Note: Original form is a jupyter notebook, which is an interactive coding workbook. Tyler Waterman contributed with problems 3,4 and 7 as well as parts of 1,2,5 and 8

Assignment #1

1. Learning Python (10 points)

1.1 Print "Hello, World!"

1.2 Comment out the text below.

```
Running this text without commenting it out will give an error
```

1.3 Create a variable named course and assign the value ESDA to it. Print the variable.

1.4 Define x to be 5 and y to be 12. Create a variable called z, assign x + y to it, and print the result.

1.5 Use the len method to print the length of the string = "CEE-690-02".

1.6 Print the first character of the string = "ESDA".

1.7 Print the characters from position 2 to position 6 (not included) from the string "Big data".

1.8 Convert the value of string = "CLASSROOM" to lower case.

1.9 Use the correct logical operator to check if at least one of the two statements is True.

```
if (5 == 10) OPR (4 == 4):
    print("At least one of the statements is true")
```

1.10 Print the first and last item in the vehicles list defined below.

```
vehicles = ['car','truck','bus']
```

1.11 Change the value from "bus" to "motorcycle" in the vehicles list and print the result.

1.12 Use the append method to add "bicycle" to the vehicles list and print the result.

1.13 Print the value of the "name" key of the course dictionary defined below.

```
course= {
    "name":"Environmental Spatial Data Analysis",
    "id":"CEE690-02",
    "semester":"Fall",
    "year":2020
}
```

1.14 Change the "year" value from 2020 to 2021 and print the revised dictionary.

1.15 Add the key/value pair "grade" : "A" to the course dictionary. Print the revised dictionary.

1.16 Use the clear method to empty the course dictionary. Print the revised dictionary.

1.17 Set make1 = "Toyota", make2 = "Mercedes", make3 = "Ford", and make4 = "Ford". Print "make1 and make2 or make3 and make4 are the same" if make1 is equal to make2, or make3 is equal to make4 (Use if statements).

1.18 Print i as long as i is less than or equal to 9.

1.19 Modify the loop defined above by adding a conditional statement to stop the loop if i is 5.

1.20 Loop through and print the items in the vehicles list defined below.

```
vehicles = ["car","bus","motorcycle"]
```

1.21 Modify the loop defined above such that when the item value is "bus", the loop is exited.

1.22 Use the range function to define a list of integers from 0 to 9. Then iterate and print those values.

1.23 Create and call a function called hello_world that prints "Hello, World!".

1.24 Create a function called many_hello_worlds that calls the hello_world function defined above n times. The parameter n should be passed into the function. Call the function many_hello_worlds with n = 3.

1.25 Create and call a function that computes and prints the third (1) through the twelfth elements (89) of the Fibonacci sequence.

2. Numpy basics (10 points)

2.0 Import numpy as `np`

2.1 Create an evenly spaced 1D array of latitude values starting at -90.0 degrees and ending at 90.0 degrees with a pixel size of 0.5 degree.

2.2 Convert the following list of lists into a 2D numpy array

```
data_list = [[3,4],[2,3]]
```

2.3 Stack the following 1D arrays vertically into a single 2D array.

```
A = np.array([1,2,1,2])
B = np.array([2,1,2,1])
```

2.4 Compute the unique values of the following array. Note that simply writing them out is not sufficient.

```
np.random.seed(1)
A = np.random.randint(1,10,10000)
```

2.5 Reshape the defined 1D array A into a 2D array B. The size of each dimension of B should be defined as the square root of the total size of A. Remember that the computed dimensions of B need to be integers.

```
A = np.arange(144)
```

2.6 Create a function that uses a for loop to compute the sum of all the elements in the array A. Use %timeit to compute the time it takes to run the defined function.

```
np.random.seed(1)
A = np.random.randn(1000)
```

2.7 Now use the function np.sum() to compute the sum of the defined array A. Use %timeit to compute the time it takes to run the operation. Using the Markdown language in the additional cell below, compare the timing results with those from exercise **2.6**.

2.8 Compute the element-wise addition of the defined arrays A and B. The results should be assigned to a variable C. Then compute and print the mean, sum, standard deviation, maximum, minimum, and median of the resulting array.

```
np.random.seed(1)
A = np.random.randn(1000)
B = np.random.randn(1000)
```

2.9 Compute the element-wise remainder after dividing the defined array B by A.

```
np.random.seed(1)
A = np.random.randint(1,10,10)
B = np.random.randint(1,10,10)
```

2.10 Let $f = x / \left(1 + x^2\right) + ln\left(x\right) - sin\left(x\right)$. After creating an evenly spaced array x between 1 and 11 with 101 elements, compute f. To finish, print the mean and standard deviation of the computed array f.

2.11 Use slicing to replace the values of indices 5,1,2 in A with indices 8,8,2 in B. Note that replacing the values one by one (i.e., A[5] = B[8], A[1] = B[8]...) is not an adequate answer.

```
A = np.linspace(0,10,11)
B = A[::-1]
```

2.12 Subsample the given 2D array A every 5 elements to create a vector b. Remember to flatten A to a 1D array before subsampling.

```
A = np.arange(30).reshape(6,5)
```

2.13 Given the array `lons` defined below, use a Boolean mask to determine the pixels that are above or equal to -45❑^{o} and below or equal to 60❑^{o}. Then compute and print the number of elements that satisfy this condition.

```
minlon = -180.0
maxlon = 180.0
lons = np.linspace(minlon,maxlon,int((maxlon-minlon)/0.25)+1)
```

2.14 Given the mask computed above, determine the indices of the array lons where the mask is True. Print the index of the first element that meets that condition.

2.15 Lets play checkers. Create a standard 8x8 checkerboard array (zeros and ones). Note that writing the checkerboard array out manually (i.e., A = np.array([[1,0,1,0],[0,1,0,1],...])) is not an adequate answer. You will want to first initialize a 8x8 array of zeros and then fill in with ones where necessary using slicing.

3. Dates (5 points)

3.1 Import the module datetime.

3.2 Construct an instance of the class datetime.datetime for January, 31st, 2019 at 00:45. Print the resulting variable `date`.

3.3 Assemble a list of dates consisting of all 3 hourly timestamps of February, 2020. Assume that the first time stamp is February 1st, 2020 at 00:00. Convert the resulting list into an array. Print the 40th element and the size of the resulting array.

3.4 Given the array `dates` defined above, use a Boolean mask to compute all dates that are after February 4th, 2020 at 3:00 am and before February 10th, 2020 at 6:00 pm. Create a new dates array by subsetting the original dates array with the mask. Print the resulting array.

4. Calculating the Number of Daylight Hours (10 points)

There are a few different models, with varying degrees of accuracy, to estimate the number of daylight hours at a given time and place for application in agriculture, ecology, and earth system modeling.

A relatively simplistic one from Brock (1981) is as follows:

First, the declination of the sun is given by:

$$\delta = .40928 \sin\left(2\pi \frac{283+J}{365}\right)$$

where J is the day of the year (number of days since January 1st). The sunset/sunrise hour angle is then computed as

$$\beta = \arccos\left(-\tan\phi \tan\delta\right)$$

Where ϕ is the latitude in radians. Finally the daylength, in hours, is calculated

$$D = 24\frac{\beta}{\pi}$$

4.1 Create a function named `daylength` that computes number of daylight hours at a given time and place. Your input parameters should be `lat` and `day` (for day of the year). Keep in mind that latitude is usually given in degrees but the equation requires latitude in radians.

4.2 Use the function defined above to compute the number of daylight hours in Durham, NC on July 4th 2021. The coordinates of Durham, NC are 35.9940❑^{o} N and 78.8986❑^{o} W. Print the computed number of daylight hours.

4.3 Create a class called point that is initialized using two variables `lat` and `name`. Then add the `daylength` function defined previously as a method.`

4.4 Instantiate the class `point` for Durham using the lat value defined in **4.2**. Name the variable storing the point `durham`. Print `durham` (the object not the string), as well as the `lat` and `name` variables of `durham`

4.5 Add another method `__str__` to the `point` class that returns the name of the point. This will override the default `__str__` method which you saw at work in the print statement in **4.4**. Reinstantiate the class for Durham as in **4.4**. Once again, print `durham`, as well as the `lat` and `name` variables of `durham`

4.6 Compute the number of daylight hours on July 4th in Durham using the object defined in the previous question. Note that this will give the same result as **4.2**, however, it will involve using classes instead of a single function.

5. Parsing filepaths (10 points)

The directory `/data/NED` contains the entire database of the National Elevation Dataset (NED) which provides maps of elevation over the entire Contiguous United States at a 1 arcsec (~30 meter) spatial resolution. Given the large size of the data, the entire dataset is split into 1❑^{o} x 1❑^{o} arcdegree boxes. Each file in the directory looks similar to "dem_lat3637_lon-105-104.tif". This file contains the elevation information for the region between the latitudes of 36.0 and 37.0 and the longitudes of -105.0 and -104.0.

5.1 Import the module glob

5.2 Use the function glob.glob() to make a list of all the files that end with .tif in the directory defined above. Print the resulting list.

5.3 Print the number of files in the computed list.

5.4 Subset the previously computed list to extract only files that fall within the bounding box of minlat = 39.0, maxlat = 44.0, minlon = -85.0, and maxlon = -82.0. Print the resulting list and the number of elements in the list.

6. ArcAscii to NetCDF4 (20 points)

The directory `/data/MSWEP/AAIGrid` contains spatial grids of the MSWEP dataset at a 1 month temporal resolution and 1 arcdegree spatial resolution for 2015 in the AAIGrid format. Create a single NetCDF4 file with one 3D array (time,lat,lon) that contains this data. Ensure that the created NetCDF4 is CF-compliant by following the guidelines described here. You can also use the ERA-Interim NetCDF4 file as an example (/data/era-interim/era_interim_monthly_197901_201512_upscaled.nc). Note that you will have to write your own function to parse the ArcAscii (i.e., AAIGRID) files; you may **not** use rasterio for this problem.

```
import glob
import numpy as np
def read_file(fname):
    #do magic
    fp = open(fname)
    count = 0
    tmp2 = []
    for line in fp:
        count+=1
        if count>=7:
            tmp = line.split(' ')
            tmp2.append(np.array(tmp[1:])
            break
    return #A
files = glob.glob('/data/MSWEP/AAIGrid/*')
B = []
for fname in files:
    print(fname)
    A = read_file(fname)
    B.append(A)
    break
B = np.array(B)
print(B.shape)
```

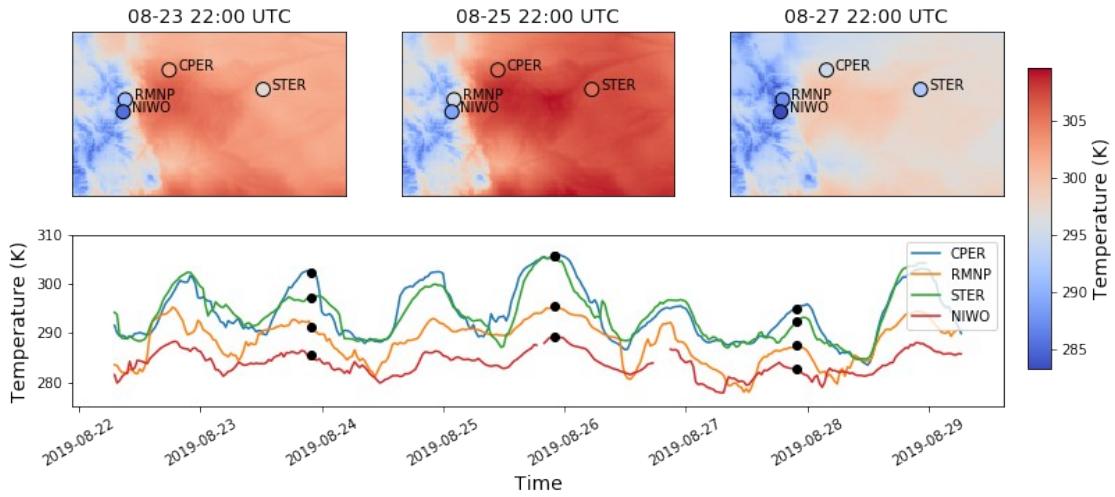
```
fp['data'][:] = B[:]
fp.close()

lats = np.linspace(1,10,100)
lat = 5.4
i = np.argmin(np.abs(lats-lat))
print(i)
```

7. Advanced Matplotlib: Showing Raster Data and Point Data for Temperature in Colorado (20 points)

Often in geospatial analysis, you may have access to high quality in-situ (point) data and lower quality satellite or reanalysis data with greater spatial coverage. Comparing the two is a common exercise. Using the reanalysis data for air temperature in August 2019 from `/data/PCF` as well as the 1 week half-hourly timeseries of tower air temperature reported from the 4 sites of the National Ecological Observation Network (NEON) in Colorado located here `/data/neon/temperature`, reverse engineer the code that created the following figure.

Note that the `.tif` files are named according to the day, and have hourly data such that `read(1)` will provide the raster for 0:00 UTC, `read(2)` will provide the raster for 1:00 UTC etc. Also note that the NEON files have a `time` in seconds since January 1st 1970 (UTC), which will need to be converted to `datetime` objects (perhaps using `datetime.timedelta`)




```
import h5py
fp = h5py.File('/data/neon/temperature/CPER_TEMP.h5')
print(fp.attrs['lat'])
print(fp.attrs['lon'])
```

8. Movie of the time evolution of global average monthly precipitation (15 points)

Using the file /data/era-interim/era_interim_monthly_197901_201512_upscaled.nc:

- Find the average precipitation for each of the 12 months over 1979 through 2015.
- Use the assembled precipitation dataset to create a movie of the time evolution of global precipitation throughout the year.

The following example will show you how to make a movie in a Jupyter Notebook: http://louistiao.me/posts/notebooks/embedding-matplotlib-animations-in-jupyter-notebooks/. [Hint: You will need to use ax.imshow() instead of ax.line()].