Introduction to Python 3.x and LaTeX

Lab 1

Spring 2022

1 Purpose

Introduce the basic programming tools and methods of Python 3.x and LaTeX that are necessary for completion of this course.

2 Deliverables Overview

The deliverables for this lab will be different from most labs this semester. Since a major focus of ECE 351 is writing high-quality reports (in preparation for future courses, like ECE 450, and your career), each report will need to be professional, organized, and succinct. Your reports and prelabs all **must** be written using a LaTeX editor, such as TeXStudio or Overleaf. A sample report should be uploaded on BBLearn.

For Lab 1 a formal report is not required, instead you will follow the lab handout to get acquainted with the Spyder IDE and with LaTeX commands and write a short summary of each section including section headers.

3 Part 1

3.1 Purpose

To become familiar with the Spyder IDE and using Python 3.x.

3.2 Deliverables

1. Summary for this section.

3.3 Tasks

1. Read the information on the following link: https://docs.spyder-ide.org/overview.html.
2. Follow the interactive tutorial (optional)
3. Read over the Spyder keyboard shortcut cheat sheet on BBLearn. *If you can't access this, please notify the lab instructor.*

4. Open a new file in Spyder and immediately save it as LastName_FirstName_ECE351_Lab1. *Note: Spyder will not run any code if the file isn't saved first, so this is a good habit to get into for all future labs.*

4 Part 2

4.1 Purpose

Introduce methods for defining variables, arrays, and matrices in python, while exploring some useful operations for each. Introduce Python syntax, packages, and proper formatting for plots. Introduce syntax for complex numbers in Python, using `numpy`. Introduce some simple commands that can make life easier while using Python and while debugging.

4.2 Deliverables

Summary for this section.

4.3 Tasks

Note: Don't forget to import any packages you think you might need at the beginning of your code.

1. Defining a variable in python is simple, as you do not need to specify a variable type (as is necessary in languages like C, C++, etc.). To view a variable's contents, use the `print()` command, as shown below. Notice the variable name is not shown with the variable by default, but this can be done by first printing a string (denoted by either apostrophes ' or quotation marks "), and units can be printed using the same method. Additionally, rounding and math operations can be performed within a `print()` statement, and new lines can be printed using '\n'. Notice each variable or expression is separated by a comma ,. Type the following code into Spyder and then run it:

```python
import numpy
import scipy.signal
import time

t = 1
print(t)
print("t =",t)
print('t =',t,"seconds")
print('t is now =',t/3,'\n...and can be rounded using `round()`',round(t/3,4))
```

This should output the following in the iPython Console.

```
1
t = 1
t = 1 seconds
t is now = 0.3333333333333333
...and can be rounded using `round()` 0.3333
```

2. To square a variable in Python, we do not use ^. Instead, we use **, as shown below:

```python
print(3**2)
```

This should output the following in the iPython Console.

```
9
```

3. Variables that contain multiple values are called lists in Python. For many function used in this lab, we will prefer to use `numpy` arrays, which operate similarly to lists. Both are demonstrated below.

```python
list1 = [0,1,2,3]
print('list1:',list1)
list2 = [[0],[1],[2],[3]]
print('list2:',list2)
list3 = [[0,1],[2,3]]
print('list3:',list3)
array1 = numpy.array([0,1,2,3])
print('array1:',array1)
array2 = numpy.array([[0],[1],[2],[3]])
print('array2:',array2)
array3 = numpy.array([[0,1],[2,3]])
print('array3:',array3)
```

This should output the following in the iPython Console.

```
list1: [0, 1, 2, 3]
list2: [[0], [1], [2], [3]]
list3: [[0, 1], [2, 3]]
array1: [0 1 2 3]
array2: [[0]
[1]
[2]
[3]]
array3: [[0 1]
[2 3]]
```

Above, list1 and array1 are row vectors, list2 and array2 are column vectors (despite looking different from one another when printing), and list3 and array3 are 2x2 matrices. Further,notice that for every `numpy.array()`, we must type out `numpy.` before `array()` since `array()` is a function within the `numpy` package. Given that the package must be specified for each use of a function within that package, it is desirable to save some time by not having to write out `numpy` every time. This can be done by changing your import line at the top of your code to match the code below.

```python
import numpy as np
import scipy.signal as sig

# Now you can call each package in the following manner
print(np.pi)
# Notice how instead of typing out numpy you can now type np
```

4. The number sign # is used in Python to make a line of code a comment. Thoroughly commenting your code is encouraged, as it demonstrates you are comfortable with the purpose of the various functions used and makes it easier for someone else to debug your code, since there are many different ways to code something correctly.

```python
# This is a comment, and the following statement is not executed:
# print(t+5)
```

5. If you want to create a larger array, Python and its packages (i.e. `numpy`) have a number of ways to do this.

```python
print(np.arange(4),'\n',
      np.arange(0,2,0.5),'\n',
      np.linspace(0,1.5,4))
```

This will output the following in the iPython console.

```
[0 1 2 3]
[0.  0.5 1.  1.5]
[0.  0.5 1.  1.5]
```

An important concept in Python to notice is that the lower boundary of the `numpy.arange()` function (and Python's `range()` function, introduced below) is inclusive, but the upper boundary is exclusive. In other words, despite passing 2 to `numpy.arange()` as its upper boundary, the final value in the created array is 1.5. In situations where you would like to go all the way up to a certain value, you must add one step size to it (e.g. `numpy.arange(0,2+0.5,0.5)`). Not shown above is Python's `range()` function, which operates like the `numpy.arange()` function but with the limitation that the step size (third argument) must be an integer (i.e. cannot be a float). `range()` is convenient to use for for loops (introduced in Lab 2), but in general `numpy.arange()` is more versatile.

6. Indexing lists and arrays is a major part of this lab. In Python, this can be done as shown below. It is important to note that indexes in Python start at 0. Also, notice `numpy` arrays offer more flexible indexing, making them a preferable option in many cases.

```python
list1 = [1,2,3,4,5]
array1 = np.array(list1) # definition of a numpy array using a list
print('list1 :',list1[0],list1[4])
print('array1:',array1[0],array1[4])
array2 = np.array([[1,2,3,4,5], [6,7,8,9,10]])
list2 = list(array2)
print('array2:',array2[0,2],array2[1,4])
print('list2 :',list2[0],list2[1])
# It is best to use numpy arrays for indexing specific values
# in multi-dimensional arrays
```

This should output the following in your iPython console.

```
list1 : 1 5
array1: 1 5
array2: 3 10
list2 : [1 2 3 4 5] [ 6  7  8  9 10]
```

To access an entire row or column of a `numpy.array()`, use the following notation.

```python
print(array2[:,2], array2[0,:])
```

This will output:

```
[3 8] [1 2 3 4 5]
```

7. Sometimes it is desirable to define a matrix as an array of zeros or ones. To do this, use the `numpy.zeros()` and `numpy.ones()` functions. See the examples below.

```python
print('1x3:',np.zeros(3))
print('2x2:',np.zeros((2,2)))
print('2x3:',np.ones((2,3)))
```

Your output should like this.

```
1x3: [0. 0. 0.]
2x2: [[0. 0.]
 [0. 0.]]
2x3: [[1. 1. 1.]
 [1. 1. 1.]]
```

8. The following code shows how to use `matlpotlib.pyplot` to plot in python. The comments in the code will explain what each funtion does. If this is still unclear look up the documentation on `matplotlib.pyplot`.

```python
# Sample code for Lab 1

import numpy as np
import matplotlib.pyplot as plt

# Define variables
steps = 0.1 # step size
x = np.arange(-2,2+steps,steps) # notice the final value is
                                # `2+steps` to include `2`
y1 = x + 2
y2 = x**2

# Code for plots
plt.figure(figsize=(12,8)) # start a new figure, with
                           # a custom figure size
plt.subplot(3,1,1) # subplot 1: subplot format(row, column, number)
plt.plot(x,y1) # choosing plot variables for x and y axes
plt.title('Sample Plots for Lab 1') # title for entire figure
                                    # (all three subplots)
plt.ylabel('Subplot 1') # label for subplot 1
plt.grid(True) # show grid on plot

plt.subplot(3,1,2) # subplot 2
plt.plot(x,y2)
plt.ylabel('Subplot 2') # label for subplot 2
plt.grid(which='both') # use major and minor grids
                       # (minor grids not available
                       # since plot is small)

plt.subplot(3,1,3) # subplot 3
plt.plot(x,y1,'--r',label='y1')
plt.plot(x,y2,'o',label='y2') # plotting both functions on one plot
plt.axis([-2.5, 2.5, -0.5, 4.5]) # define axis
plt.grid(True)
plt.legend(loc='lower right') # prints a legend on the plot
plt.xlabel('x') # x-axis label for all three subplots (entire figure)
plt.ylabel('Subplot 3') # label for subplot 3
plt.show() ### --- This MUST be included to view your plots! --- ###
```

The following plots should show up in your iPython console. *Note: If you want to have plots show up in a seperate window go to Tools >Preferences >iPython Consoles >Graphics >Select Automatic in the dropdown menu under Graphics backend.*

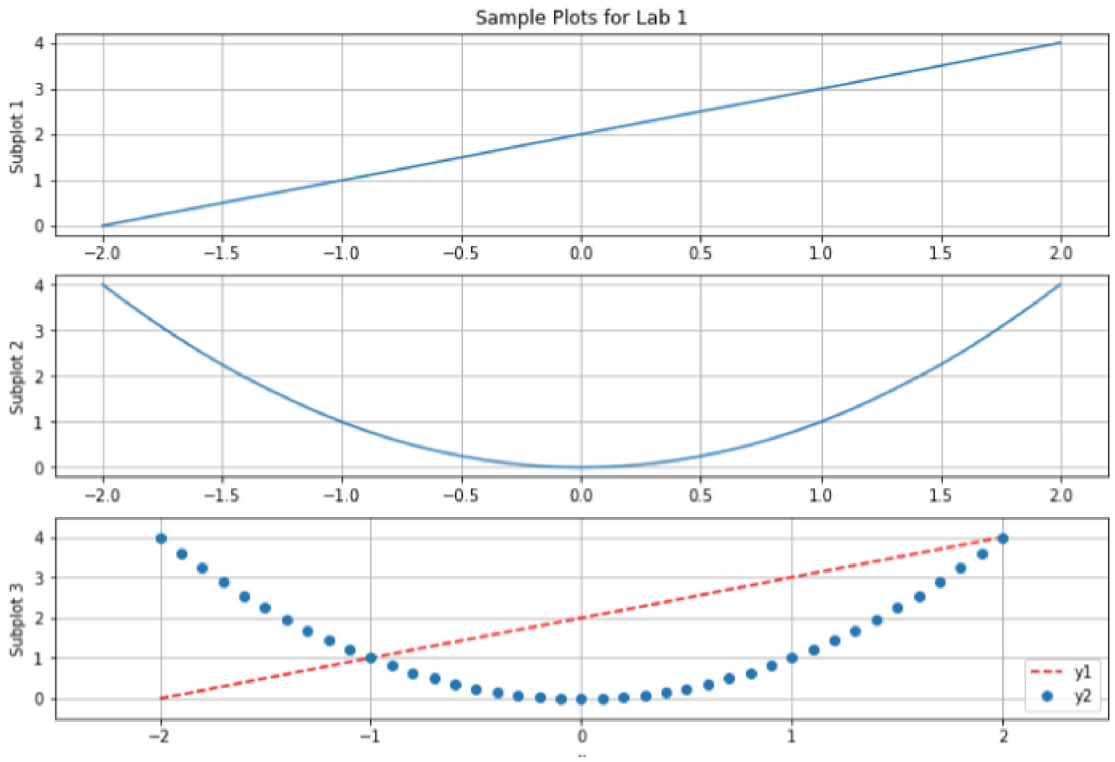

Figure 1: Sample plots for lab 1

First, notice the hierarchy: a figure can contain a single plot or multiple subplots,and a subplot can contain one or multiple traces (individual lines plotted). Regarding the title and axis labels, each figure has a single title and (when the units are the same) a single x-axis label, and each subplot has a y-axis label. Looking at the code above, we see the title for the entire figure is implemented as the title of the first (top) subplot, and the x-axis label for the entire figure (again, assuming the units are the same for all three subplots) is implemented on the last (bottom) subplot. Regarding subplot syntax, suppose you have `plt.subplot(row,col,num)`. In this case, `row` will be the number of rows of subplots, `col` will be the number of columns of subplots, and `num` will be which subplot in the subplot "array". When deciding whether to use subplots instead of separate figures, consider whether the individual plots are related to one another, ad whether they will be visible if plotted in subplots.

9. Complex numbers are defined in Python as follows, with j being the imaginary number.

```python
cRect = 2 + 3j
print(cRect)

cPol = abs(cRect) * np.exp(1j*np.angle(cRect))
print(cPol) # notice Python will store this in rectangular form

cRect2 = np.real(cPol) + 1j*np.imag(cPol)
print(cRect2) # converting from polar to rectangular
```

This will output the following in your iPython console.

```
(2+3j)
(2+2.9999999999999996j)
(2+2.9999999999999996j)
```

In some cases, you will receive a `nan` from a negative number inside the `numpy.sqrt()` function. To tell the interpreter (what Python uses instead of a compiler) that you want a complex number, include an addition of 0j in the square root, as below.

```python
print(numpy.sqrt(3*5 - 5*5))
```

This will show an error if you try to run this code.

```
nan
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1:
Run time Warning: invalid value encountered in sqrt
"""Entry point for launching an IPython kernel.
```

Make the following change to get the code to run correctly.

```python
print(numpy.sqrt(3*5 - 5*5 + 0j))
```

Now, this should output.

```
3.1622776601683795j
```

10. The following code listing shows a number of packages that will be used throughout this semester, for your reference, once again the `as` is optional and arbitrary, as mentioned before. For more detailed information on each, plenty of documentation is readily available for each. This list may not be exhaustive of the packages used throughout the semester in this course, and each package only needs to be included in labs for which it is needed.

```python
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp
import scipy.signal as sig
import pandas as pd
import control
import time
from scipy.fftpack import fft, fftshift
```

11. The following code listing shows some Python commands, which are given for your information. The comments describe their purpose. Throughout the semester, you will likely become familiar with many other functions (in addition to these) as you work through the labs.

```python
range() # create a range of numbers (nice for `for` loops)
np.arange() # create a numpy array that is a range of number with a defined
            # step size
np.append() # add values to the end of a numpy array
np.insert() # add values to the beginning of a numpy array
np.concatenate() # combine two numpy arrays
np.linspace() # create a numpy array that contains a specified (linear) range
              # of values with a specified number of elements
np.logspace() # create a numpy array that contains a specified (logarithmic,
              # base specified) range of values with a specified number of
              # elements
np.reshape() # reshape a numpy array
np.transpose() # transpose a numpy array
len() # return the number of elements in an array (horizontal)
.size # return the number of elements in an array (vertical)
.shape # return the dimensions of an array
.reshape # reshape the dimensions of an array (similar to numpy.reshape()
         # above)
.T # transpose an array (similar to np.transpose() above)
```

5 Part 3

5.1 Purpose

To become familiar with pep8 coding practices.

5.2 Deliverables

1. Summary for this section.

1. Use 4 spaces per indentations and no tabs. Tabs are only used for if and else statements and function definitions.
   Example:

```python
# Aligned with opening delimiter.
x = func_name(var_one, var_two,
              var_three, var_four)

# More indentation needed to distinguish this section from the rest.
def func_name(
        var_one, var_two, var_three,
        var_four):
    print(var_one)

# Hanging indents should add a level.
x = func_name(
    var_one, var_two,
    var_three, var_four)
```

2. Use docstrings to define particular functions or programs.
   Example:

```python
"""This is single line docstring for the next function"""
def base_func():

"""This is
a
multiline comment for the next function"""
def one_func():
```

3. Wrap lines so that they don't exceed 79 characters.The lines can be wrapped using parenthesis, brackets, and braces.
   Example:

```python
    with open('/path/from/where/you/want/to/read/file') as file_one, \
         open('/path/where/you/want/the/file/to/be/written', 'w') as file_two:
        file_two.write(file_one.read())
```

   To set a visual reminder, go to Tools < Preferences < Editor and check box that says Show vertical line after 79 characters.

4. Use regular and updated comments. Block comments with more than one sentence should form complete sentences with the first word capitalized, unless it is an identifier that begins with a lower case letter. Short comments don't need periods. Comments can be written followed by a single #.
   Example:

```python
counter = counter + 1 #increment
print('list 2: ', list2[0], list2[1])
# It is best to use numpy arrays for indexing specific values in
# multi-dimensional arrays
```

5. Use spaces around operators and after commas, but not directly inside bracketing constructs.
   Example:

```python
x = f(1, 2) + g(3, 4)
func(method[1], {numbers: 2})
```

6. Use these naming conventions:
   b (single lowercase letter)

   B (single upper case letter)

   lowercase

   lower_case_with_underscores

   UPPERCASE

   UPPER_CASE_WITH_UNDERSCORES

   CapitalizedWords

   Note: While using abbreviations in CapWords, capitalize all the letters of the abbreviation. Thus HTTPServerError is better than HttpServerError.

   mixedCase (differs from CapitalizedWords by initial lowercase character!)

   Capitalized_Words_With_Underscores

   The convention is to use CapWords for classes and lower_case_with_underscores for functions and methods.

   If more detail is needed, please refer to this link for full documentation on pep8 coding practices: `https://pep8.org/`

6 Part 4

6.1 Purpose

To become familiar with some LaTeX commands.

6.2 Deliverables

1. Summary for this section.

6.3 Tasks

1. Read through the LaTeX cheat sheets provided by the instructor.
2. Find a template that you like to use in all lab reports for this class, if you want to start your own document to gain a more in-depth knowledge of how LaTeX works consider using the following code for your package imports. (*Note: All this needs to be before the begin document command*).

```
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %                                                                     %
        % Your heading created in lab 0                                       %
        %                                                                     %
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        \documentclass[11pt,a4]{article}
\usepackage[utf8]{inputenc}
\usepackage{fullpage}
\usepackage{hyperref}
\usepackage{listings}
\usepackage{xcolor}
\usepackage{graphicx}

\definecolor{codegreen}{rgb}{0,0.6,0}
\definecolor{codegray}{rgb}{0.5,0.5,0.5}
\definecolor{codeblue}{rgb}{0,0,0.95}
\definecolor{backcolour}{rgb}{0.95,0.95,0.92}

\lstdefinestyle{mystyle}{
    backgroundcolor=\color{backcolour},
    commentstyle=\color{codegreen},
    keywordstyle=\color{codeblue},
    numberstyle=\tiny\color{codegray},
    stringstyle=\color{codegreen},
    basicstyle=\ttfamily\footnotesize,
    breakatwhitespace=false,
    breaklines=true,
    captionpos=b,
    keepspaces=true,
    numbers=left,
    numbersep=5pt,
    showspaces=false,
    showstringspaces=false,
    showtabs=false,
    tabsize=2
}

\lstset{style=mystyle}

\title{Title}
\author{Your Name}
\date{\today}
```

3. You might want to look up more documentation on how to insert figures, write complex

mathematical equations, and insert code into your report.

7 Questions

1. Which course are you most excited for in your degree? Which course have you enjoyed the most so far?
2. Leave any feedback on the clarity of the expectations, instructions, and deliverables.