22-matplotlib

November 17, 2020

1 Matplotlib

In this lecture we will talk about how to produce scientific graphs using the python library matplotlib. Matplotlib provides a variety of functions that will allow you to quickly and easily produce a variety of useful, pretty graphs.

Matplotlib is not included by default withpPython. It is a separate python library that is downloaded and installed alongside python. If you installed python using anaconda, you probably already have it.

In order to use matplotlib you need to import it like this

```
import matplotlib.pyplot as plt
```

The name after `as` is just an abbreviation for the library. That means that, whenever we want to use a function from matplotlib, we will prefix it with `plt`.

DISCLAIMER: The graphs we will plot here are highly customizable, and we are far from exhausting every configuration option. We encourage you to check matplotlib's documentation and the gallery of examples to find out more.

1.1 Prelude

The code below is similar for all that comes afterwards, so we will keep it once here to avoid having to copy/paste it everywhere.

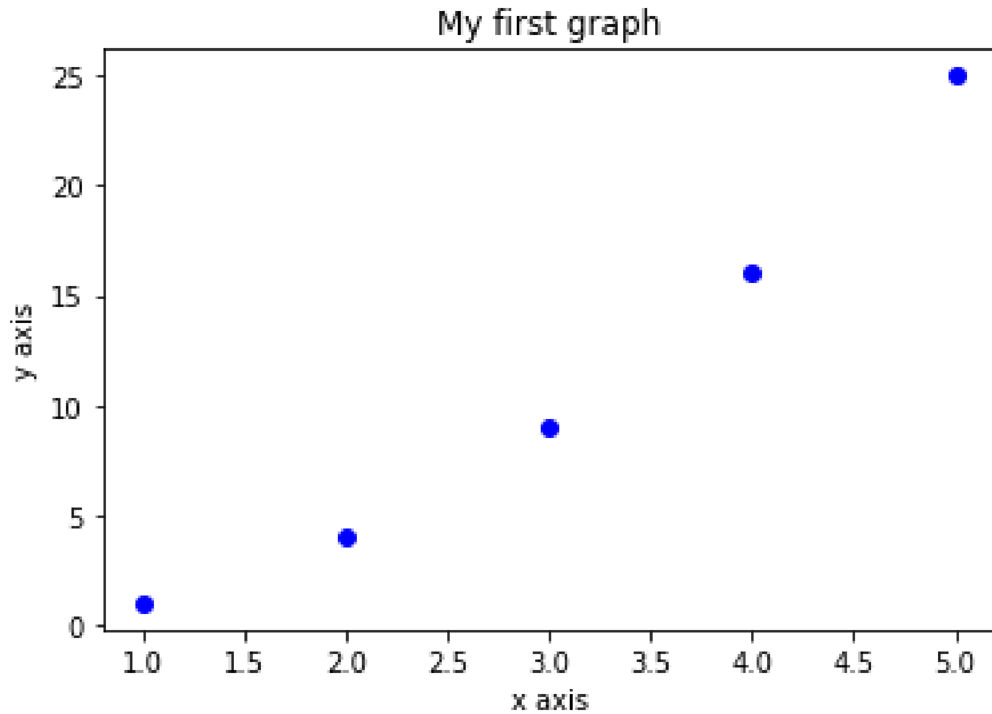
[1]:
```
import matplotlib.pyplot as plt
import math

# Generates a range of floating point numbers
def rangeFloat(low, hi, step):
    res = []
    i = low
    while i < hi:
        res.append(i)
        i += step
    return res
```

1.2 Plot

The `plot` graph simply plots whatever coordinates we pass to it.

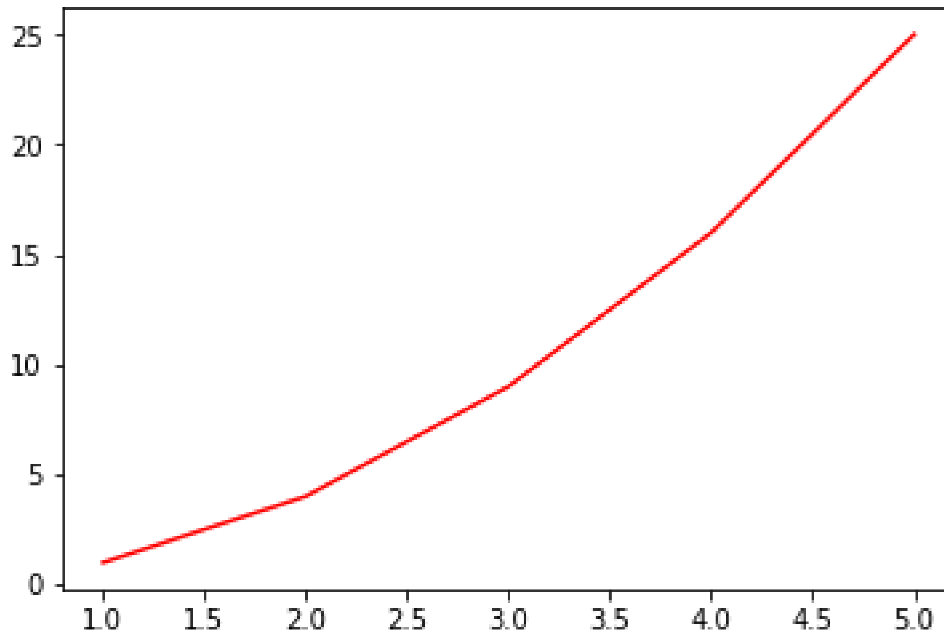
If we pass coordinates separately, it plots the points:

[2]:
```
import matplotlib.pyplot as plt

def drawPoints():
    # Setting the plot's title
    plt.title("My first graph")

    # Setting the label for the x and y axes
    plt.xlabel("x axis")
    plt.ylabel("y axis")

    # Plotting each point separately
    # 'b' stands for blue, and 'o' stands for circle
    plt.plot(1,1,'bo')
    plt.plot(2,4,'bo')
    plt.plot(3,9,'bo')
    plt.plot(4,16,'bo')
    plt.plot(5,25,'bo')

    # Renders the graph on the screen
    plt.show()

drawPoints()
```



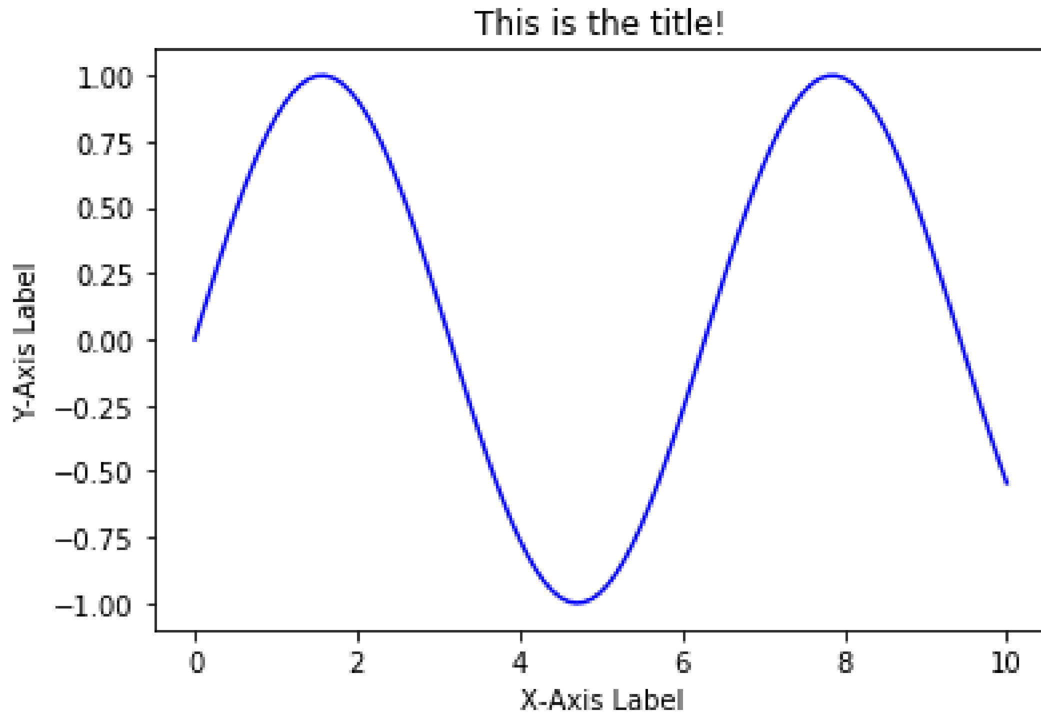
Alternatively, we can pass the x coordinates as a list (first parameter) and the y coordinates as a list (second parameter). In this case, mathplotlib will connect the dots. Observe that the lists must be in the correct order. That means that (x,y) coordinates of the same point need to be at the same position in both lists.

[3]:
```
def drawLine():
    # First parameter: x coordinates
    # Second parameter: y coordinates
    # 'r' stands for red
    plt.plot([1,2,3,4,5],[1,4,9,16,25], 'r')

    plt.show()

drawLine()
```



If we make the coordinates closer, the line will look smoother. The function below plots a graph of the sine function, using x coordinates at every 0.1.

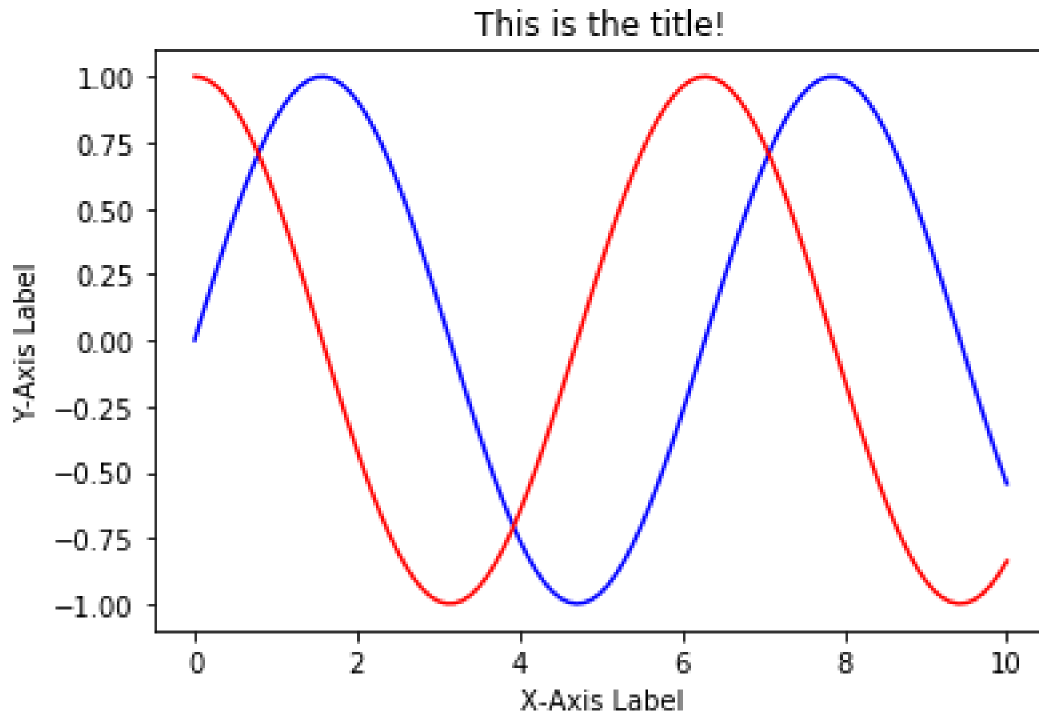
[4]:
```
def drawSineGraph():
    # Generate a list containing all the values from 0 to 10
    # with a step of 0.1.  We'll use these as our x-values.
    x = rangeFloat(0,10,0.1)

    # For each x value, generate the appropriate y-value.
    sin_x = []
    for i in x:
        sin_x += [math.sin(i)]

    # Plot the x,y values with a blue line
    plt.plot(x, sin_x, 'b')
    plt.title("This is the title!")
    plt.xlabel("X-Axis Label")
    plt.ylabel("Y-Axis Label")

    # Show the plot
    plt.show()

drawSineGraph()
```



We can plot more than one graph in one.

[5]:
```
def drawSineCosineGraph():
    # Generate a list containing all the values from 0 to 10
    # with a step of 0.1.  We'll use these as our x-values.
    x = rangeFloat(0,10,0.1)

    # For each x value, generate the appropriate y-value.
    sin_x = []
    cos_x = []
    for i in x:
        sin_x += [math.sin(i)]
        cos_x += [math.cos(i)]

    # Plot sine with a blue line and cosine with a red line
    plt.plot(x, sin_x, 'b')
    plt.plot(x, cos_x, 'r')
    plt.title("This is the title!")
    plt.xlabel("X-Axis Label")
    plt.ylabel("Y-Axis Label")

    # Show the plot
    plt.show()
```

```
drawSineCosineGraph()
```



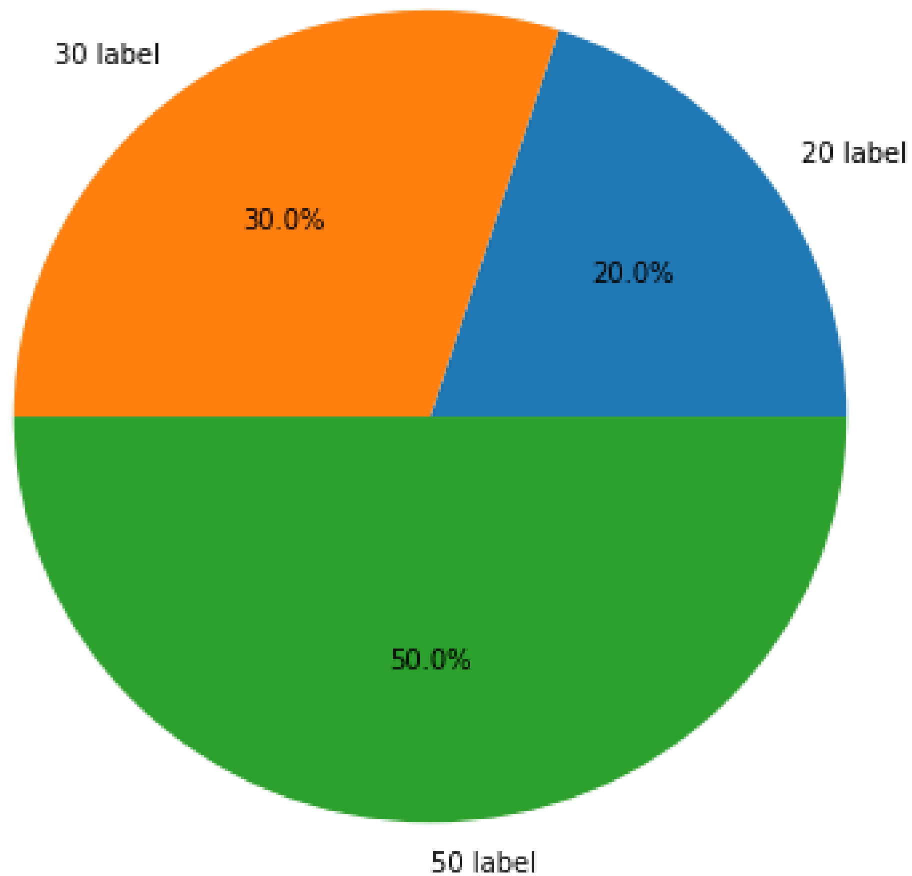
1.3 Pie charts

The `pie` function from matplotlib plots pie charts. It expects a list of values as a parameter, and it will create one pie slice for each one of those values. The whole pie corresponds to the sum of values.

Optionally, we can pass a named parameter to this function to determine the label of each slice. The name of this parameter is `labels`, and the label must be at the same position in the list as its slice.

Another useful optional paramter for the `pie` function is `autopct="%.1f%%"` which writes the percentage of each slice on the pie.

[6]:
```
def drawPie():
    plt.figure(figsize=(7,7)) # Make the chart a perfect square
    plt.title("A Pie Chart")
    plt.pie([20,30,50], labels=["20 label", "30 label", "50 label"], autopct="%.
 ↪1f%%")
    plt.show()

drawPie()
```



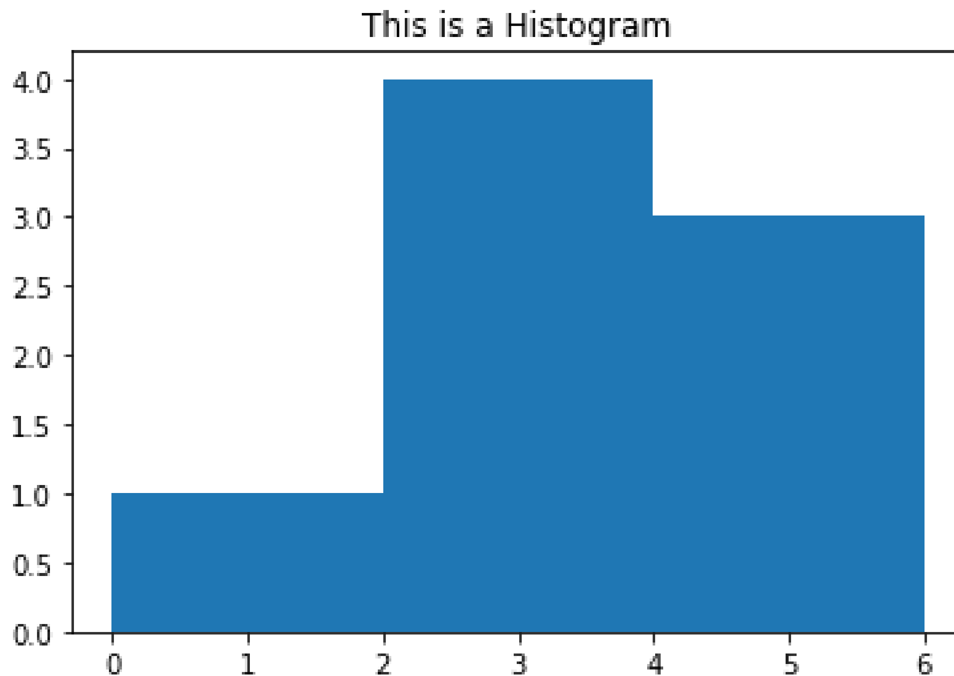
1.4 Histograms

The `hist` function plots histograms. A histogram takes a list of values, organizes them into bins, and then graphs how many items are in each bin. Note that the order of this list of values does not matter.

Let's start with a basic example that takes a small list of numbers and generates a histogram using the bins [0,2) (2 is not included in the bin), [2,4) (4 is not included in the bin), and [4,6]. Observe that the bin list specifies where each bin begins, and where the last bin ends.

[7]:
```
def drawHistogram():
    L = [1, 2, 3, 4, 4, 4, 2, 2]
    plt.title("This is a Histogram")
    plt.hist(L, [0,2,4,6])
```

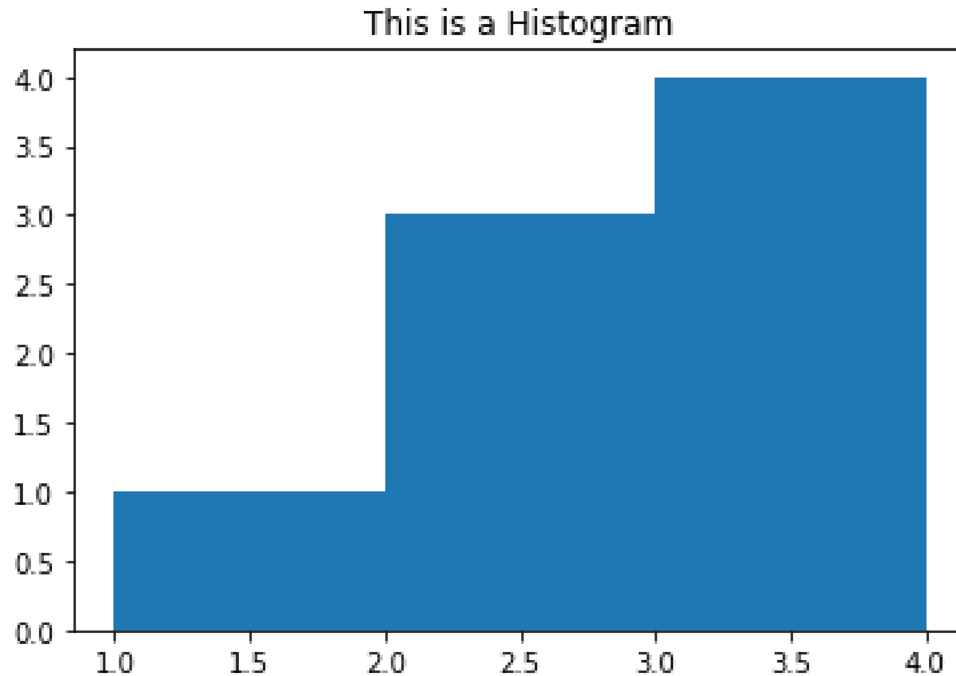
```
    plt.show()

drawHistogram()
```



As you can see from the output, there is one number in the range [0,2), four numbers in the range [2,4), and three numbers in the range [4,6].

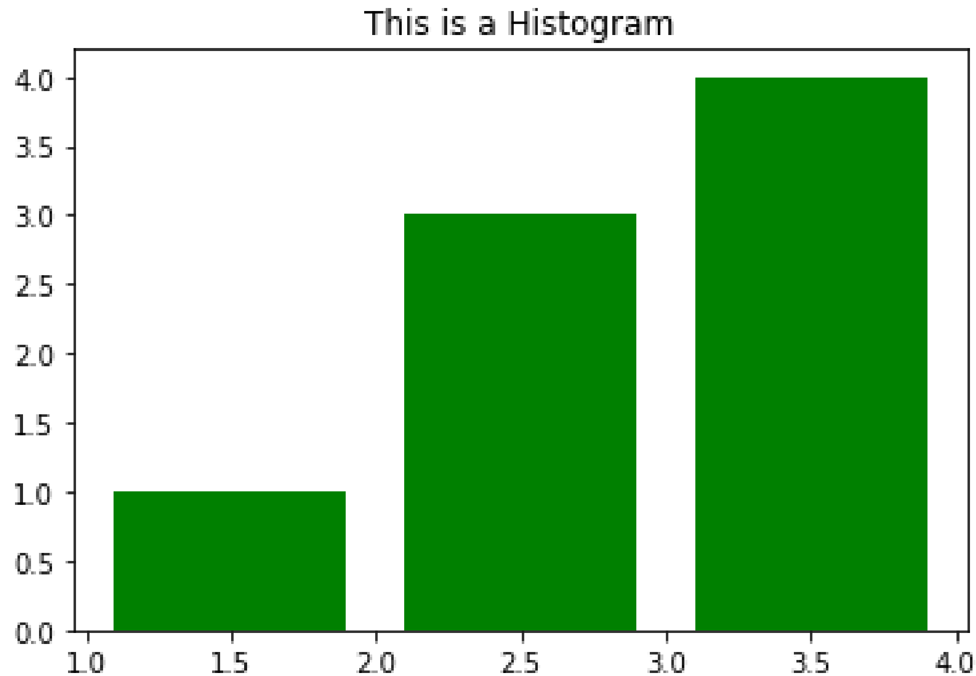
Instead of including a list indicating the start and end point of the bins, you can also just specify the total number of bins you want and matplotlib will automatically generate the bin ranges. Matplotlib generates the bins by taking the lowest and highest values in the list, and splitting this interval into the number of bins requested.

[8]:
```
def drawHistogram():
    L = [1, 2, 3, 4, 4, 4, 2, 2]
    plt.title("This is a Histogram")
    plt.hist(L, 3) # We want three bins.
    plt.show()

drawHistogram()
```



There are a number of simple arguments you can pass to `hist` in order to improve the appearance of the histogram. From the example below, can you figure out what `rwidth` and `color` do? Change the values and experiment to see what happens to the graph.

[9]:
```
def drawHistogram():
    L = [1, 2, 3, 4, 4, 4, 2, 2]
    plt.title("This is a Histogram")
    plt.hist(L, 3, rwidth=0.8, color="g")
    plt.show()

drawHistogram()
```

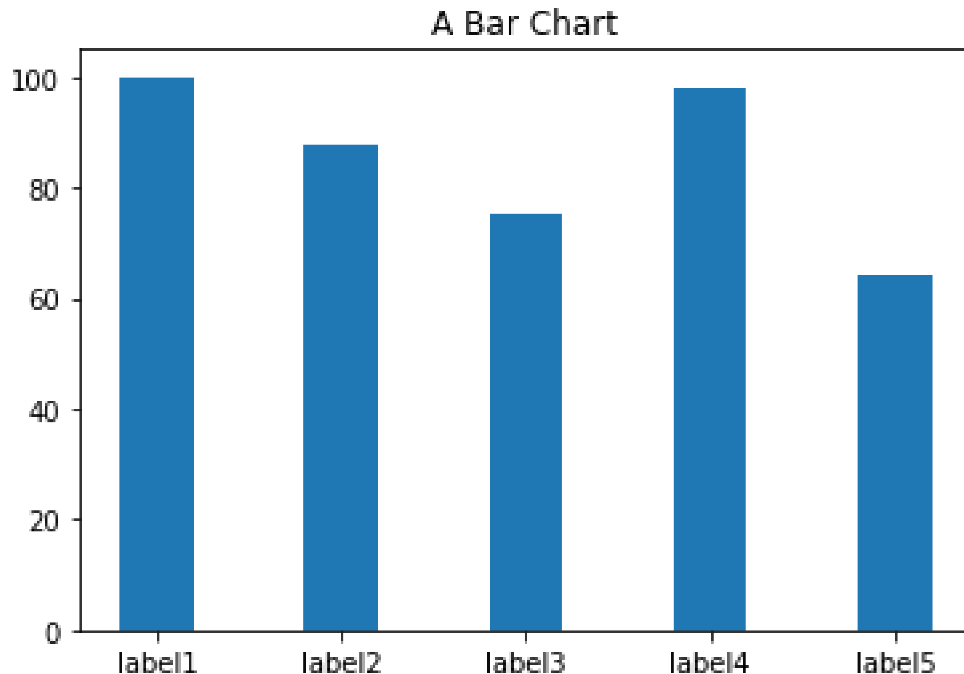


1.5 Bar charts

The function `bar` is used to plot bar charts. A bar chart looks a lot like a histogram, but the difference is that the "bins" on the x-axis may be completely unrelated categories, while in histograms these are continuous values.

The `bar` function takes as paramters a list of the positions of the bars on the x-axis, and a list of the heights of each bar. Optionally you can define the named parameter `labels` as the list of labels for the bars to be placed on the x-axis.

[10]:
```
def drawBarChart():
    plt.title("A Bar Chart")
    # Position of bars, height of bars
    plt.bar([0,2,4,6,8], [100,88,75,98,64], tick_label=["label1", "label2",␣
 ↪"label3", "label4", "label5"])
    plt.show()

drawBarChart()
```



1.6 Subplots

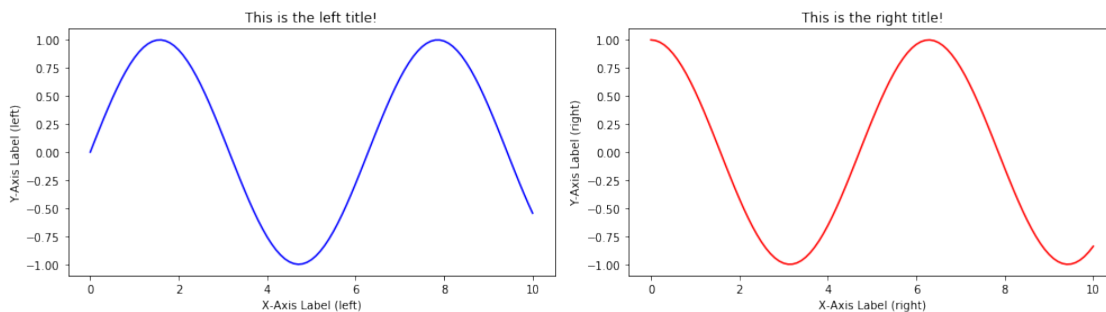
Sometimes you want to produce multiple different plots and display all of them at the same time, but not on the same set of axes. We can accomplish this with subplots.

The `subplot` function can be used to allow multiple plots to be displayed in the same figure. It takes as arguments three numbers: the number of rows, the number of columns, and which subplot you want to activate. For example, `plt.subplot(121)` says to arrange the subplots in a grid with 1 row and two columns, and then activate the first subplot. (In the case, the left one.) `plt.subplot(122)` say to arrange the subplots the same way, but activate the 2nd subplot.

Consider the following example that graphs both sine and cosine in the same figure, but on different subplots.

[11]:
```
def drawSimpleSubplots():
    # Generate a list containing all the values from 0 to 10
    # with a step of 0.1.  We'll use these as our x-values.
    x = rangeFloat(0,10,0.1)

    # For each x value, generate the appropriate y-value.
    sin_x = []
    cos_x = []
    for i in x:
        sin_x += [math.sin(i)]
        cos_x += [math.cos(i)]
```

```
    # Set the size of the plot
    plt.figure(figsize=(14,4))

    # Configure the subplot. The layout as a whole is 1 rows with 2 columns,
    # and we are currently plotting the first location.
    plt.subplot(121)

    # Make the first plot
    plt.title("This is the left title!")
    plt.xlabel("X-Axis Label (left)")
    plt.ylabel("Y-Axis Label (left)")
    plt.plot(x,sin_x,'b')

    # Switch the subplot to the 2nd subplot
    plt.subplot(122)

    # Make the second plot
    plt.title("This is the right title!")
    plt.xlabel("X-Axis Label (right)")
    plt.ylabel("Y-Axis Label (right)")
    plt.plot(x,cos_x,'r')

    # This help avoid overlap between the two plots
    plt.tight_layout()

    # Show the final, combined plot
    plt.show()

drawSimpleSubplots()
```



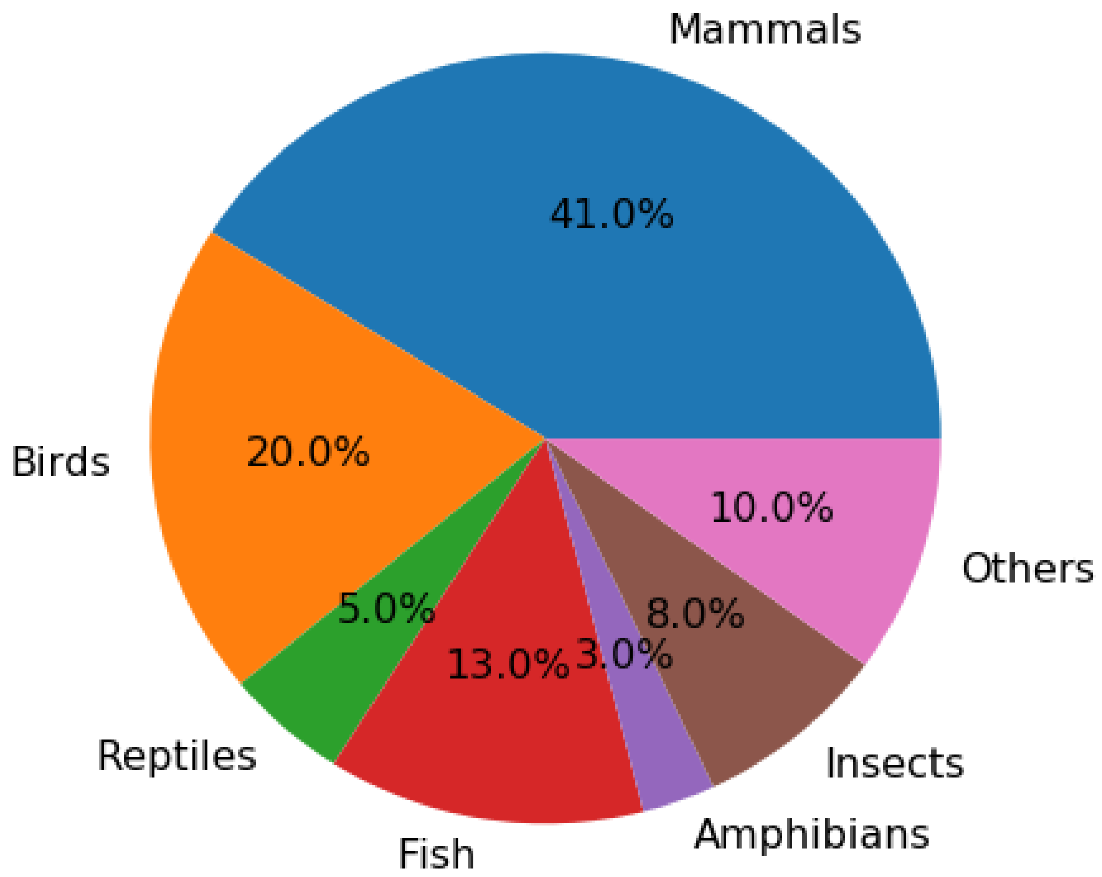
1.7 Exercise

Use the file https://web2.qatar.cmu.edu/cs/15110/resources/zoo.csv containing information about animals to build a pie chart showing the division of animals per class. The class is indi-

cated in the type column by a number ranging from 1 to 7. The mapping of numbers to classes is: 1. Mammals 2. Birds 3. Reptiles 4. Fish 5. Amphibians 6. Insects 7. Others

[12]:
```
def classPieChart():
    file = open("18-files/zoo.csv")

    traits = ['hair',
          'feathers',
          'eggs',
          'milk',
          'airborne',
          'aquatic',
          'predator',
          'toothed',
          'backbone',
          'breathes',
          'venomous',
          'fins',
          'legs', # Numeric {0,2,4,5,6,8}
          'tail',
          'domestic',
          'catsize',
          'type'] # Numeric [1..7]

    typeNames = ["Mammals", "Birds", "Reptiles", "Fish", "Amphibians",␣
 ↪"Insects", "Others"]

    # Reads the csv file as a dictionary of dictionaries
    animals = {}
    for line in file:
        if not line.startswith('#'):
            vals = line.strip().split(",")
            animal = vals[0]

            # Builds the internal dictionary for each animal
            d = {}
            for i in range(len(vals[1:])):
                v = vals[i+1]
                # Number of legs is numeric
                if i == 12:
                    v = int(v)
                # Save the type with the proper name instead of a code
                elif i == 16:
                    typeIdx = int(v) - 1
                    v = typeNames[typeIdx]
                # All else is boolean
                else:
```

```
                v = bool(int(v))

            d[traits[i]] = v

        animals[animal] = d

    # Collects the type column as a list
    types = []
    for animal in animals:
        types += [animals[animal]['type']]

    # Counts how many animals of each type
    mammals = types.count('Mammals')
    birds = types.count('Birds')
    reptiles = types.count('Reptiles')
    fish = types.count('Fish')
    amphibians = types.count('Amphibians')
    insects = types.count('Insects')
    others = types.count('Others')

    plt.rcParams.update({'font.size': 16}) # Makes font bigger
    plt.figure(figsize=(7,7))
    plt.title("Representation of classes")
    plt.pie([mammals, birds, reptiles, fish, amphibians, insects, others],␣
↪labels=typeNames, autopct="%.1f%%")
    plt.show()


classPieChart()
```


Representation of classes
Mammals
41.0%
Birds
20.0%
10.0%
Others
5.0%
8.0%
13.0%
3.0%
Reptiles
Insects
Fish
Amphibians