Programming with Python - Day 2

EOAS Software Carpentry Workshop

September 25th, 2015

Creating Functions - Defining a Function

Learning Goals

1. Explain why we should divide programs into small, single-purpose functions.
2. Define a function that takes parameters.
3. Return a value from a function.

Example Code

-
  ```
  def fahr_to_kelvin(temp):
       return ((temp - 32) * (5/9)) + 273.15
  ```

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

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

Exercise

Write a function called analyze that takes a filename as a parameter and displays the three graphs produced in the previous lesson, i.e., analyze('inflammation-01.csv') should produce the graphs already shown, while analyze('inflammation-02.csv') should produce corresponding graphs for the second data set. Hint: a function can just “do” something. It doesn't necessarily need to return anything.

Solution

```
def analyze(filename):
    data = np.loadtxt(fname=filename, delimiter=',')
    fig = plt.figure(figsize=(10.0, 3.0))

    axes1 = fig.add_subplot(1, 3, 1)
    axes2 = fig.add_subplot(1, 3, 2)
    axes3 = fig.add_subplot(1, 3, 3)

    axes1.set_ylabel('average')
    axes1.plot(data.mean(axis=0))

    axes2.set_ylabel('max')
    axes2.plot(data.max(axis=0))

    axes3.set_ylabel('min')
    axes3.plot(data.min(axis=0))

    fig.tight_layout()
    plt.show(fig)
```

Defining a Function

```
def detect_problems(filename):

    data = np.loadtxt(fname=filename, delimiter=',')

    if data.max(axis=0)[0] == 0 and data.max(axis=0)[20] ==
        print('Suspicious looking maxima!')
    elif data.min(axis=0).sum() == 0:
        print('Minima add up to zero!')
    else:
        print('Seems OK!')
```

Testing and Documentation

Learning Goal

3. Test and debug a function.

Example Code

-
  ```
  def centre(data, desired):
      return (data - data.mean()) + desired
  ```
- `z = numpy.zeros((2,2))`
- `print centre(z, 3)`
- `print data.std() - centred.std()`

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

Defining Defaults

Learning Goals

6. Set default values for function parameters.

Example Code

-
  ```
  def center(data, desired = 0):
  ```

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

Exercise

“Adding” two strings produces their concatenation: `’a’ + ’b’` is `’ab’`. Write a function called fence that takes two parameters called original and wrapper and returns a new string that has the wrapper character at the beginning and end of the original. A call to your function should look like this:

```
print(fence(’name’, ’*’))
*name*
```

Exercise

"Adding" two strings produces their concatenation: `'a' + 'b'` is `'ab'`. Write a function called fence that takes two parameters called original and wrapper and returns a new string that has the wrapper character at the beginning and end of the original. A call to your function should look like this:

```
print(fence('name', '*'))
*name*
```

Solution

```
def fence(original, wrapper):
    '''Returns a string with charcter wrapper added to the
       beginning and end of string original.'''

       return wrapper + original + wrapper
```

Command-line programs

1. Create a Python module containing functions that can be imported into notebooks and other modules.
2. Use the values of command-line arguments in a program.
3. Read data from standard input in a program so that it can be used in a pipeline.

Switching to shell commands

$ in front of a command that tells you to run that command in the shell rather than the Python interpreter

Write a command-line program that does addition and subtraction:

```
& python arith.py add 1 2
3
& python arith.py subtract 3 4
-1
```

Rewrite `readings.py` so that it uses `-n`, `-m`, and `-x` instead of `--min`, `--mean`, and `--max` respectively. Is the code easier to read? Is the program easier to understand?

Separately, modify `readings.py` so that if no action is given it displays the means of the data.