Assignment2_Skeleton

January 23, 2024

Hi students, this is the in-lecture notebook that you need to complete before the end of the class.

Lecture 2 is about **libraries** in Python, and we are going to cover `Numpy` and `matplotlib`. This notebook companion is for you to practice what you learned from the lecture.

0.0.1 Check your Python version

As of Janurary 1, 2020, Python has officially dropped support for `python2`. We'll be using Python 3.7 for this course.

Run the following code to see your python version!

[1]:
```
!python --version
```

```
Python 3.8.16
```

0.1 Numpy

Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

To use Numpy, we first need to import the numpy package:

You can import the a package by

`import package_name as alias`

[2]:
```
# Import packages
########################
## YOUR CODE STARTS HERE
########################

# Import the Numpy package (package_name = numpy) using the alias "np"
import numpy as np

########################
## YOUR CODE ENDS HERE
########################
```

[3]:
```
## You can use this code block to check your numpy version
## Please finish and execute the previous code block first
print("Numpy version: ", np.__version__)
```

```
Numpy version:  1.23.5
```

Note that the package name is `numpy`, which means that you cannot change it to something else. However, you have the freedom to rename this package. Python programmers usually name the `numpy` package as “np”.

This is very important when you are collaborating with other Python programmers as you don’t want to have different names for the same package.

0.1.1 Task 1: Create a simple (1D) numpy array

Let’s create a simple numpy array

Recall from last lecture, we can create a Python list with this simple command.

```
## Create a list
num_list = [1, 2, 3]

## What is the length of this list?
num_shape = len(num_list)

## Print out the result
print(num_shape)
```

We can use a python list to initialize our numpy array

```
## Initialize a numpy array with python list
num_numpy = np.array(num_list)

## Sanity Checks
print("The shape of this numpy array is", num_numpy.shape)

print("The type of this numpy array is", type(num_numpy))
```

Here is your task:

1. Create a list `myList` with values **34, -5, 0, -1, 1.5**
2. (Sanity Check) Output the length and the type of `myList`
3. Create a numpy array `myArray` and initialize it with `myList`
4. (Sanity Check) Output the shape and the type of `myArray`

[5]:

```
# Task1: Create a simple numpy array
########################
## YOUR CODE STARTS HERE
########################

myList = [-34, -5, 0, -1, 1.5]
print("The length and type of this array is ", len(myList), " and ",␣
 ↪type(myList))
myArray = np.array(myList)
print("Ths shape and type of this numpy array is ", myArray.shape, " and ",␣
 ↪type(myList))
```

```
########################
## YOUR CODE ENDS HERE
########################
```

```
The length and type of this array is  5  and  <class 'list'>
Ths shape and type of this numpy array is  (5,)  and  <class 'list'>
```

0.1.2 Task 2: Access and modify elements in your numpy array

Now you have your `myArray`. How should we access elements in this numpy array?

Recall from last lecture, we can access the elements in a list using square brackets `[]`. Can we use that for numpy arrays

```
# Let me use the same list from the previous example
num_list = [1, 2, 3]

# To access the first element in the list
num1 = num_list[0]

# Sanity Check
print(num1)              ------>  1

# I want to change the first element from 1 to 5
num_list[0] = 5

# Sanity Check
print(num_list)            ------> [5, 2, 3]
```

Slicing: Similar to Python lists, numpy arrays can be sliced.

```
# Slicing in python list

print(num_list[1:])     ------>  [2, 3]

print(num_list[-1:])    ------>  [3]

print(num_list[0:1])    ------>  [5]
```

Here is your task:

1. Access your numpy array `myArray` using index number **0, 3, and -2**
2. Change the first number (the 0th element) in `myArray` to **3**
3. (Sanity Check) Print out your numpy array
4. Declare and initialize two variables `m = 2` and `n = 4`
5. Use slicing to access the element in between `m` and `n`
6. (Sanity Check) Try to explain this to yourself. Does the result make sense?
7. (Challenge) What if we change `n = -1` ?

[6]:

```
# Task2: Access and modify elements
########################
## YOUR CODE STARTS HERE
########################
print(myArray[0])
print(myArray[3])
print(myArray[-2])

myArray[0] = 3

print(myArray)

m, n = 2, 4

print(myArray[m + 1:n])

'''
    myArray[-1] would refer to the last element of the array, so slicing up to␣
 ↪-1, would basically
    include all the elements from m+1 to the second last element in the array.

'''



########################
## YOUR CODE ENDS HERE
########################
```

```
-34.0
-1.0
-1.0
[ 3.  -5.   0.  -1.   1.5]
[-1.]
```

[6]:

```
' \n    myArray[-1] would refer to the last element of the array, so slicing up
to -1, would basically\n    include all the elements from m+1 to the second last
element in the array.\n\n'
```

0.1.3 Pop Quiz 1!

What command should we use to output the dimensions (length) of a python list?

A `.shape`

B `.len`

C `.shape` and `.len`

What command should we use to output the dimensions (as tuples) of a numpy array?

A `.shape`

B `.len`

C `.shape` and `.len`

[7]:

```
# Cast the numbers to "chr" to see answers
print("Pop Quiz 1 answers:", chr(66), " , ", chr(65))
```

```
Pop Quiz 1 answers: B  ,  A
```

0.1.4 Task 3: Manipulating 2D numpy array

We provide you the code to create a 2D numpy array.

Here is your task:

1. (Sanity Check) Print out the 2D numpy array `myArray_2D` and its dimensions
2. Access elements `2` and `5` from `nested_lists` with list indexing
3. Access elements `2` and `5` from `myArray_2D` with numpy array indexing.
4. Increment all elements in the `nested_lists` by 1
5. Increment all elements in the `myArray_2D` by 1
6. Which one is easier?
7. (Challenge) Use slicing to access the first row of `myArray_2D`? How about the second column?

[20]:

```
# We have created these variables
nested_lists = [[2, 3], [4, 5]]
myArray_2D = np.array(nested_lists)

# Task3: Manipulating 2D numpy array
########################
## YOUR CODE STARTS HERE
########################

print(nested_lists[0][0])
print(nested_lists[1][1])

print(myArray_2D[0][0])
print(myArray_2D[1][1])

for i in range(0,2):
    for j in range(0,2):
        nested_lists[i][j] += 1
print(nested_lists)

myArray_2D += 1
print(myArray_2D)

''' Definitely numpy array was easier '''

print("slicing")
print(myArray_2D[0, :])
print(myArray_2D[:, 1])


########################
## YOUR CODE ENDS HERE
########################
```

```
2
5
2
5
[[3, 4], [5, 6]]
[[3 4]
 [5 6]]
slicing
[3 4]
[4 6]
```

0.1.5 Numpy Challenge Question ( )

Given an array of integers [a1, a2, a3, …, an], how to output [a2 - a1, a3 - a2, a4 - a3, …, an - an-1]

Example:

```
a = [ 1, 2, 4, 7 ]
```

```
Output = [ 1, 2, 4 ]
```

Hint

[21]:

```
int_array = [5, 3, 9, 99, -1]
# Numpy challenge question
########################
## YOUR CODE STARTS HERE
########################

myArray2 = np.array(int_array)

myArray2 = myArray2[1:] - myArray2[0:-1]

print(myArray2)


########################
## YOUR CODE ENDS HERE
########################
```

```
[  -2    6   90 -100]
```

0.1.6 Task 4: Understand some of the common functions Python programmers use!

Example 1: Evenly spaced values within a given interval.

```
# Create an array with values
even_spaced_array = np.arange(7) ------> [0, 1, 2, 3, 4, 5, 6]
```

[24]:

```
#Task4: Understanding np.arange
########################
## YOUR CODE STARTS HERE
########################

# Create a list of numbers from 1 to 6 (including 6)
even_spaced_array = np.arange(7)
print(even_spaced_array)

even_spaced_list = [i for i in range(0,7)]
print(even_spaced_list)

########################
## YOUR CODE ENDS HERE
########################
```

```
[0 1 2 3 4 5 6]
[0, 1, 2, 3, 4, 5, 6]
```

0.2 Matplotlib

Matplotlib is a plotting library. In this section give a brief introduction to the `matplotlib.pyplot` module, which provides a plotting system similar to that of MATLAB.

First, let's import the pyplot module from matplotlib package.

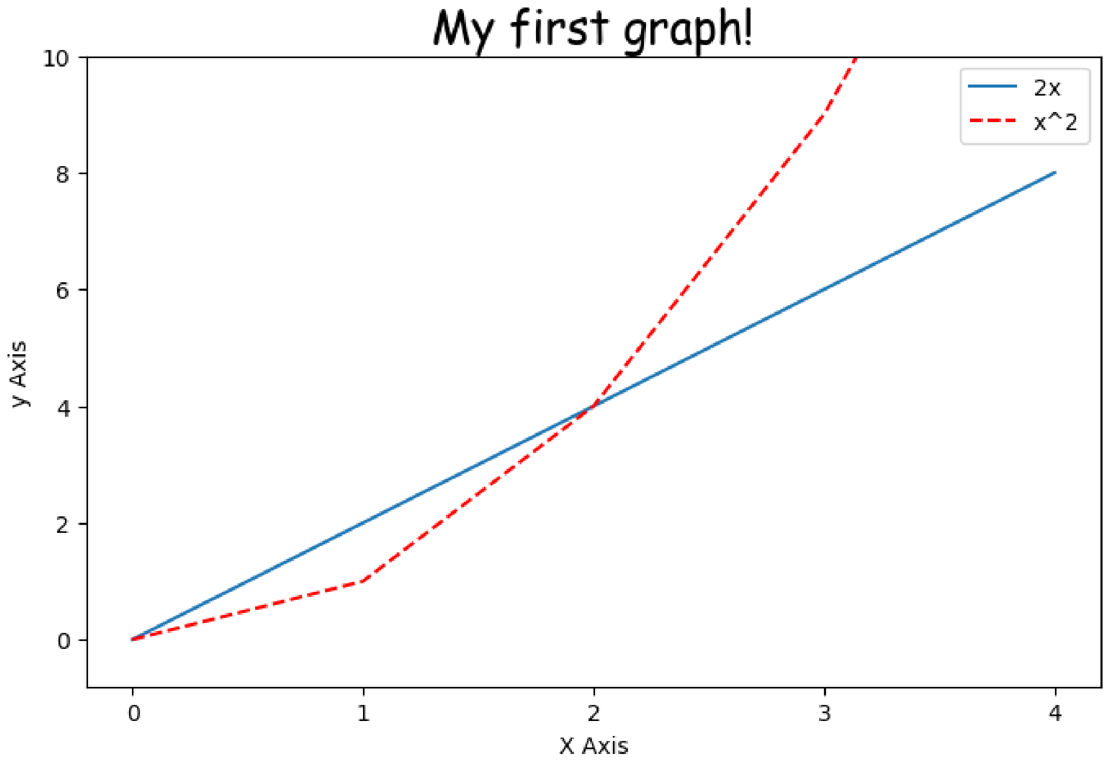
```
import package_name.module_name as alias
```

[25]:

```
# Import matplotlib
########################
## YOUR CODE STARTS HERE
########################

# # Import the pyplot module from matplotlib package (package_name = matplotlib.
 ↪pyplot) using the alias "plt"

import matplotlib.pyplot as plt

########################
## YOUR CODE ENDS HERE
########################
```

[26]:

```
x = [0, 1, 2, 3, 4]
y = [0, 2, 4, 6, 8]

# plt.figure
plt.figure(figsize = (8, 5))

# plot y = 2x
plt.plot(x, y, label = '2x')

# plot y = x^2
plt.plot(x, np.array(x) ** 2, color = 'red', linestyle = '--', label = 'x^2')

plt.title("My first graph!", fontdict={'fontname' : 'Comic Sans MS', 'fontsize':
  ↪ 20})

plt.xlabel('X Axis')
plt.ylabel('y Axis')

plt.xticks([0, 1, 2, 3, 4])


plt.legend()
plt.ylim(top=10)
plt.show()
```



0.2.1 Pop Quiz 2!

Ben wants to plot only the data points with matplotlib. Which function should he use?

A `plt.plot()`

B `plt.scatter()`

C `plt.pie()`

D `plt.bar()`

Bruce wants to use green triangular markers (pointing left) for his graph. How should he set the `marker=` ?

This webpage might be helpful

A `marker = 'g<'`

B `marker = 'r<'`

C `marker = 'g^'`

0.2.2 Task 5: Create your first matplotlib graph (Big-O complexity chart)

What is a Big-O complexity chart?

We want to plot the functions below

- y1(n) = O(1)
- y2(n) = O(log2 n)
- y3(n) = O(n)
- y4(n) = O(n * log2 n)
- y5(n) = O(n2)
- y6(n) = O(2n)
- y7(n) = O(n!)

Hints:

1. In Python, you can express power using `**`. For example, 2 ** 3 = 8 (23 = 8)
2. For factorial, you can write a function that calculates that or use the `factorial()` in the `math` library. And Yes, you need to import the math library :)

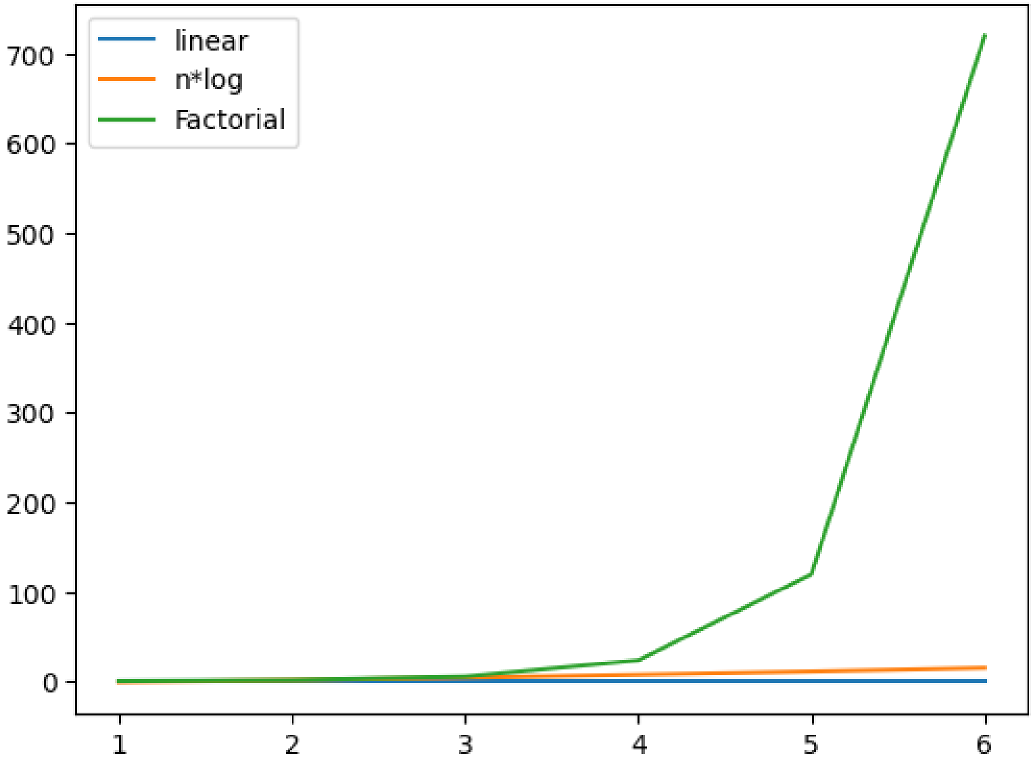
[31]:

```
# Import math
import math

# X_axis
x = np.arange(1,7,1) # start, stop, step

# y_axis
y1 = np.repeat(1, 6)
y2 = np.log2(x)
```

```

########################
## YOUR CODE STARTS HERE
########################

# Sanity Check
print("y1: ", y1)
print("y2: ", y2)

# Generate y3-y7
y3 = x
y4 = x * np.log2(x)
y5 = x ** 2
y6 = 2 ** x
y7 = [math.factorial(i) for i in x]


########################
## YOUR CODE ENDS HERE
########################

# Plot y1, y4, and y7 on the x-y plane
########################
## YOUR CODE STARTS HERE
########################

# Plot y1, y4, an y7
plt.plot(x, y1, label="linear")
plt.plot(x,y4, label="n*log")
plt.plot(x,y7, label="Factorial")
plt.legend()


########################
## YOUR CODE ENDS HERE
########################
```

```
y1:  [1 1 1 1 1 1]
y2:  [0.         1.         1.5849625  2.         2.32192809 2.5849625 ]
```

```
[31]: <matplotlib.legend.Legend at 0x126ff5f40>
```



0.2.3 Task 6 (Homework) Real World Data:

Adapted from “matplotlib_tutorial” by Keith Galli

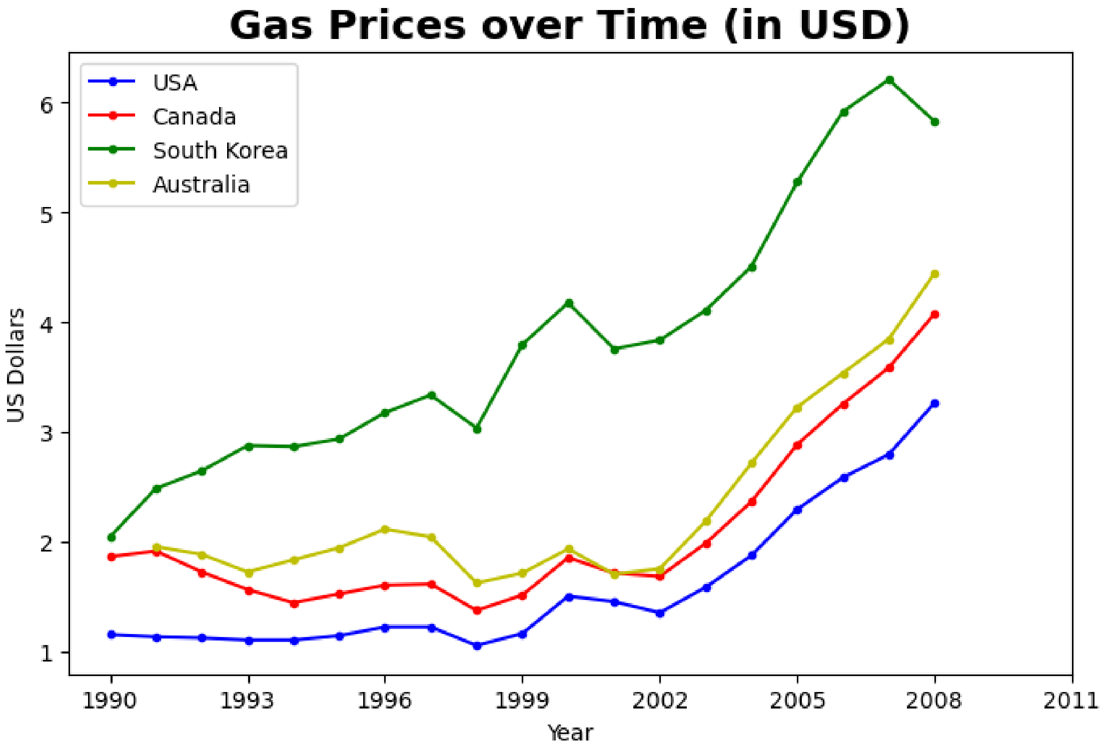
[32]:
```
# Load libraries
import numpy as np # You should understand this line
import matplotlib.pyplot as plt # You should understand this line
import pandas as pd
```

[37]:
```
# mount Google Drive
# This code connects this jupyter notebook with all the files in your google␣
  ↪drive
# from google.colab import drive
# drive.mount('/content/gdrive')
```

[41]:
```
gas = pd.read_csv('gas_prices.csv') # Don't worry about this line, we will␣
  ↪learn it next lecture
# If you are curious:
# 'gas_price.csv' is a "csv" file (you don't have to understand csv).
# 'read_csv' function reads in the data so now Python can understand.
# 'pd' is the alias for the "pandas" library.
# 'gas' is just the variable that stores the data.
```

```
plt.figure(figsize=(8,5)) # Let's define the size of our figure

plt.title('Gas Prices over Time (in USD)', fontdict={'fontweight':'bold',␣
 ↪'fontsize': 18}) # This should seem familiar!

# Initialize X and ys
X_year = gas['Year']
y_USA = gas['USA']
y_Canada = gas['Canada']
y_SouthKorea = gas['South Korea']
y_Australia = gas['Australia']

# Plot!
########################
## YOUR CODE STARTS HERE
########################

# See if you can use a for loop to prevent repetitive codes
# Use 'b.-' for USA, 'r.-' for Cananda, 'g.-' for South Korea, and 'y.-' for␣
 ↪Australia
# Don't forget to label each plot (later used by legend())

countries = {
    "USA": "b.-",
    "Canada": "r.-",
    "South Korea": "g.-",
    "Australia": "y.-"
}

# Add x and y axis labels and call legend()
# Your x-axis should be 'Year', and y-axis should be 'US Dollars'

for country, style in countries.items():
    plt.plot(gas['Year'], gas[country], style, label=country)

# Add x and y axis labels and call legend()
# Your x-axis should be 'Year', and y-axis should be 'US Dollars'


# Add x and y axis labels and call legend()
plt.xlabel('Year')
plt.ylabel('US Dollars')
plt.legend()

########################
## YOUR CODE ENDS HERE
```

```
#########################


plt.xticks(gas.Year[::3].tolist()+[2011])


plt.show()
```



0.2.4 Write Your Report Here

```
[34]: '''
      Included in this notebook is basically a tutorial on using the numpy library,␣
       ↪slicing, and using the matplotlib library

      No real difficulties were encountered

      '''
```