Welcome to your Python 3 bootcamp

Joe Corneli

Thursday, October 5, 2017

Outline

Introduction

Analysing Patient Data

Creating Functions

Analyzing Multiple Data Sets

Making Choices

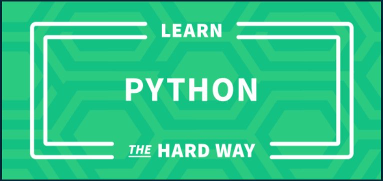
Defensive Programming

Bonus Round: Seaborn

Introduction

Source of these exercises:

Today's method inspired by:



Idea: you will try to figure out the answers, if you can't get them I will show you. Either way, you have to type them all in.

Note: if you get something “close” to the answer on the slides, that’s just fine, and maybe more interesting!

Analysing Patient Data

```python
import numpy
```

```
numpy.loadtxt(fname='inflammation-01.csv',
 ↪  delimiter=',')
```

```
weight_kg = 55
```

```
print(weight_kg)
```

```
print('weight in pounds:', 2.2 * weight_kg)
```

```
weight_kg = 57.5
print('weight in kilograms is now:', weight_kg)
```

```python
weight_lb = 2.2 * weight_kg
print('weight in kilograms:', weight_kg, 'and in pounds:', weight_lb)
```

```
weight_kg = 100.0
print('weight in kilograms is now:', weight_kg,
 ↪  'and weight in pounds is still:', weight_lb)
```

```
data = numpy.loadtxt(fname='inflammation-01.csv',
 ↪  delimiter=',')
```

```
print(data)
```

```
print(type(data))
```

```
print(data.shape)
```

```
print('first value in data:', data[0, 0])
```

```python
print('middle value in data:', data[30, 20])
```

```
print(data[0:4, 0:10])
```

```
print(data[5:10, 0:10])
```

```
print(data[0:10:3, 0:10:2])
```

```python
small = data[:3, 36:]
print('small is:')
print(small)
```

```
print(data.mean())
```

```python
print('maximum inflammation:', data.max())
print('minimum inflammation:', data.min())
print('standard deviation:', data.std())
```

```
patient_0 = data[0, :] # 0 on the first axis,
 ↪  everything on the second
print('maximum inflammation for patient 0:',
 ↪  patient_0.max())
```

```
print('maximum inflammation for patient 2:',
 ↪  data[2, :].max())
```

```
print(data.mean(axis=0))
```

```
print(data.mean(axis=0).shape)
```

```
print(data.mean(axis=1))
```

```
element = 'oxygen'
print('first three characters:', element[0:3])
print('last three characters:', element[3:6])
```

```
%matplotlib inline
```

```python
from matplotlib import pyplot
pyplot.imshow(data)
pyplot.show()
```

```
ave_inflammation = data.mean(axis=0)
pyplot.plot(ave_inflammation)
pyplot.show()
```

```
print('maximum inflammation per day')
pyplot.plot(data.max(axis=0))
pyplot.show()

print('minimum inflammation per day')
pyplot.plot(data.min(axis=0))
pyplot.show()
```

```
import numpy as np
from matplotlib import pyplot as plt

data = np.loadtxt(fname='inflammation-01.csv',
 ↪  delimiter=',')

plt.figure(figsize=(10.0, 3.0))
plt.subplot(1, 3, 1)
plt.ylabel('average')
plt.plot(data.mean(0))
plt.subplot(1, 3, 2)
plt.ylabel('max')
plt.plot(data.max(0))
plt.subplot(1, 3, 3)
plt.ylabel('min')
plt.plot(data.min(0))
plt.tight_layout()
plt.show()
```

Creating Functions

```
def fahr_to_kelvin(temp):
    return ((temp - 32) * (5/9)) + 273.15
print('freezing point of water:',
 ↪  fahr_to_kelvin(32))
print('boiling point of water:',
 ↪  fahr_to_kelvin(212))
```

```python
def kelvin_to_celsius(temp):
    return temp - 273.15

print('absolute zero in Celsius:',
 ↪  kelvin_to_celsius(0.0))
```

```
def fahr_to_celsius(temp):
    temp_k = fahr_to_kelvin(temp)
    result = kelvin_to_celsius(temp_k)
    return result

print('freezing point of water in Celsius:',
 ↪  fahr_to_celsius(32.0))
```

```
original = 32.0
final = fahr_to_celsius(original)
```

```
print('final value of temp after all function
 ↪  calls:', temp)
```

```python
import numpy

def span(a):
    diff = a.max() - a.min()
    return diff

data = numpy.loadtxt(fname='inflammation-01.csv',
 ↪  delimiter=',')
print('span of data', span(data))
```

```
diff = numpy.loadtxt(fname='inflammation-01.csv',
 ↪  delimiter=',')
print('span of data:', span(diff))
```

```python
def center(data, desired):
    return (data - data.mean()) + desired
```

```
z = numpy.zeros((2,2))
print(center(z, 3))
```

```
data = numpy.loadtxt(fname='inflammation-01.csv',
 ↪  delimiter=',')
print(center(data, 0))
```

```
print('original min, mean, and max are:',
 ↪  data.min(), data.mean(), data.max())
centered = center(data, 0)
print('min, mean, and and max of centered data
 ↪  are:', centered.min(), centered.mean(),
 ↪  centered.max())
```

```
print('std dev before and after:', data.std(),
 ↪  centered.std())
```

```
print('difference in standard deviations before
 ↪  and after:', data.std() - centered.std())
```

```python
# center(data, desired): return a new array
 ↪  containing the original data centered around
 ↪  the desired value.
def center(data, desired):
    return (data - data.mean()) + desired
```

```
def center(data, desired):
    '''Return a new array containing the original
     ↪  data centered around the desired
     ↪  value.'''
    return (data - data.mean()) + desired
```

```
help(center)
```

```
def center(data, desired):
    '''Return a new array containing the original
     ↪  data centered around the desired value.
    Example: center([1, 2, 3], 0) => [-1, 0,
 ↪  1]'''
    return (data - data.mean()) + desired

help(center)
```

```
numpy.loadtxt('inflammation-01.csv',
 ↪  delimiter=',')
```

```
numpy.loadtxt('inflammation-01.csv', ',')
```

```
def center(data, desired=0.0):
    '''Return a new array containing the original
     ↪  data centered around the desired value (0
     ↪  by default).
    Example: center([1, 2, 3], 0) => [-1, 0,
 ↪  1]'''
    return (data - data.mean()) + desired
```

```
test_data = numpy.zeros((2, 2))
print(center(test_data, 3))
```

```
more_data = 5 + numpy.zeros((2, 2))
print('data before centering:', more_data)
print('centered data:', center(more_data))
```

```python
def display(a=1, b=2, c=3):
    print('a:', a, 'b:', b, 'c:', c)

print('no parameters:')
display()
print('one parameter:')
display(55)
print('two parameters:')
display(55, 66)
```

```
print('only setting the value of c')
display(c=77)
```

```
help(numpy.loadtxt)
```

```
def foo(bar=[]):           # bar is optional and
 ↪  defaults to [] if not specified
       bar.append("baz")
       return bar
foo()
```

```
foo()
```

```
def foo(bar=None):
       if bar is None:
           bar = []
       bar.append("baz")
       return bar
```

```
foo()
foo()
```

Analyzing Multiple Data Sets

```
%matplotlib inline
import numpy as np
from matplotlib import pyplot as plt
def analyze(filename):
    data = np.loadtxt(fname=filename,
     ↪  delimiter=',')
    plt.figure(figsize=(10.0, 3.0))
    plt.subplot(1, 3, 1)
    plt.ylabel('average')
    plt.plot(data.mean(0))
    plt.subplot(1, 3, 2)
    plt.ylabel('max')
    plt.plot(data.max(0))
    plt.subplot(1, 3, 3)
    plt.ylabel('min')
    plt.plot(data.min(0))
    plt.tight_layout()
    plt.show()
```

```
analyze('inflammation-02.csv')
```

```python
def print_characters(element):
    print(element[0])
    print(element[1])
    print(element[2])
    print(element[3])

print_characters('lead')
```

```
print_characters('tin')
```

```python
def print_characters(element):
    for char in element:
        print(char)

print_characters('lead')
```

```
print_characters('oxygen')
```

```python
length = 0
for vowel in 'aeiou':
    length = length + 1
print('There are', length, 'vowels')
```

```python
letter = 'z'
for letter in 'abc':
    print(letter)
print('after the loop, letter is', letter)
```

```
print(len('aeiou'))
```

```python
odds = [1, 3, 5, 7]
print('odds are:', odds)
```

```python
print('first and last:', odds[0], odds[-1])
```

```
for number in odds:
    print(number)
```

```python
names = ['Newton', 'Darwing', 'Turing'] # typo in Darwin's name
print('names is originally:', names)
names[1] = 'Darwin' # correct the name
print('final value of names:', names)
```

```
name = 'Bell'
name[0] = 'b'
```

```
odds.append(11)
print('odds after adding a value:', odds)
```

```python
del odds[0]
print('odds after removing the first element:', odds)
```

```
odds.reverse()
print('odds after reversing:', odds)
```

```python
import glob
```

```python
print(glob.glob('*.ipynb'))
```

```python
print(glob.glob('*.csv'))
```

```python
filenames = glob.glob('*.csv')
filenames = filenames[0:3]
for f in filenames:
    print(f)
    analyze(f)
```

Making Choices

```
from ipythonblocks import ImageGrid
```

```
grid = ImageGrid(5, 3)
grid.show()
```

```python
print('grid width:', grid.width)
print('grid height:', grid.height)
print('grid lines on:', grid.lines_on)
```

```python
position = (12.3, 45.6)
print('position is:', position)
color = (10, 20, 30)
print('color is:', color)
```

```
print('first element of color is:', color[0])
```

```
color[0] = 40
print('first element of color after change:',
 ↪  color[0])
```

```
row = ImageGrid(8, 1)
row[0, 0] = (0, 0, 0)   # no color => black
row[1, 0] = (255, 255, 255) # all colors => white
row[2, 0] = (255, 0, 0) # all red
row[3, 0] = (0, 255, 0) # all green
row[4, 0] = (0, 0, 255) # all blue
row[5, 0] = (255, 255, 0) # red and green
row[6, 0] = (255, 0, 255) # red and blue
row[7, 0] = (0, 255, 255) # green and blue
row.show()
```

```python
from ipythonblocks import show_color
show_color(214, 90, 127)
```

```
from ipythonblocks import colors
c = ImageGrid(3, 2)
c[0, 0] = colors['Fuchsia']
c[0, 1] = colors['Salmon']
c[1, 0] = colors['Orchid']
c[1, 1] = colors['Lavender']
c[2, 0] = colors['LimeGreen']
c[2, 1] = colors['HotPink']
c.show()
```

```python
num = 37
if num > 100:
    print('greater')
else:
    print('not greater')
print('done')
```

```python
num = 53
print('before conditional...')
if num > 100:
    print('53 is greater than 100')
print('...after conditional')
```

```
def sign(num):
    if num > 0:
        return 1
    elif num == 0:
        return 0
    else:
        return -1

print('sign of -3:', sign(-3))
```

```
if (1 > 0) and (-1 > 0):
    print('both parts are true')
else:
    print('one part is not true')
```

```
if (1 < 0) or ('left' < 'right'):
    print('at least one test is true')
```

```python
numbers = [-5, 3, 2, -1, 9, 6]
total = 0
for n in numbers:
    if n >= 0:
        total = total + n
print('sum of positive values:', total)
```

```
pos_total = 0
neg_total = 0
for n in numbers:
    if n >= 0:
        pos_total = pos_total + n
    else:
        neg_total = neg_total + n
print('negative and positive sums are:',
 ↪  neg_total, pos_total)
```

```python
for consonant in 'bcd':
    for vowel in 'ae':
        print(consonant + vowel)
```

```
square = ImageGrid(5, 5)
for x in range(square.width):
    for y in range(square.height):
        if x < y:
            square[x, y] = colors['Fuchsia']
        elif x == y:
            square[x, y] = colors['Olive']
        else:
            square[x, y] = colors['SlateGray']
square.show()
```

```python
import numpy as np
data = np.loadtxt(fname='inflammation-01.csv',
 ↪  delimiter=',')
print('data shape:', data.shape)
```

```
width, height = data.shape
heatmap = ImageGrid(width, height)
```

```
for x in range(width):
    for y in range(height):
        if data[x, y] < data.mean():
            heatmap[x, y] = colors['Red']
        elif data[x, y] == data.mean():
            heatmap[x, y] = colors['Green']
        else:
            heatmap[x, y] = colors['Blue']
heatmap.show()
```

```python
flipped = data.transpose()
width, height = flipped.shape
heatmap = ImageGrid(width, height, block_size=5)
center = flipped.mean()
for x in range(width):
    for y in range(height):
        if flipped[x, y] < (0.8 * center):
            heatmap[x, y] = colors['Orchid']
        elif flipped[x, y] > (1.2 * center):
            heatmap[x, y] = colors['HotPink']
        else:
            heatmap[x, y] = colors['Fuchsia']
heatmap.show()
```

```
def make_heatmap(values, low_color, mid_color,
 ↪  high_color, low_band, high_band, block_size):
    '''Make a 3-colored heatmap from a 2D array
     ↪  of data.'''
    width, height = values.shape
    result = ImageGrid(width, height,
     ↪  block_size=block_size)
    center = values.mean()
    for x in range(width):
        for y in range(height):
            if values[x, y] < low_band * center:
                result[x, y] = low_color
            elif values[x, y] > high_band *
             ↪  center:
                result[x, y] = high_color
            else:
                result[x, y] = mid_color
    return result
```

```
h = make_heatmap(flipped, colors['Orchid'],
 ↪  colors['Fuchsia'], colors['HotPink'], 0.8,
 ↪  1.2, 5)
h.show()
```

```
h = make_heatmap(flipped, colors['Gray'],
 ↪  colors['YellowGreen'], colors['SpringGreen'],
 ↪  0.5, 1.5, 5)
h.show()
```

```
def make_heatmap(values, low_band=0.5,
 ↪  high_band=1.5,
                 low_color=colors['Gray'],
                  ↪  mid_color=colors['YellowGreen'],
                  ↪  high_color=colors['SpringGreen'],
                 block_size=5):
    '''Make a 3-colored heatmap from a 2D array.
    Default color scheme is gray to green.'''
    width, height = values.shape
    result = ImageGrid(width, height,
     ↪  block_size=block_size)
    center = values.mean()
    # ...
```

```
# ...
for x in range(width):
    for y in range(height):
        if values[x, y] < low_band * center:
            result[x, y] = low_color
        elif values[x, y] > high_band * center:
            result[x, y] = high_color
        else:
            result[x, y] = mid_color
return result
```

```
h = make_heatmap(flipped, 0.5, 1.5,
 ↪  colors['Gray'], colors['YellowGreen'],
 ↪  colors['SpringGreen'], 5)
h.show()
```

```
h = make_heatmap(flipped, 0.4, 1.6)
h.show()
```

Defensive Programming

```python
numbers = [1.5, 2.3, 0.7, -0.001, 4.4]
total = 0.0
for n in numbers:
    assert n >= 0.0, 'Data should only contain
     ↪  positive values'
    total += n
print('total is:', total)
```

```
def normalize_rectangle(rect):
    '''Normalizes a rectangle so that it is at
     ↪  the origin and 1.0 units long on its
     ↪  longest axis.'''
    assert len(rect) == 4, 'Rectangles must
     ↪  contain 4 coordinates'
    x0, y0, x1, y1 = rect
    assert x0 < x1, 'Invalid X coordinates'
    assert y0 < y1, 'Invalid Y coordinates'

    dx = x1 - x0
    dy = y1 - y0
    # ...
```

```python
# ...
if dx > dy:
    scaled = float(dx) / dy
    upper_x, upper_y = 1.0, scaled
else:
    scaled = float(dx) / dy
    upper_x, upper_y = scaled, 1.0

assert 0 < upper_x <= 1.0, 'Calculated upper X coordinate invalid'
assert 0 < upper_y <= 1.0, 'Calculated upper Y coordinate invalid'

return (0, 0, upper_x, upper_y)
```

```
print(normalize_rectangle( (0.0, 1.0, 2.0) )) #
 ↪  missing the fourth coordinate
```

```
print(normalize_rectangle( (4.0, 2.0, 1.0, 5.0)
 ↪  )) # X axis inverted
```

```
print(normalize_rectangle( (0.0, 0.0, 1.0, 5.0)
 ↪  ))
```

```
print(normalize_rectangle( (0.0, 0.0, 5.0, 1.0)
 ↪  ))
```

```
assert range_overlap([ (0.0, 1.0) ]) == (0.0,
 ↪  1.0)
assert range_overlap([ (2.0, 3.0), (2.0, 4.0) ])
 ↪  == (2.0, 3.0)
assert range_overlap([ (0.0, 1.0), (0.0, 2.0),
 ↪  (-1.0, 1.0) ]) == (0.0, 1.0)
```

```
assert range_overlap([ (0.0, 1.0), (5.0, 6.0) ])
 ↪  == None
assert range_overlap([ (0.0, 1.0), (1.0, 2.0) ])
 ↪  == None
```

```python
def range_overlap(ranges):
    '''Return common overlap among a set of [low, high] ranges.'''
    lowest = 0.0
    highest = 1.0
    for (low, high) in ranges:
        lowest = max(lowest, low)
        highest = min(highest, high)
    return (lowest, highest)
```

```python
def test_range_overlap():
    assert range_overlap([ (0.0, 1.0), (5.0, 6.0)
     ↪  ]) == None
    assert range_overlap([ (0.0, 1.0), (1.0, 2.0)
     ↪  ]) == None
    assert range_overlap([ (0.0, 1.0) ]) == (0.0,
     ↪  1.0)
    assert range_overlap([ (2.0, 3.0), (2.0, 4.0)
     ↪  ]) == (2.0, 3.0)
    assert range_overlap([ (0.0, 1.0), (0.0,
     ↪  2.0), (-1.0, 1.0) ]) == (0.0, 1.0)
```

```
test_range_overlap()
```

Bonus Round: Seaborn

Congratulations, you've made it through the entire bootcamp.

You're welcome to go through the material on command line python on your own, later: `http://swcarpentry.github.io/python-novice-inflammation/10-cmdline/`

Now, however, we will have a look at a plotting library called Seaborn.