Lab_3_Andrade

January 30, 2024

31 January 2024 #

Lab 3 Assignment - CS 4315

Doug Andrade

1. Create a numpy array that is contains $[1, 2, 3, 4]$ and set A to that.

[1]:
```
# import the Numpy module
import numpy as np

# Create a 1D ndarray as object 'A'
A = np.arange(start = 1, stop = 5)

# Print and check the new 1D ndarray A
print('The Numpy array "A" = %s' % (A))
```

```
The Numpy array "A" = [1 2 3 4]
```

2. Reshape A to a 2 by 2 array and use numpy to print the shape of A.

[2]:
```
# Resize the 1D ndarray to a 2x2 ndarry matrix
A = np.resize(a = A, new_shape = (2, 2))

# Print and check the new matrix A
print('Matrix A =\n%s' % A)

# Print and check the resized ndarray shape
print('\nThe new shape of object A is "%s".' % str(A.shape))
```

```
Matrix A =
[[1 2]
 [3 4]]

The new shape of object A is "(2, 2)".
```

3. Multiply A by 2, then add 1 to A, and then set the result to variable B.

[3]:
```
# Multiply each element in A by 2, and then add 1 to the product of each element
B = (A * 2) + 1
```

```
# Print and check the new matrix B
print('Matrix B =\n%s' % B)
```

```
Matrix B =
[[3 5]
 [7 9]]
```

4. Print the second value in the first row of A and print the second row of A.

[4]:
```
# Use indexing to print the 2nd value in the 1st row of matrix A, and␣
  ↪separately the 2nd row of matrix A
print('The 2nd value in the 1st row of A is "%d".\n\nThe second row of A is␣
  ↪"%s".' % (A[0][1], str(A[1])))
```

```
The 2nd value in the 1st row of A is "2".

The second row of A is "[3 4]".
```

5. Matrix multiply A and B and print the result, setting the result to variable C and printing it.

[5]:
```
# Use matrix multiplication (dot product using "@") to get the new matrix C
C = A @ B

# Print and check the new matrix C
print('The dot product of A @ B is = \n%s' % str(C))
```

```
The dot product of A @ B is =
[[17 23]
 [37 51]]
```

6. Print the transpose of C.

[6]:
```
# Use the transpose operator "T" on matrix C, and print using string formatting
print('The transpose of C is = \n%s' % str(C.T))
```

```
The transpose of C is =
[[17 37]
 [23 51]]
```

7. Print the minimum and maximum of C using numpy functions.

[7]:
```
# Apply the np.min and np.max functions to extract and print the associated␣
  ↪values from matrix C
print('The minimum value in C is "%d". \n\nThe maximum value in C is "%d".' %␣
  ↪(np.min(a = C), np.max(a = C)))
```

```
The minimum value in C is "17".

The maximum value in C is "51".
```

8. Use `numpy` to return a `numpy` array that contains the index for the row with the highest value for each column of C.

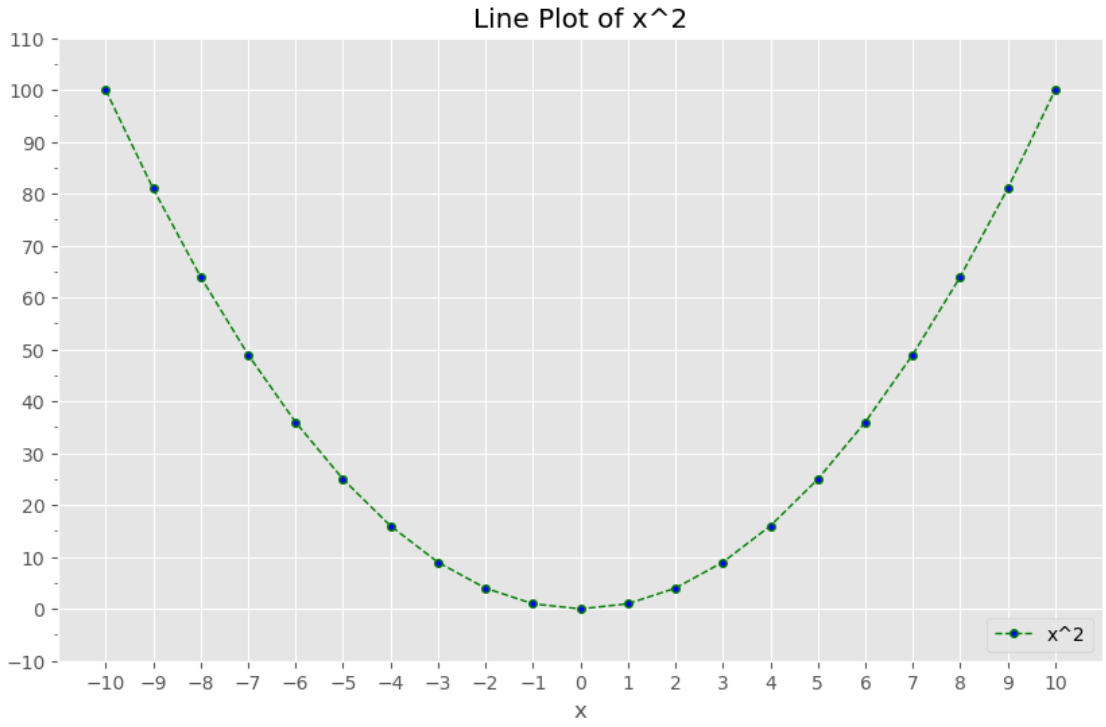
[8]:
```
# Apply the np.argmax functions to extract and print the type and row index for␣
 ↪the column with the greatest value in matrix C
row_output = np.argmax(a = C, axis = 0)
print('The %s of indexes for the rows with the highest value in each column of␣
 ↪C is "%s".' % (type(row_output), row_output))
```

```
The <class 'numpy.ndarray'> of indexes for the rows with the highest value in
each column of C is "[1 1]".
```

9. Use `matplotlib` to create a line plot for the function x^2 for x that are integers from -10 to 10, including -10 and 10.

[9]:
```
# Import matplotlib.pyplot module for the plotting API
import matplotlib.pyplot as plt
# Magic command useful for Jupyter Notebooks to embed plots within the notebook
%matplotlib inline

# Create the array of x integers between and including -10 and 10
x = np.linspace(start = -10, stop = 10, num = 21)

# Create the array of y integers by squaring the x array using vectorization
y = x**2

# Apply R's ggplot grid and background styling
plt.style.use('ggplot')

# Set the chart size 10x6 for ease of visualization
plt.figure(figsize = (10, 6))

# Line plot of x and y, with additional customization - a labeled, dashed green␣
 ↪line
plt.plot(x, y, linestyle = 'dashed', linewidth = 1, c = 'g', marker = 'o',␣
 ↪markersize = 4, markerfacecolor = 'b', label = 'x^2')

# Add a graph title, x-axis label, set x and y-axis tick mark intervals, and␣
 ↪add the legend to the bottom right
plt.title('Line Plot of x^2')
plt.xlabel('x')
plt.xticks(ticks = x, minor = False)
plt.yticks(ticks = np.arange(start = -10, stop = 111, step = 10), minor = False)
plt.yticks(ticks = np.arange(start = -10, stop = 111, step = 5), minor = True)
plt.legend(loc = 4) # no need for a y-axis label since the legend provides the␣
 ↪same information
```

```
# Remove the right and top border (spines) and add a grid for better visualization
plt.gca().spines['right'].set_visible(False)
plt.gca().spines['top'].set_visible(False)
#plt.grid(True) # ggplot's grid looks better
```



10. Use `matplotlib` to create a scatter plot for y of $[1.1, 2.0, 4.5, 7.2]$ and x of $[2.3, 3.7, 4.2, 1.3]$.

[10]:
```
# create a numpy array of of the provided values for x and y
x = np.array([2.3, 3.7, 4.2, 1.3])
y = np.array([1.1, 2.0, 4.5, 7.2])

# Scatter plot of x and y, as green circles of random sizes
plt.scatter(x = x, y = y, c = 'g', s = 500 * np.random.rand(4))

# Add a graph title and corresponding x and y-axis labels
plt.title('Scatter Plot of x and y')
plt.xlabel('x')
plt.ylabel('y')

plt.xticks(ticks = np.arange(start = np.floor(np.min(x) - 1), stop = np.floor(np.max(x) + 3)))
```

```
plt.yticks(ticks = np.arange(start = np.floor(np.min(y) - 1), stop = np.
 ↪floor(np.max(y) + 3)))

# Remove the right and top border (spines) and add a grid for better␣
 ↪visualization
plt.gca().spines['right'].set_visible(False)
plt.gca().spines['top'].set_visible(False)
plt.grid(True)
```