CSCE 110: Programming I

David Kebo Hougninou

Basics of Python

Simulation and Plotting
https://matplotlib.org/tutorials/
Simulation and Plotting

Now that we know how to perform simulations, we need a better way to visualize our results.

Python makes this easy by providing us with the **matplotlib** library to create elegant plots.

```
import matplotlib.pyplot as plot
```

**matplotlib** is the most popular graphing and data visualization library for Python.

**pyplot** is a collection of command style functions to generate plots quickly.

**pyplot** functions makes some change to a figure: e.g.

- creates a figure
- creates a plotting area in a figure
- plots some lines in a plotting area
- decorates the plot with labels, etc.

```
import matplotlib.pyplot as plot
```
Plot y versus x as lines and/or markers.

```python
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.ylabel('some numbers')
plt.show()
```

If the argument to the `plot()` command is a one list, matplotlib assumes that the list is a sequence of y values.

The matching list of x coordinates starts at 0. [0, 1, 2, 3].

Plot y versus x as lines and/or markers.

```python
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.ylabel('Some numbers')
plt.show()
```

The x values are [1, 2, 3, 4] and the y values are [1, 4, 9, 16].
Plotting: customizing a plot

Plot y versus x as lines and/or markers.

```python
import matplotlib.pyplot as plt
plt.plot([74, 133, 495, 644], [2.8, 2.9, 5.6, 7], color='r', marker='o')
plt.ylabel('Acceleration of the racing car (s)')
plt.xlabel('Horse power of the racing car (HP)')
plt.title('Gran Turismo Sport cars specs')
plt.legend(['Specs'])
plt.grid()
plt.show()
```

Plotting: multiple plots

```python
import matplotlib.pyplot as plt

# plot values
plt.plot(list(range(0,5)), [9,4,5,2,3], label='curve 1',
          marker='o', color='blue', linestyle='-.')
plt.plot(list(range(0,5)), [12,5,33,2,4], label='curve 2',
          marker='x', color='red', linestyle='--')

# customize plot
plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title('a simple plot')
plt.legend()
plt.grid()

# add space around the border of the axes
plt.xlim(-0.1, 4.1)
plt.ylim(-1, 35)

# save and view plot
plt.savefig("sample-plot.pdf") # saves file in PDF
plt.savefig("sample-plot.png") # saves file in PNG format
plt.show() # allows for interactive exploration of the plot
```
Plotting: markers

```python
3 # plot values
4 plot.plot(list(range(0,5)), [9,4,5,2,3], label='curve 1',
5 marker='o', color='blue', linestyle='--')
```

<table>
<thead>
<tr>
<th>Marker</th>
<th>String Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circle</td>
<td>o</td>
</tr>
<tr>
<td>Diamond</td>
<td>D</td>
</tr>
<tr>
<td>Point</td>
<td>.</td>
</tr>
<tr>
<td>Plus</td>
<td>+</td>
</tr>
<tr>
<td>Square</td>
<td>s</td>
</tr>
<tr>
<td>Star</td>
<td>*</td>
</tr>
<tr>
<td>X</td>
<td>x</td>
</tr>
</tbody>
</table>

Plotting: linestyle

```python
3 # plot values
4 plot.plot(list(range(0,5)), [9,4,5,2,3], label='curve 1',
5 marker='o', color='blue', linestyle='--')
```

<table>
<thead>
<tr>
<th>Linestyle</th>
<th>String Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solid line</td>
<td>-</td>
</tr>
<tr>
<td>Single dashed line</td>
<td>--</td>
</tr>
<tr>
<td>Single dashed-dot line</td>
<td>-.</td>
</tr>
<tr>
<td>Dotted line</td>
<td>:</td>
</tr>
</tbody>
</table>
Plotting: colors

```python
3     # plot values
4     plot.plot(list(range(0,5)), [9,4,5,2,3], label='curve 1',
5     marker='o', color='blue', linestyle='--')
```

For a complete list of named colors in matplotlib, visit: [http://matplotlib.org/examples/color/named_colors.html](http://matplotlib.org/examples/color/named_colors.html)

---

Plotting: multiple plots

![A simple plot with two curves](image)
Exercise

Write a program in Python that simulates the number of heads and tails that appear using a fair coin with:

• 10 flips
• 100 flips
• 1,000 flips
• 10,000 flips
• 100,000 flips
• 1,000,000 flips

Calculate the average and plot the results in a line chart, defining one curve of each output.

Exercise: perform simulation

```python
def flip_coin():
    return random.choice(['heads', 'tails'])

def simulate(coin_tosses):
    heads = 0
    tails = 0
    for i in range(0, coin_tosses):
        result = flip_coin()
        if result == 'heads':
            heads += 1
        tails = coin_tosses - heads
    # Print results
    print('Total number of coin tosses: %d' % (coin_tosses))
    print('Number of heads: %d (%.2f percent)' %
          (heads, float(heads)/coin_tosses * 100))
    print('Number of tails: %d (%.2f percent)' %
          (tails, float(tails)/coin_tosses * 100))
    print()
return heads, tails
```
Exercise: save the plot data

```python
def main():
    heads_y_axis = []
    tails_y_axis = []
    x_axis = [10, 100, 10**3, 10**4, 10**5, 10**6]
    for tosses in x_axis:
        heads, tails = simulate(tosses)
        heads_y_axis += [float(heads)/tosses * 100]
        tails_y_axis += [float(tails)/tosses * 100]
    plot_results(heads_y_axis, tails_y_axis, x_axis)

main()
```

Exercise: plot the line chart

```python
# Simulates the number of heads and tails that appear
# using a fair coin.
import random
import matplotlib.pyplot as plot

def plot_results(heads_list, tails_list, x_axis):
    plot.plot(x_axis, heads_list, label='heads', marker='o')
    plot.plot(x_axis, tails_list, label='tails', marker='^')
    plot.xlabel('Coin Tosses')
    plot.ylabel('Percentage (%)')
    plot.legend()
    plot.grid()
    plot.xlim(0,80)
    plot.yscale('log')
    plot.savefig('fair-coin-plot.pdf')  # saves file in PDF
    plot.show()
```
Exercise: line chart analysis

```
... [code example]
```

Plotting: bar chart

`pyplot.bar()` creates bar charts. Bar charts are useful for visualizing counts, or statistics.

```python
import matplotlib.pyplot as plt
import random

quizzes = ['Quiz 1', 'Quiz 2', 'Quiz 3', 'Quiz 4', 'Quiz 5', 'Quiz 6']
grades = [65, 80, 75, 90, 89, 100]

bar_chart = plt.bar(quizzes, grades, align='center', alpha=0.5)
plt.xticks(range(0, 100, 10))
plt.ylabel('Grade')
plt.title('CSCE 110 Quiz grades')

# Set a random colors for every bar
for i in range(len(grades)):
    color = random.choice(['b', 'g', 'r', 'c', 'm', 'y', 'k'])
    bar_chart[i].set_color(color)

plt.show()
```
Plotting: bar chart

`pyplot.bar()` creates bar charts. Bar charts are useful for visualizing counts, or statistics.

![Bar Chart Example]

Plotting: bar chart

`pyplot.barh()` creates horizontal bar charts. Bar charts are useful for visualizing counts, or statistics.

![Horizontal Bar Chart Example]
Plotting: using NumPy

NumPy is the fundamental package for scientific computing with Python.

NumPy provides:

- N-dimensional array objects
- Sophisticated functions
- Tools for integrating C/C++ code
- Linear algebra, trigonometry, Fourier transform, and random numbers

Plotting waves using NumPy

Using the NumPy \texttt{sin()} and \texttt{cos()} functions, we can plot the sine and cosine functions.

```python
import numpy as np
import matplotlib.pyplot as plt

cycles = int(input("Enter the number of cycles: "))

cycles = cycles * np.pi

x = np.arange(0, cycles * np.pi, 0.1)
y = np.sin(x)
z = np.cos(x)

plt.xlabel('radians')
plt.grid()
plt.title("sine and cosine waves")

plt.plot(x, y, x, z)
plt.legend(['sin(x)', 'cos(x)'])
plt.show()
```
Plotting waves using NumPy

Using the NumPy \texttt{sin()} and \texttt{cos()} functions, we can plot the sine and cosine functions.

![Sine and Cosine Waves](image)

Plotting: Barnsley Fern Fractal

The mathematician Michael Barnsley described how to create fern-like fractal shape using repeated applications of a simple transformation on a point.

The transformation is called the Iterated Function System (IFS).

Write a program to draw a Barnsley Fern using a transformation of point.
Plotting: Barnsley Fern transformations

Start with the point (0,0) and randomly select one of the following transformations with the assigned probability:

- **Transformation 1** (85% probability)
  \[
  \begin{align*}
  x_{n+1} &= 0.85x_n + 0.04y_n \\
  y_{n+1} &= -0.04x_n + 0.85y_n + 1.6
  \end{align*}
  \]

- **Transformation 2** (7% probability)
  \[
  \begin{align*}
  x_{n+1} &= 0.2x_n - 0.26y_n \\
  y_{n+1} &= 0.23x_n + 0.22y_n + 1.6
  \end{align*}
  \]

- **Transformation 3** (7% probability)
  \[
  \begin{align*}
  x_{n+1} &= -0.15x_n - 0.28y_n \\
  y_{n+1} &= 0.26x_n + 0.24y_n + 0.44
  \end{align*}
  \]

- **Transformation 4** (1% probability)
  \[
  \begin{align*}
  x_{n+1} &= 0 \\
  y_{n+1} &= 0.16y_n
  \end{align*}
  \]
Plotting: Barnsley Fern iterations

1000 iterations

10000 iterations

References

Python 2D plotting library: https://matplotlib.org/

PyPlot: https://matplotlib.org/tutorials/introductory/pyplot.html

NumPy: http://www.numpy.org/

Barnsley Fern: https://en.wikipedia.org/wiki/Barnsley_fern