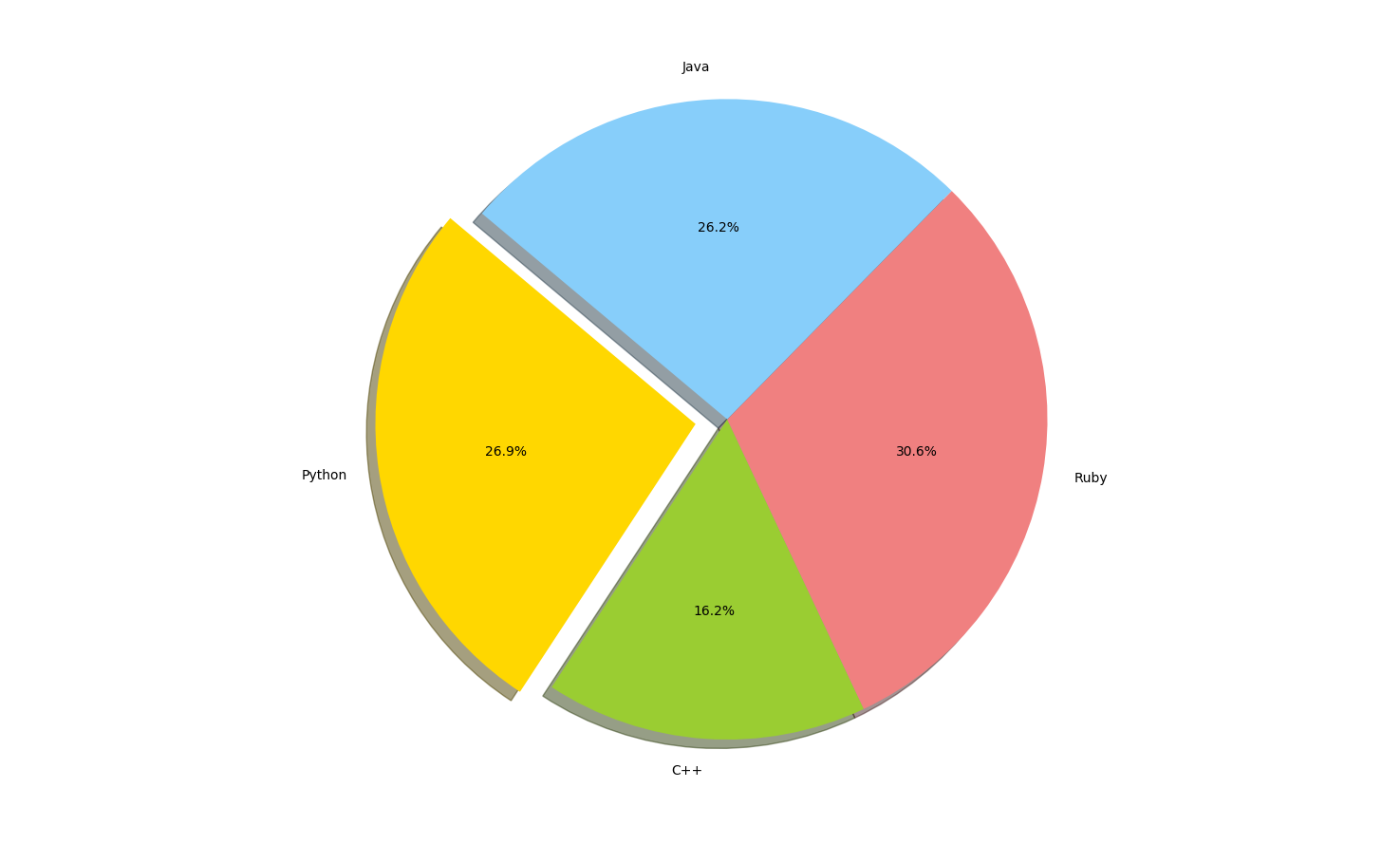
Matplotlib

**Matplotlib pie chart**  
The code below creates a pie chart:

|  |
| --- |
| **import** matplotlib.pyplot **as** plt    *# Data to plot*  labels = ['Python', 'C++', 'Ruby', 'Java']  sizes = [215, 130, 245, 210]  colors = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue']  explode = (0.1, 0, 0, 0) *# explode 1st slice*    *# Plot*  plt.pie(sizes, explode=explode, labels=labels, colors=colors,  autopct='%1.1f%%', shadow=True, startangle=140)    plt.axis('equal')  plt.show() |



To add a legend use the plt.legend() function:

|  |
| --- |
| **import** matplotlib.pyplot **as** plt    labels = ['Cookies', 'Jellybean', 'Milkshake', 'Cheesecake']  sizes = [38.4, 40.6, 20.7, 10.3]  colors = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral']  patches, texts = plt.pie(sizes, colors=colors, shadow=True, startangle=90)  plt.legend(patches, labels, loc="best")  plt.axis('equal')  plt.tight\_layout()  plt.show() |

plt.title('Pakistan Population Province Wise')

Matplotlib Bar chart

**Bar chart code**  
The code below creates a bar chart:

**import** matplotlib.pyplot **as** plt; plt.rcdefaults()

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

objects = ('Python', 'C++', 'Java', 'Perl', 'Scala', 'Lisp')

y\_pos = np.arange(len(objects))

performance = [10,8,6,4,2,1]

plt.bar(y\_pos, performance, align='center', alpha=0.5)

plt.xticks(y\_pos, objects)

plt.ylabel('Usage')

plt.title('Programming language usage')

plt.show()

Horizontal

**import** matplotlib.pyplot **as** plt; plt.rcdefaults()

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

objects = ('Python', 'C++', 'Java', 'Perl', 'Scala', 'Lisp')

y\_pos = np.arange(len(objects))

performance = [10,8,6,4,2,1]

plt.barh(y\_pos, performance, align='center', alpha=0.5)

plt.yticks(y\_pos, objects)

plt.xlabel('Usage')

plt.title('Programming language usage')

plt.show()

**More on bar charts**

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

*# data to plot*

n\_groups = 4

means\_frank = (90, 55, 40, 65)

means\_guido = (85, 62, 54, 20)

*# create plot*

fig, ax = plt.subplots()

index = np.arange(n\_groups)

bar\_width = 0.35

opacity = 0.8

rects1 = plt.bar(index, means\_frank, bar\_width,

alpha=opacity,

color='b',

label='Frank')

rects2 = plt.bar(index + bar\_width, means\_guido, bar\_width,

alpha=opacity,

color='g',

label='Guido')

plt.xlabel('Person')

plt.ylabel('Scores')

plt.title('Scores by person')

plt.xticks(index + bar\_width, ('A', 'B', 'C', 'D'))

plt.legend()

plt.tight\_layout()

plt.show()

# Matplotlib Line chart

**Line chart example**  
The example below will create a line chart.

|  |
| --- |
| **from** pylab **import** \*  t = arange(0.0, 2.0, 0.01)  s = sin(2.5\*pi\*t)  plot(t, s)    xlabel('time (s)')  ylabel('voltage (mV)')  title('Sine Wave')  grid(True)  show() |

The lines:

|  |
| --- |
| **from** pylab **import** \*    t = arange(0.0, 2.0, 0.01)  s = sin(2.5\*pi\*t) |

simply define the data to be plotted.

|  |
| --- |
| **from** pylab **import** \*    t = arange(0.0, 2.0, 0.01)  s = sin(2.5\*pi\*t)  plot(t, s)  show() |

plots the chart.  The other statements are very straightforward: statements xlabel() sets the x-axis text, ylabel() sets the y-axis text, title() sets the chart title and grid(True) simply turns on the grid.

If you want to save the plot to the disk, call the statement:

|  |
| --- |
| savefig("line\_chart.png") |

**Plot a custom Line Chart**

If you want to plot using an array (list), you can execute this script:

|  |
| --- |
| **from** pylab **import** \*    t = arange(0.0, 20.0, 1)  s = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]  plot(t, s)    xlabel('Item (s)')  ylabel('Value')  title('Python Line Chart: Plotting numbers')  grid(True)  show() |

The statement:

|  |
| --- |
| t = arange(0.0, 20.0, 1) |

defines start from 0, plot 20 items (length of our array) with steps of 1.

**Multiple plots**

If you want to plot multiple lines in one chart, simply call the plot() function multiple times. An example:

|  |
| --- |
| **from** pylab **import** \*    t = arange(0.0, 20.0, 1)  s = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]  s2 = [4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23]  plot(t, s)  plot(t, s2)    xlabel('Item (s)')  ylabel('Value')  title('Python Line Chart: Plotting numbers')  grid(True)  show() |

python line chart multiple

In case you want to plot them in different views in the same window you can use this:

|  |
| --- |
| **import** matplotlib.pyplot **as** plt  **from** pylab **import** \*    t = arange(0.0, 20.0, 1)  s = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]  s2 = [4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23]    plt.subplot(2, 1, 1)  plt.plot(t, s)  plt.ylabel('Value')  plt.title('First chart')  plt.grid(True)    plt.subplot(2, 1, 2)  plt.plot(t, s2)  plt.xlabel('Item (s)')  plt.ylabel('Value')  plt.title('Second chart')  plt.grid(True)  plt.show() |

Matplotlib scatterplot

Matplot has a built-in function to create *scatterplots* called scatter(). A scatter plot is a type of plot that shows the data as a collection of points. The position of a point depends on its two-dimensional value, where each value is a position on either the horizontal or vertical dimension.

**Scatterplot example**  
Example:

|  |
| --- |
| **import** numpy **as** np  **import** matplotlib.pyplot **as** plt    *# Create data*  N = 500  x = np.random.rand(N)  y = np.random.rand(N)  colors = (0,0,0)  area = np.pi\*3    *# Plot*  plt.scatter(x, y, s=area, c=colors, alpha=0.5)  plt.title('Scatter plot pythonspot.com')  plt.xlabel('x')  plt.ylabel('y')  plt.show() |

**Scatter plot with groups**  
Data can be classified in several groups. The code below demonstrates that:

|  |
| --- |
| **import** numpy **as** np  **import** matplotlib.pyplot **as** plt    *# Create data*  N = 60  g1 = (0.6 + 0.6 \* np.random.rand(N), np.random.rand(N))  g2 = (0.4+0.3 \* np.random.rand(N), 0.5\*np.random.rand(N))  g3 = (0.3\*np.random.rand(N),0.3\*np.random.rand(N))    data = (g1, g2, g3)  colors = ("red", "green", "blue")  groups = ("coffee", "tea", "water")    *# Create plot*  fig = plt.figure()  ax = fig.add\_subplot(1, 1, 1, axisbg="1.0")    **for** data, color, group **in** zip(data, colors, groups):  x, y = data  ax.scatter(x, y, alpha=0.8, c=color, edgecolors='none', s=30, label=group)    plt.title('Matplot scatter plot')  plt.legend(loc=2)  plt.show() |

Matplotlib Histogram

**Matplotlib** can be used to create histograms. A histogram shows the frequency on the vertical axis and the horizontal axis is another dimension. Usually it has bins, where every bin has a minimum and maximum value. Each bin also has a frequency between x and infinite.

**import** numpy **as** np

**import** matplotlib.mlab **as** mlab

**import** matplotlib.pyplot **as** plt

x = [21,22,23,4,5,6,77,8,9,10,31,32,33,34,35,36,37,18,49,50,100]

num\_bins = 5

n, bins, patches = plt.hist(x, num\_bins, facecolor='blue', alpha=0.5)

plt.show()

**matplotlib python histogram**  
Many things can be added to a histogram such as a fit line, labels and so on. The code below creates a more advanced histogram.

|  |
| --- |
| *#!/usr/bin/env python*    **import** numpy **as** np  **import** matplotlib.mlab **as** mlab  **import** matplotlib.pyplot **as** plt      *# example data*  mu = 100 *# mean of distribution*  sigma = 15 *# standard deviation of distribution*  x = mu + sigma \* np.random.randn(10000)    num\_bins = 20  *# the histogram of the data*  n, bins, patches = plt.hist(x, num\_bins, normed=1, facecolor='blue', alpha=0.5)    *# add a 'best fit' line*  y = mlab.normpdf(bins, mu, sigma)  plt.plot(bins, y, 'r--')  plt.xlabel('Smarts')  plt.ylabel('Probability')  plt.title(r'Histogram of IQ: $**\m**u=100$, $**\s**igma=15$')    *# Tweak spacing to prevent clipping of ylabel*  plt.subplots\_adjust(left=0.15)  plt.show() |