Carleton University
Department of Systems and Computer Engineering
ECOR 1051 - Fundamentals of Engineering I

Lab 11 - Using Tuples and Sets to Structure Data

Objective

To learn about two of Python's compound data types (types that store collections of value): tuples and sets.

Learning outcomes: 1, 4, 5, 7; Graduate attributes: 1.3, 5.3 (see the course outline)

Prerequisite Reading

Read these sections from *Practical Programming*, 3rd Edition, Chapter 11, before you start this lab:

- *Storing Data Using Sets* (pages 203 - 209)
- *Storing Data Using Tuples* (pages 209 - 214)

Getting Started

Begin by creating a new file within Wing 101. **Save it as lab11.py**

Exercise 1

Step 1: Suppose we want to represent points on a two-dimensional Cartesian plane. Earlier in the term, you learned how to use Python lists to store collections of data. One way to represent a point is to store its (*x*, *y*) coordinates in a list. For example, we can represent the point (1.0, 2.0) by the list `[1.0, 2.0]`.

Try this experiment in the Python shell.

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

What is displayed when Python evaluates `point1`?

Step 2: The problem with the approach used in Step 1 is that Python lists are *mutable*. For example, we could call the `append` method to insert a `float` at the end of the list. Try this experiment. What is displayed when Python evaluates `point1`?

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

If the list is supposed to represent a two-dimensional point, this doesn't make sense, because the list now contains three values.

We could then call the `pop` method on this list to remove numbers. Try this experiment. What is displayed each time Python evaluates `point1`?

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

>>> point1.pop()   # Remove the last item 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. A *tuple* is similar to a list, but it can't be modified after is created. In the next experiment, you'll learn how to create a tuple that represents a point in the 2-D Cartesian coordinate system.

A tuple is created by a sequence of comma-separated values, enclosed in parentheses. Try this experiment. What is displayed when the last two statements are evaluated?

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

Aside: enclosing the `float`s in parenthesis is optional, so this statement will create the same tuple:

```
>>> point1 = 1.0, 2.0
>>> point1
```

When tuples are displayed, they are always enclosed in parentheses, so some programmers prefer to use surrounding parentheses in expressions that create tuples.

Step 4: As with lists, tuples are ordered sequences, so an item stored in a tuple `t` can be retrieved by the expression `t[i]`, where `i` is the index (position) of the item.

Try this experiment to retrieve the *x* and *y* coordinates of the point represented by `point1`. What is displayed when Python evaluates `x` and `y`?

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

We can *unpack* all the items 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 a variable bound to a tuple containing *n* items. This is equivalent to:

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

Try this experiment. What is displayed when Python evaluates `x` and `y`?

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

Step 5: We can easily demonstrate that tuples are immutable. You can't replace items in a tuple, or insert new items or remove items. Try this experiment. What is displayed when Python executes each statement?

```
>>> 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?
```

Step 6: The elements in a compound data type, such as a list or tuple, aren't limited to integers and floating point values. You can, for example, have lists that contain lists and tuples that contain tuples. Sometimes it makes sense to mix the two types, for example, a tuple that contains lists or a list that contains tuples.

Try this experiment, which demonstrates how to create a list of tuples that represent the points (1.0, 5.0), (2.0, 8.0) and (3.5, 12.5):

```
>>> points = [(1.0, 5.0), (2.0, 8.0), (3.5, 12.5)]
>>> points[0]
>>> points[1]
>>> points[2]
```

Exercise 2

Use the function design recipe from Chapter 3 of *Practical Programming* to develop a function named `average` in file `lab11.py`. The function takes a list of tuples, and each tuple contains three non-negative integers. The function returns a new list of tuples. The three numbers in each tuple are the integer average values of the numbers in the tuple at the same position in the original list.

For example, suppose the first tuple in the list passed to average is `(27, 219, 134)`. The integer average of these values is 126, so the first tuple in the new list will be: `(126, 126, 126)`. (Hint: what Python division operator will produce a value of type `int` when the average is calculated?)

Your function definition must have type annotations and a complete docstring. Use the Python shell to test your function, then add your test code below the function in your file.

Exercise 3

Step 1: When you read *Storing Data Using Sets*, you learned that Python's `set` type allows us to create mutable collections of unordered distinct items. The items stored in a set must be immutable, so sets can contain values of type `int`, `float` or `str`, but we can't store lists or sets in sets. Tuples are immutable, so we can store tuples in sets.

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. What is displayed when `points` is evaluated?

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

We could instead start with an empty set, and call the `add` method to initialize it, one point at a time. Try this experiment. What is displayed when `points` is evaluated?

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

Step 2: What happens if we try to insert a point that is already in the set? Try this experiment:

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

How many copies of point (4.0, 6.0) are in the set?

Step 3: Can we retrieve individual points in the set be retrieved by specifying an index (position)? Try this experiment. What is displayed when `points[0]` is evaluated?

```
>>> points[0]
```

Step 4: We can use a `for` loop to iterate over all the points in the set. What is displayed when this loop is executed?

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

Exercise 4

Use the function design recipe from Chapter 3 of *Practical Programming* to develop a function named `sum_x` in file `lab11.py`. The function takes a set of *n* points: { (x_0, y_0), (x_1, y_1), … (x_{n-1}, y_{n-1}) }. Each point is represented by a tuple containing two `floats`. The first and second `floats` are the point's *x* and *y* coordinates, respectively.

The function returns the sum of all the *x* coordinates: $x_0 + x_1 + x_2 + \ldots + x_{n-1}$.

Your function definition must have type annotations and a complete docstring. Use the Python shell to test your function, and again add your test code into your file below that for Exercise 2.

Wrap Up

Ensure that your code meets the posted marking rubrics for the labs.

Submit lab11.py.

You are required to keep a backup copy of (all) your work for the duration of the term.

Last edited: April 23, 2020