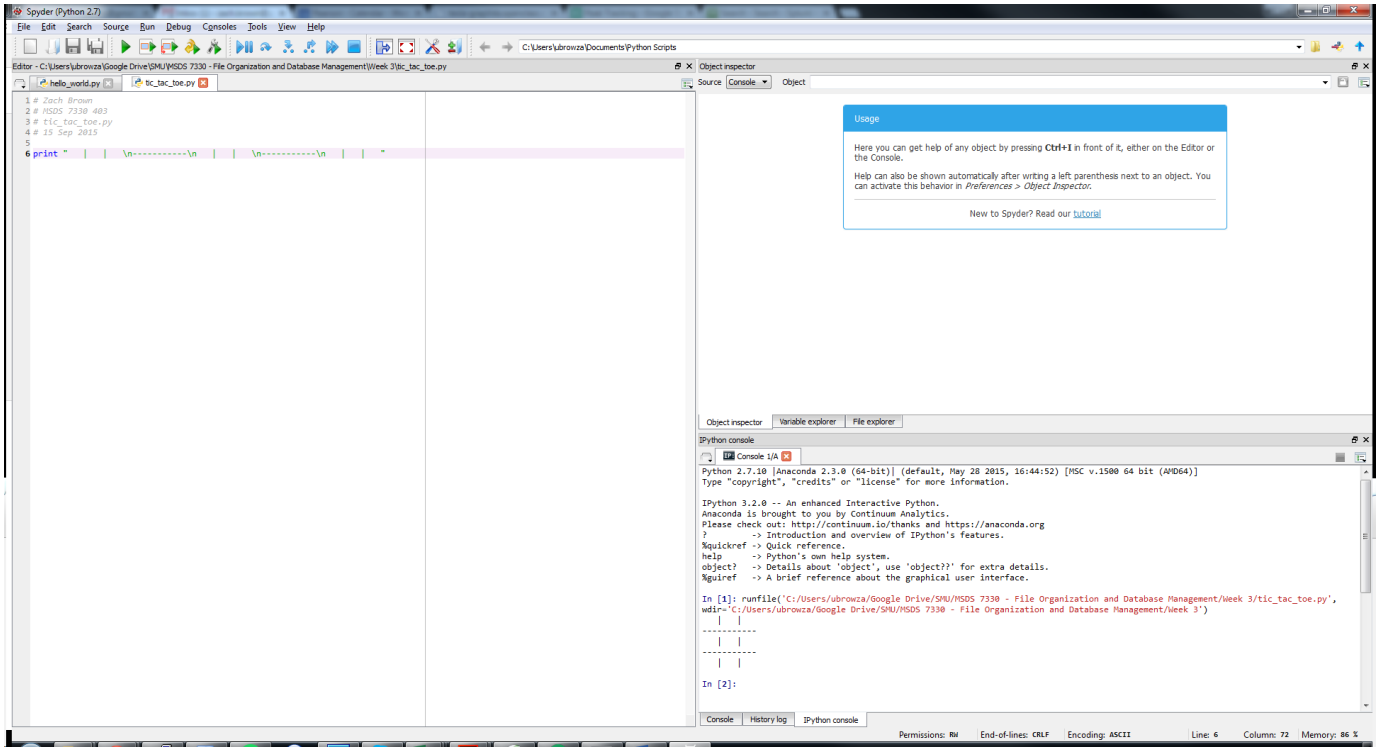
Exercise 3

The file named “tic_tac_toe.py” prints a blank Tic Tac Toe board when run. The file contains the following code:

```
# Zach Brown
# MSDS 7330 403
# tic_tac_toe.py
# 15 Sep 2015

print "   |   |   \n-----------\n   |   |   \n-----------\n   |   |   "
```

This is what the output looks like:



Exercise 4

The file named "tic_tac_toe_step_by_step.py" runs a two player interactive Tic Tac Toe game which prompts the user to choose which square to mark on each turn and includes a sample board with space numbers each time it prompts the user. After every turn, the updated game board is printed to the screen. When the game is over, a "GAME OVER" message is printed to the screen also stating whether X, O or nobody won. This script also includes some rudimentary error checking to make sure that the user enters a valid move to an unoccupied square. The file contains the following code:

```
# Zach Brown
# MSDS 7330 403
# tic_tac_toe_step_by_step.py
# 15 Sep 2015

# Initialize variables for each space on the board
spaces = [" "] * 9

# Initialize variable containing string represting numbered board spaces
guideBoard = " 0 | 1 | 2 \n-----------\n 3 | 4 | 5 \n-----------\n 6 | 7 | 8 "

# Initialize turn counter
turn = 0

# Define function to print the board in its current state
def printBoard():
    print "Current board:"
    print " " + spaces[0] + " | " + spaces[1] + " | " + spaces[2]
    print "-----------"
    print " " + spaces[3] + " | " + spaces[4] + " | " + spaces[5]
    print "-----------"
    print " " + spaces[6] + " | " + spaces[7] + " | " + spaces[8]

# Define function to check if game is over or not
def gameOver():
    if (spaces[0] != " "):
        if ((spaces[0] == spaces[1] == spaces[2]) | (spaces[0] == spaces[4] == spaces[8]) | (spaces[0] == spaces[3] == spaces[6])):
            return spaces[0]
    if (spaces[1] != " "):
        if (spaces[1] == spaces[4] == spaces[7]):
            return spaces[1]
    if (spaces[2] != " "):
        if ((spaces[2] == spaces[4] == spaces[6]) | (spaces[2] == spaces[5] == spaces[8])):
            return spaces[2]
    if (spaces[3] != " "):
        if (spaces[3] == spaces[4] == spaces[5]):
            return spaces[3]
    if (spaces[6] != " "):
        if (spaces[6] == spaces[7] == spaces[8]):
            return spaces[6]
    if (" " not in spaces):
        return "Nobody"
    return False

# Define function to get next move from user input
def getMove():
    print "\n"
```

```
    print "It's " + player + "'s turn"
    print "Where would you like to move?"
    print guideBoard

    # Get user input
    move = input("Specify which space to place your " + player + ": ")

    # Change the appropriate space variable to record the move
    # and increment the turn counter
    if ((move == 0) & (spaces[0] == " ")):
        spaces[0] = player
    elif ((move == 1) & (spaces[1] == " ")):
        spaces[1] = player
    elif ((move == 2) & (spaces[2] == " ")):
        spaces[2] = player
    elif ((move == 3) & (spaces[3] == " ")):
        spaces[3] = player
    elif ((move == 4) & (spaces[4] == " ")):
        spaces[4] = player
    elif ((move == 5) & (spaces[5] == " ")):
        spaces[5] = player
    elif ((move == 6) & (spaces[6] == " ")):
        spaces[6] = player
    elif ((move == 7) & (spaces[7] == " ")):
        spaces[7] = player
    elif ((move == 8) & (spaces[8] == " ")):
        spaces[8] = player
    # Print error and reprompt if input is invalid
    else:
        print "Please enter a valid move."
        getMove()

while (gameOver() == False):
    # Determine whose turn it is
    if (turn % 2 == 0):
        player = "X"
    else:
        player = "O"

    # Get the next move
    getMove()

    # Increment the turn counter
    turn += 1

    # Print the board in its current state
    printBoard()

print "GAME OVER - " + gameOver() + " wins!"
```

The following screenshot shows a sample of the output: