Python 1 - The Sun's Declination Jupyter Notebook Template

September 17, 2020

0.1 Chris Compierchio - compier1

1 Python Assignment 1: The Sun's Declination Jupyter Notebook Template

1.1 Before you begin

Assuming you have loaded this file into your Jupyter Notebooks workspace, make sure to press the "play" button at the top of the page in each box. This will render the Markdown into nicely-formatted text and execute all of the sections of Python code.

1.2 Introduction

This Jupyter Notebook will serve as the template for your Python 1 assignment. You will be working through the content it contains and then filling in your own work in the spaces provided. There are detailed instructions on how to prepare your assignment available on Quercus under "Python 1: The Sun's Declination" in the week 2 module. Make sure you consult those instructions.

Those of you with some programming experience will find the early parts of this assignment very easy. That's okay. Please just speed through them and don't worry about it. But for those who are new to programming, the early portions should give you some experience in modifying pre-existing code and showing you the syntax for typical calculations and plots that you will be asked to produce later in the course.

1.3 PART 1: MODIFYING CODE

The following sections of code will take you through a short progression from one of the very simplest pieces of code you can write to a few things that are a little bit more complex. In case you wind up searching the Internet for help, keep in mind that this code is written in Python 3. You will find a lot of documentation for previous versions of Python which may not be compatible.

As you modify each of the following segments of code, remember to hit the "play" button at the top of the notebooks interface so that the cell containing the code "runs" (you can also hit Shift+Return for the same functionality). It should display the output of the piece of code just below that cell. You can run every cell in this notebook before you modify it, just to see what it produces. If you find any of this confusing or overwhelming, please refer back to the programming resources in the "Guides" section on Quercus. The "Introduction to Programming" guide contains links to some documents that should help you.

Let's get started!

Modify the following print statement so that it prints "This is a print statement in Python."

[1]:
```python
# QUESTION 1: Modify the following statement

print("This is a print statement in Python.")
```

```
This is a print statement in Python.
```

Now modify the following print statement so that it prints the output of the calculation 3 times 7.

[2]:
```python
# QUESTION 2: Modify the following statement

print("Three times seven equals",3*7)
```

```
Three times seven equals 21
```

In programming, we often use loops to execute the same piece of code repeatedly, altering some aspect each time it executes. For example, we might want to print The numbers from 1 to 10, which we can do like this:

[3]:
```python
for number in range(1,11):
    print(number)
```

```
1
2
3
4
5
6
7
8
9
10
```

There are a few things to note here. One is the use of a colon (:) at the end of the first line. In this case, the colon indicates that we are beginning a loop. The subsequent lines must be indented. If you are writing your program using Jupyter Notebooks, it will automatically indent code for you on lines following a colon. Note that we have also used the “range” function and given it the parameters (1,11). This means we are asking it to give us all the whole numbers greater than or equal to one and less than 11. This is why it stops at 10. 10 is the last whole number less than 11.

In the blank cell that follows, write a for loop that will print the numbers from 39 to 52, including both ends of that sequence.

[4]:
```python
# QUESTION 3: Write a simple for loop

for number in range(39,53):
    print(number)
```

```
39
40
```

```
41
42
43
44
45
46
47
48
49
50
51
52
```

Now try combining the two things you have learned so far: print statements and for loops. Write a for loop that prints “The square of (number) equals (result)”, for all numbers from 1 to 10.

[5]:
```python
# QUESTION 4: Write a more complicated for loop

for number in range(1,11):
    print("the square of " + str(number) + " equals " + str(number*number))
```

```
the square of 1 equals 1
the square of 2 equals 4
the square of 3 equals 9
the square of 4 equals 16
the square of 5 equals 25
the square of 6 equals 36
the square of 7 equals 49
the square of 8 equals 64
the square of 9 equals 81
the square of 10 equals 100
```

1.4 PART 2: READING IN DATA & CALCULATIONS

In this course, one of the things you will be doing fairly often is reading and manipulating data, including performing mathematical calculations and making various types of plots. In this assignment, for example, you will be reading in a table showing the predicted location of the Sun in the sky as a function of time.

Before we can read and plot data, we have to recruit some pre-existing code to help us out. Writing all of the code ourselves would be very tedious, so instead we use “libraries” of code. The libraries you will need already exist on the server you are using. To incorporate them into your code, you use the “import” function like this:

import LibraryWithALongName as lb

This line of code will import the library called “LibraryWithALongName” and rename it “lb”. The purpose of abbreviating the name is so that we don’t have to constantly type out the incredibly long but descriptive name of the library. It makes the code harder to read, but much easier to produce. So, there is a trade-off.

Within the library "lb", There will be many different pieces of code. You can think of these as books within an actual library. To invoke any given book, we simply append its name to that of the library with the period, like this:

lb.book

Usually these "books" are "methods" or "functions" that do something. Often we have to give them variables or "arguments" to do something *to*. That would look like this:

lb.book(arguments)

Again, if you aren't a computer scientist, you don't need to worry too much right now about the technical terminology for all of this or how it works in detail. We will introduce the syntax and usage of most functions you will need to complete the assignments.

Here is some code to read in a dataset that is contained in a text file, in the form of columns separated by spaces. The code starts by importing a library called numpy, which contains many useful functions for working with data. Any line in the code that begins with a hash mark (#), is a comment, meaning a note to ourselves that will not actually be run as code. These comments are included to help us remember what our code does. You should always include them to help the TAs interpret your code. If you don't include enough of them, your grade will be reduced.

Try to see if you can read the following code with the help of the comments. Note that the convention is usually that we place the comment above the section of code to which it pertains.

[13]:
```python
import numpy as np

# We will investigate the elevation of the Sun as viewed from different points␣
 ↪on Earth over the
# course of a year.
# First, we will read in an ephemeris for the Sun - an ephemeris gives the␣
 ↪trajectory of astronomical objects
# in the sky over some time period. You can get these from various sources. We␣
 ↪have used the HORIZONS service
# from the NASA Jet Propulsion Laboratory to calculate these values for the Sun␣
 ↪in 2020.
# https://ssd.jpl.nasa.gov/?ephemerides

# If you look at the file (just double click in your jupyter folder), you will␣
 ↪see that it contains, in columns,
# the date, the Julian date, the Sun's Right Ascension (R.A.), and Declination␣
 ↪(Dec.), along with a few other things.
# The R.A. is in units of hours/minutes/seconds, while the Dec. is in units of␣
 ↪degrees/minutes/seconds.
# See C&O Chapter 1.3 for a description of R.A. and Dec.

# The numpy function 'loadtxt' can read in text data with columns, and load␣
 ↪each column into an array like so:
jdate, ra1, ra2, ra3, de1, de2, de3 = np.loadtxt('Python1_Solar_ephemeris_JPL.
 ↪txt',skiprows=34,
                                                                   ␣
 ↪usecols=(2,3,4,5,6,7,8),unpack=True)
```

```
# Note that if your text file has columns that are strings (letters), and you␣
 ↪try to read them into an array,
# 'loadtxt' will not work! If you need to read strings into python, the numpy␣
 ↪function 'genfromtxt' is a better bet.
# Here, because we are not reading in those particular columns, we can use␣
 ↪'loadtxt'.

# Next we will create new arrays for the R.A. and Dec. where R.A. is in␣
 ↪fractions of hours and Dec. is in fractions
# of degrees.
# In python, you can perform mathematical operations on arrays similar to how␣
 ↪you would perform an operation
# on single numbers.

de_deg = abs(de1 + (de2+de3/60.)/60.)
# Here we use the numpy 'where' function to determine where the declination is␣
 ↪negative, and make sure our
# declination in degrees is also negative for those declinations.
# The where function returns all the indices in the array where the relation in␣
 ↪the brackets is true:
de_deg[np.where(de1 < 0)] = de_deg[np.where(de1 < 0)] * (-1.)

# Note that we don't need to worry about negative values in R.A.
```

[14]:
```
# QUESTION 5: Write your own calculation to create an array 'ra_hours'.

jdate, ra1, ra2, ra3, de1, de2, de3 = np.loadtxt('Python1_Solar_ephemeris_JPL.
 ↪txt',skiprows=34,
                                                                    ␣
 ↪usecols=(2,3,4,5,6,7,8),unpack=True)

ra_hours = abs(ra1 + (ra2+ra3/60.)/60.)

# You can always use print statements to see what you've calculated. You can␣
 ↪print whole arrays to the screen, but
# sometimes that will take up a lot of space! Instead, use a range of indices␣
 ↪to only look at a subset of your data.
# Check the first few values in your array to see whether what you calculated␣
 ↪looks reasonable:
print('First ten values in my R.A. array:', ra_hours[0:9])
```

```
First ten values in my R.A. array: [18.70648889 18.78012222 18.85366111 18.9271
19.00042778 19.07363611
 19.14671944 19.21967222 19.29248333]
```

1.5 Part 3: Plotting

It is very easy to make simple plots in python. We will often make use of the matplotlib library for this. Matplotlib allows you to make many different types of plots, and contains many different functions you can use to present your data in a readable way. You can find the documentation for matplotlib, including examples, here.

Next we will plot the Dec. of the Sun as a function of date in 2020:

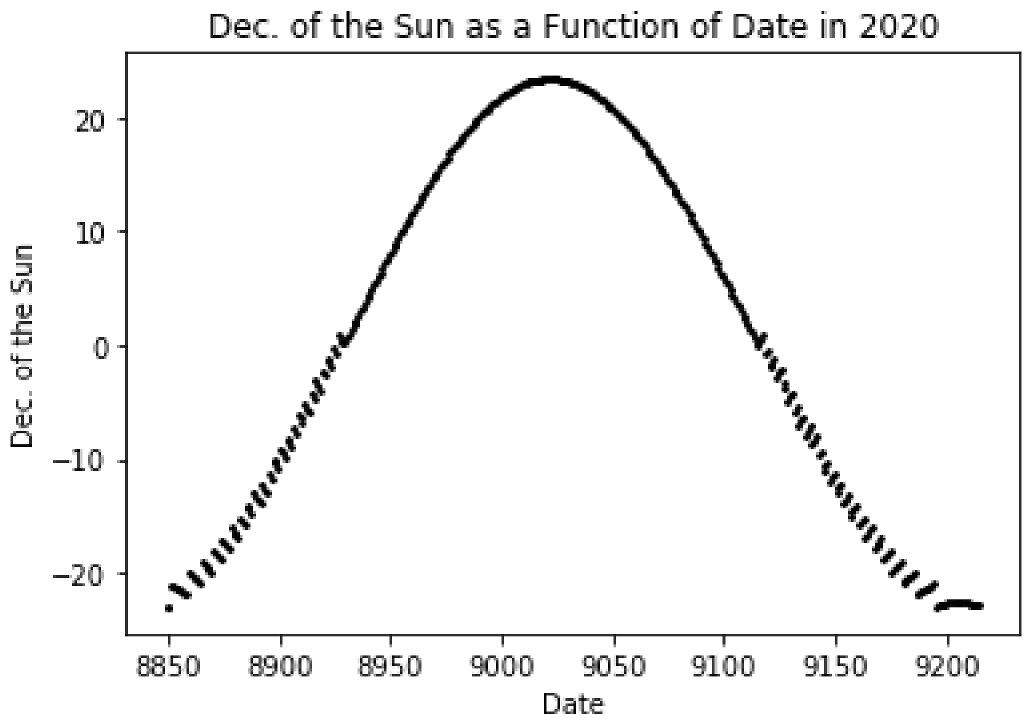
[15]:
```
# QUESTION 6: Complete the plot
import matplotlib.pyplot as plt
import matplotlib
# This command allows your plots to show up below your code in your Jupyter␣
↪notebook
%matplotlib inline

# Here, subtract off a common number (use 2450000) from the Julian date to make␣
↪it easier to see the
# labels on the x-axis

jdate_new = jdate - 2450000

# Create a figure to plot in. Inside the brackets, you can set things like␣
↪figsize=(6,4),
# or the dpi (dots per inch).
fig=plt.figure()
# 'scatter' creates a scatter plot. You can use different keywords to make the␣
↪data points
# different shapes, sizes, or colours. 'plot' will join points together.
plt.scatter(jdate_new,de_deg,s=3,color='black')
# Always label your axes. Include labels within the quotes here:
plt.xlabel('Date')
plt.ylabel('Dec. of the Sun')
plt.title("Dec. of the Sun as a Function of Date in 2020")
```

[15]:
```
Text(0.5, 1.0, 'Dec. of the Sun as a Function of Date in 2020')
```



You should be able to assess whether the above plot looks reasonable. If not, check your math in the previous box!

Next, based on the Sun's declination, calculate the Sun's maximum elevation as viewed from Toronto for each day and plot the results. If you're not sure how to do this, consult C&O Chapter 1.3. You will need to look up Toronto's latitude. Overlay on the plot the same results for your hometown - modify the label and legend plot commands to indicate what location you are using. If your hometown is Toronto, please select a different city.

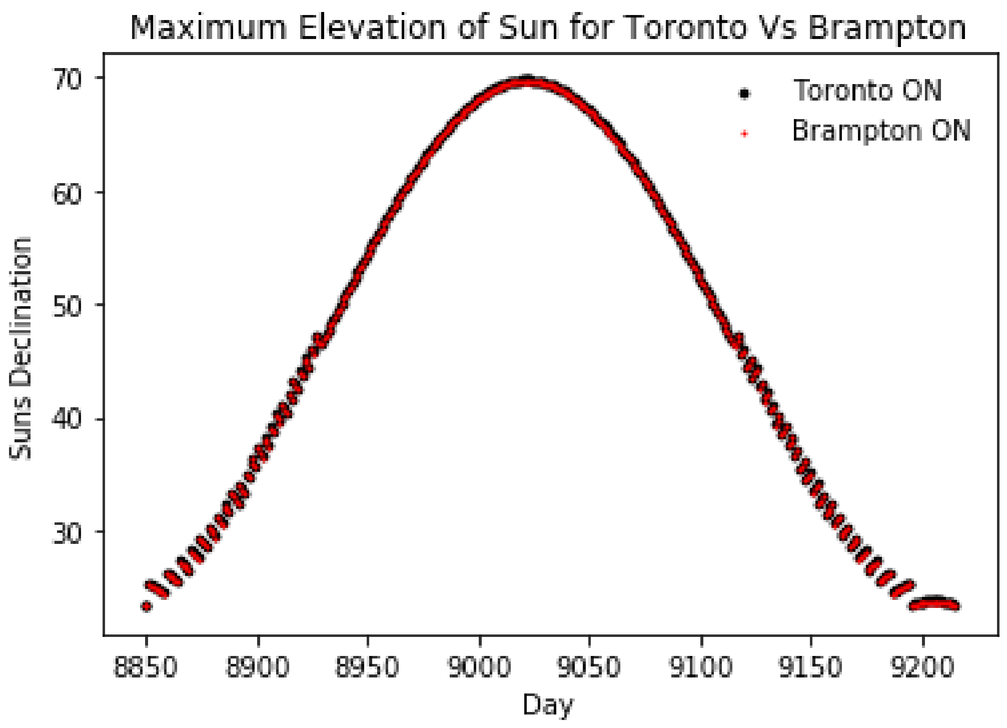
[20]:
```python
# QUESTION 7: Calculate and plot the Sun's maximum elevation as viewed from␣
↪Toronto
lat_toronto = 43.6532 # degrees
max_el = 90 - lat_toronto + de_deg

# What city are you using here? Add the name to the plot label below.
lat_brampton = 43.7315 # degrees
max_el_brampton = 90 - lat_brampton + de_deg

fig=plt.figure()
# Use the appropriate arrays to create the scatter plots here
# Feel free to adjust colours, point sizes, point shapes, etc. if you like
# but make sure your plot is always readable!
plt.scatter(jdate_new, max_el,s=8,color='black',label='Toronto ON')
plt.scatter(jdate_new, max_el_brampton,s=1,color='red',label='Brampton ON')
plt.xlabel('Day')
plt.ylabel('Suns Declination')
```

```
# By including labels in your scatter plots, we can include a legend easily:
plt.legend(frameon=False)
plt.title("Maximum Elevation of Sun for Toronto Vs Brampton")
```

[20]:
```
Text(0.5, 1.0, 'Maximum Elevation of Sun for Toronto Vs Brampton')
```



1.6 Part 4 - Other planets

The Right Ascension and Declination coordinate system is based on the latitude and longitude system of Earth. Think about how this is reflected in the range of Declination values the Sun passes through in a year.

If we assume all the planets in our Solar system orbit around the Sun in the same plane, can you determine the minimum and maximum elevation of the planets in the Table below? Fill in the table with the planets' inclination (their tilt) relative to the equatorial plane (the plane of the planets' orbits) from Appendix C. Then calculate the Sun's minimum and maximum elevation for a point at a latitude of 45 degrees (North) for each planet and fill in the appropriate table columns.

[17]:
```
## QUESTION 8: Perform any calculations here

##max elevation = 90 - latitude + (equatorial inclination)
##min elevation = 90 - latitude - (equatorial inclination)
```

1.6.1 QUESTION 8: Fill in the markdown table below with your results

Planet	Equatorial Inclination	Min. El at 45ř	Max. El at 45ř
Mercury	0.01deg	44.99	45.01
Mars	25.19deg	19.81	70.19
Jupiter	3.13deg	41.87	48.13
Uranus	97.77deg	-52.77	142.77

To better show your work, modify the Markdown code below to insert below a drawing or sketch you have made of the relevant angles. You must submit this image file with your notebook or we will not be able to see it! Follow the instructions for filenames when you submit your work.

1.7 Handing In

To hand this assignment in, follow the instructions in the "Writing Assignments in Jupyter Notebooks" guide on Quercus.

Bibliography

```
Toronto, ON, Canada. Lat Long Finder, www.latlong.net/place/toronto-on-canada-27230.html.

Brampton, ON, Canada. Lat Long Finder, www.latlong.net/place/brampton-on-canada-4037.html.
```

[ ]: