Carleton University
Department of Systems and Computer Engineering
SYSC 1005 - Introduction to Software Development - Fall 2014

Lab 9 - Using Tuples and Sets to Structure Data

Attendance/Demo

To receive credit for this lab, you must make reasonable progress towards completing the exercises and demonstrate the code you complete. **Also, you must submit your lab work to cuLearn by the end of the lab period**. (Instructions are provided in the *Wrap Up* section at the end of this handout.)

When you have finished all the exercises, call a TA, who will review the code you wrote. For those who don't finish early, a TA will ask you to demonstrate whatever code you've completed, starting about 30 minutes before the end of the lab period. **Any unfinished exercises should be treated as "homework"; complete these on your own time, before your next lab.**

Getting Started

1. Launch Wing IDE 101. Check the message Python displays in the shell window and verify that Wing is running Python version 3.4. If another version is running, ask your instructor or a TA for help to reconfigure Wing to run Python 3.4.

2. In the editor, open a new file. Save this file as points.py.

Part 1 - Comparing Lists and Tuples (Review); Storing Tuples in Sets

Exercise 1

Step 1: Suppose we want to represent points on a two-dimensional Cartesian plane. We could store the (x, y) coordinates of each point in a list. Type this statement, which binds `point1` to a list that represents the point (1.0, 2.0):

```
>>> point1 = [1.0, 2.0]
```

What is displayed when Python evaluates `point1`? Try this:

```
>>> point1
```

Step 2: The problem with the approach used in Step 1 is that Python lists are *mutable*. We could call the `append` method to insert a `float` at the end of the list. Try this:

```
>>> point1.append(3.0)
>>> point1
```

The list now represents a point with three coordinates; that is, a point in three-dimensional space.

We could then call the `pop` method on this list to remove numbers. Try this:

```
>>> point1.pop(0)  # Remove the number at index 0 in the list
>>> point1

>>> point1.pop()   # Remove the last number in the list
>>> point1
```

The list now has only one value, so it doesn't represent a point.

Step 3: To avoid the problems explored in the previous step, we should represent points using an *immutable* container. Recall that a *tuple* is a container that is similar to a list, except that it can't be modified after is initialized. In the next experiment, we'll see how to represent the point (1.0, 2.0) using a tuple containing two real numbers, 1.0 and 2.0.

Type these statements in the shell (note that the numbers are enclosed in parenthesis, not square brackets):

```
>>> point1 = (1.0, 2.0)
>>> point1
>>> type(point1)
```

What is displayed when variable `point1` is evaluated? What is the type of the object bound to `point1`?

Step 4: As with lists, an object stored in a tuple can be retrieved by using the `[]` operator to specify its position in the tuple. Type these statements to retrieve the *x* and *y* coordinates of the point represented by `point1`. What values are displayed when variables `x` and `y` are evaluated?

```
>>> x = point1[0]
>>> y = point1[1]
>>> x
>>> y
```

We can unpack all the objects in a tuple, binding them to individual variables, by using a statement of the form:

```
var_1, var_2, var_3, ..., var_n = t
```

where `t` is variable bound to a tuple containing *n* objects. This is equivalent to:

```
var_1 = t[0]
var_2 = t[1]
    ...
var_n = t[n-1]
```

Try this experiment. What values are displayed when variables `x` and `y` are evaluated?

```
>>> point2 = (4.0, 6.0)
>>> x, y = point2
>>> x
>>> y
```

Step 5: We can easily demonstrate that tuples are immutable. You can't replace objects in a tuple, or add objects to or remove objects from a tuple. Type these statements in the shell. What is displayed when each statement is executed?

```
>>> point2[0] = 2.0   # Can we change the point to (2.0, 6.0)?
>>> point2.append(4.0)  # Can we add a third coordinate?
>>> point2.pop(0)   # Can we remove the first coordinate?
```

Exercise 2

In points.py, define a function that is passed two tuples, each representing a point on a two-dimensional plane. The function returns the distance between the two points. The function header is:

```
def distance(pt1, pt2):
```

Use the shell to test your function. Here is one test case:

```
>>> point1 = (1.0, 2.0)
>>> point2 = (4.0, 6.0)
>>> distance(point1, point2)
5.0
```

Exercise 3

Try this experiment, which creates a set containing the points (1.0, 2.0), (4.0, 6.0) and (10.0, -2.0). What is displayed when `points` is evaluated?

```
>>> points = {(1.0, 2.0), (4.0, 6.0), (10.0, -2.0)}
>>> points
```

We can also initialize the set this way. Try this experiment::

```
>>> point1 = (1.0, 2.0)
>>> point2 = (4.0, 6.0)
>>> point3 = (10.0, -2.0)
>>> points = {point1, point2, point3}
>>> points
```

Or, we can call the `add` method to initialize the set, one point at a time. What is displayed when `points` is evaluated?

```
>>> points = set()
>>> points.add(point1)
>>> points.add(point2)
>>> points.add(point3)
>>> points
```

What happens if we try to insert a point that is already in the set? Try this experiment:

```
>>> points.add(point2)
>>> points
```

We can use a `for` loop to iterate over all the tuples in the set. What is displayed when this loop is executed?

```
>>> for point in points:
...     print(point)
...
```

Part 2 - Curve Fitting Using the Method of Least Squares

Every engineering student has tackled the problem of fitting a line through a set of points obtained during a lab experiment.

Linear regression is a technique for fitting a curve through a set of points by applying a goodness-of-fit criterion. The most common form of linear regression is *least-squares fitting*. The mathematical derivation of this technique is beyond the scope of this course, but if you're interested, you can read this page: http://mathworld.wolfram.com/LeastSquaresFitting.html

Suppose we have a set of n points, $\{ (x_0, y_0), (x_1, y_1), \ldots (x_{n-1}, y_{n-1}) \}$. The equation of a straight line through these points has the form $y = mx + b$, where m is the slope of the line and b is the y-intercept.

Using the method of least squares, the slope and y-intercept of the line with the best fit are calculated this way:

$$m = (sumx \times sumy - n \times sumxy) \div (sumx \times sumx - n \times sumxx)$$

$$b = (sumx \times sumxy - sumxx \times sumy) \div (sumx \times sumx - n \times sumxx)$$

where:

$sumx$ is $x_0 + x_1 + x_2 + \ldots + x_{n-1}$; i.e., the sum of all the x values

sumy is $y_0 + y_1 + y_2 + \ldots + y_{n-1}$; i.e., the sum of all the y values

sumxx is $x_0^2 + x_1^2 + x_2^2 + \ldots + x_{n-1}^2$; i.e., the sum of the all the squares of the x values

sumxy is $x_0 \times y_0 + x_1 \times y_1 + x_2 \times y_2 + \ldots + x_{n-1} \times y_{n-1}$; i.e., the sum of all the products of the (x, y) pairs

Exercise 5

To ensure that you understand these formulas, use the method of least squares to calculate the slope and y-intercept of the line through this set of points: {(1.0, 5.0), (2.0, 8.0), (3.5, 12.5)}. Don't write a program to do this; use a calculator. If your calculations are correct, the equation of the line will be:

$y = 3.0x + 2.0$.

Exercise 6

Download linear_regression.py from cuLearn and open the file in Wing IDE 101. This file contains a function named `get_points`, which returns a set of tuples. Each tuple represents one (x, y) point. In the shell, type these statements to verify that the set returned by this function contains three tuples:

```
>>> samples = get_points()
>>> len(samples)  # How many elements are in the set?
>>> samples
```

Exercise 7

In linear_regression.py, define a function named `fit_line_to_points` which is passed a set of tuples, with each tuple representing an (x, y) point. This function should use the method of least squares to calculate the slope and y-intercept of the best-fit straight line through the points. The slope and intercept must be returned in a tuple.

The function header is:

```
def fit_line_to_points(points):
```

Use the shell to call `fit_line_to_points`, passing it the set returned by `get_points`. Verify that the slope and y-intercept returned by the function are 3.0 and 2.0.

Exercise 8

The block after the statement, `if __name__ == "__main__":` contains a single statement, `pass`. Replace `pass` with a short script that:

- calls `fit_line_to_points`, passing it the set of points returned by `get_points`;
- prints "The best-fit line is y = *m*x + *b*", where *m* and *b* are the values returned by your

function.

Test your script.

Part 3 - Working with Data Stored in a File

Suppose we want to fit lines through different sets of points. You could modify the script you wrote in Exercise 8 to read the (x, y) coordinates of each point from the keyboard, but this would be tedious and error-prone when you want to fit a line through many points. Instead, we'll store the points in a text file that can be prepared with any text editor, and modify the script to read the points from the file.

Exercise 9 - Experiments with Text Files

Step 1: You've been provided with a file named data.txt that contains the (x, y) coordinates of three points, one per line:

1.0 5.0

2.0 8.0

3.5 12.5

Download this file from cuLearn to the same folder where linear_regression.py is stored.

To read data from a file, we must first open it, using Python's built-in `open` function. Type this:

```
>>> infile = open("data.txt", "r")
```

`open` takes two arguments, The first argument is a string containing the name of the file to open. The second argument is a string that specifies the mode; `"r"` means open the file for reading.

`open` returns an object that stores information about the opened file. Here, we've bound that object to variable `infile`.

Next, we'll use the `readline` method to read the file, one line at a time. Type these statements. What is displayed each time variable `s` is evaluated?

```
>>> s = infile.readline()
>>> s
>>> s = infile.readline()
>>> s
>>> s = infile.readline()
>>> s
>>> s = infile.readline()
>>> s
>>> infile.close()
```

Notes:

- The `readline` method returns each line as a string (an object of type `str`).
- There is a `\n` at the end of each string. This is the *newline character*, which terminates every line of text in the file.
- The `readline` method returns an empty string (`''`) if it is called after all the lines have been read from the file.
- After you've finished reading a file, you should always close it by calling the `close` method.

Step 2: We can use a `for` loop to read lines from a file. Define the following function in linear_regression.py:

```
def read_and_print_lines():
    infile = open("data.txt", "r")
    for line in infile:
        print(line)
    infile.close()
```

On each iteration of the `for` loop, Python automatically calls `readline` and binds the line read from the file to variable `line`.

Call `read_and_print_lines` from the shell, and observe what is printed.

Exercise 10 - Converting Strings to Real Numbers

Step 1: Recall that the string representation of a real number can be converted to a `float` by calling Python's built-in `float` function. Try this:

```
>>> s = "2.0"
>>> x = float(s)
>>> x
```

Step 2: The `float` function doesn't work with strings containing multiple numbers. Try this:

```
>>> float('1.0   5.0')
```

We need to break up each line read from the file into two strings, each containing one number, then individually convert each string to a `float`. This is easy to do, using the `split` method. Try this:

```
>>> s = '1.0   5.0'
>>> numbers = s.split()
>>> numbers
```

Notice that `split` chops the string into two substrings, each containing one of the numbers, and

returns a list containing both substrings.

Step 3: How would you convert both strings in list `numbers` to values of type `float`? Design some experiments that show how to do this, and execute them in the shell.

Exercise 11 - Reading Points from a Text File

Apply what you learned in Exercises 9 and 10 by defining a function named `read_points` in linear_regression.py. This function takes a single argument, a string containing the name of a text file. The function header is:

```
def read_points(filename):
```

Each line in the text file will contain two real numbers. The function will return a set of tuples, with each tuple containing one (x, y) point (i.e., a pair of `float`s).

Interactively test this function. If you call `read_points` with "data.txt" as the argument, the set returned by the function should be identical to the set returned by `get_points`.

Exercise 12 - Modifying the Curve-Fitting Script to Read Points from a File

Modify your main script (not function `read_points`) so that it prompts the user to enter the name of a text file. The script should then read the points from the file, then calculate and display the formula for the best-fit line through the points.

Test your script.

Wrap-up

1. Remember to have a TA review your solutions to the exercises, assign a grade (Satisfactory, Marginal or Unsatisfactory) and have you initial the demo/sign-out sheet.

2. Before you leave the lab, log in to cuLearn and submit linear_regression.py. You do not have to submit points.py. To do this:

    2.1. Click the Submit Lab 9 link. A page containing instructions and your submission status will be displayed. After you've read the instructions, click the Add submission button. A page containing a File submissions box will appear. Drag linear_regression.py to the File submissions box. Do not submit another type of file (e.g., a zip file, a RAR file, a .txt file, etc.)

    2.2. After the icon for linear_regression.py appears in the box, click the Save changes button. At this point, the submission status of your file is "Draft (not submitted)". If you're ready to finish submitting the file, jump to Step 2.4. If you instead want to replace or delete your "draft" file submission, follow the instructions in Step 2.3.

    2.3. You can replace or delete the file by clicking the Edit my submission button.

The page containing the File submissions box will appear.

2.3.1. To overwrite a file you previously submitted with a file having the same name, drag another copy of the file to the File submissions box, then click the Overwrite button when you are told the file exists ("There is already a file called..."). After the icon for the file reappears in the box, click the Save changes button.

2.3.2. To delete a file you previously submitted, click its icon. A dialogue box will appear. Click the Delete button., then click the OK button when you are asked, "Are you sure you want to delete this file?" After the icon for the file disappears, click the Save changes button.

2.4. Once you're sure that you don't want to make any changes, click the Submit assignment button. A Submit assignment page will be displayed containing the message, "Are you sure you want to submit your work for grading? You will not be able to make any more changes." Click the Continue button to confirm that you are ready to submit your lab work. This will change the submission status to "Submitted for grading". **Make sure you do this before you leave the lab.**