

Activity 3.3.6 Tuples and Lists (14 Questions)

By: Your Names…

Introduction

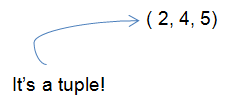
|  |  |
| --- | --- |
| **Strings** can store letters, numbers, and anything else that can be represented by characters. If we want to know the **n**’th character, we look at the **n**’th element in the string.  If you call in to a radio station… Will you be the 10th caller? Will you win the Million Dollars?  Each caller holds one place in line. As you will see, this is similar to a *Python* tuple or list: each element is an different object. You must access the **n**’th item in the list to find out what it is.  Strings, tuples, and lists are just three examples of collections of data. The power of computers comes from iterating across large collections of data. What are some large collections you can think of? | http://www.thehindu.com/multimedia/dynamic/01459/16MN_QUEUE_1459032f.jpg  Image courtesy the Hindu ©2013    File:Messier81 highres.jpg  Image courtesy NASA ©2005  M81, a galaxy 12 million light years behind the Big Dipper, contains about 250 billion stars. |

Procedure

1. Form pairs as directed by your teacher. Meet or greet each other to practice professional skills and establish norms.
2. Launch Canopy and open an editor window.
3. Start a new program in the code editor by choosing **File** > **New** > **Python file**. Save the file as 3\_3\_6\_Your\_Names.py. You will submit this for your assignment

**Part I: Tuples, Syntax**

1. There are other iterables besides strings. One kind is called a **tuple**, or a sequence of values:



Write another tuple. Be sure to put it in parentheses – that’s what tells *Python* that it’s a tuple!

1. One variable can be **bound** to an entire tuple with assignment operator =.

a = (2, 4, 5)

a is now a tuple

Bound means that the variable name is now listed in the namespace in a table that shows variable names and memory addresses. *Python* programmers usually choose all lowercase letters and numbers for any variable name, with underscores separating the words. This is a style **convention** and not a syntax rule. Other conventions for making up variable names exist, like “camelCase.” Employers sometimes specify conventions for their programmers to follow, and your teacher might direct you to use a particular convention for your work.

What convention do you plan to follow for *Python* variable names in the rest of your programs?

\_camel case

1. **Syntax** is the required spelling and grammar of a computer language. *Python* syntax, for example, says that you can use an underscore '\_' but not the dash '-'character in a variable name.

In []: some\_values = ('a', 'b', 3)

**Input the code above** ⤴ **into the ipython console.**

The characters in parentheses are a **tuple**, a list of values that can’t change. A tuple begins and ends with parentheses: ( and ). The variable some\_values is bound to the tuple.

In []: some\_values

Out[]: ('a', 'b', 3)

Just like characters in strings, the individual elements of the tuple are **indexed** so that you can access them one at a time. They are indexed from 0, so a 3-tuple like ('a', 'b', 3) has elements at indices 0, 1, and 2.

It will always work out like this:

if you have ***n*** values in a tuple, then the last one is indexed ***n-1***.

You can retrieve individual values.

In []: some\_values[0]

Out[]: 'a'

In []: some\_values[2]

Out[]: 3

1. Predict the output below and test the outcome in the ipython console.

In []: some\_values[1]

In []: **\_** # 6a. *(list your output.)*

(‘b’)

1. Predict the output below and test the outcome in the ipython console.

In []: some\_values[0:2]

In []: **\_** # 6a. *(list your output.)*

(‘a’, ‘b’)

1. Explain why the input above doesn’t output all of the elements of the tuple.

\_ it stops before listing the end value

1. Tuples are **immutable**, which means that you’re not allowed to change the elements of a tuple. You can only assign a variable that was a tuple to a whole new value, which might be another tuple.

**Tuples are Immutable**

**• Can’t be changed**

**• Can only be reassigned.**

**Result: Tuples are faster than Lists.**

In []: some\_values[2]= '3'

----------------------------------------------------------

TypeError Traceback (most recent call last)

<ipython-input-14-62eb70a8fd4d> in <module>()

----> 1 some\_values[2]= '3'

TypeError: 'tuple' object does not support item assignment

Tuples only are bound to data, never to other int or str variables. If you use a variable expression when assigning a tuple, the expression is evaluated and the value is stored in the tuple. That value won’t change, even if the expression involving the int variable has a different value later.

In []: a = 10

In []: tup = (9, a, 11) *# tup was defined by using a*

In []: tup

Out[]: (9, 10, 11)

In []: a = 15

In []: a *# a has changed*

Out[]: 15

In []: tup *# but tup has not changed*

Out[]: (9, 10, 11)

Predict the output for the following inputs. Then try it, discuss, and explain.

In []: tup[1] == 10

In []: **\_** # 8a. *(Discuss and explain.)*

true

In []: tup = (9, a , 11)

In []: tup[1]

In []: **\_** # 8b. *(Discuss and explain.)*

10

\_

**Part II: Lists**

1. If you list elements inside parentheses, *Python* creates a tuple. If you list the elements inside square brackets, *Python* creates a **list**. Lists are similar to tuples, but their elements can be changed, inserted, and deleted.

In []: values = ['a', 'b', 3]

In []: values

Out[]: ['a', 'b', 3]

List elements can be accessed individually, just like tuple elements.

In []: values[1]

Out[]: 'b'

Predict, try, discuss, and explain the output for this input:

In []: values[1:]

In []: **\_** # 9. *(Discuss and explain.)*

\_starts at the second value and lists

1. Unlike tuples, lists are **mutable:** you can reassign the value of individual elements in a list. Note, however, that using an element in an expression does not change its value.

In []: values[2] == 3

Out[]: True

In []: values[2] + 5

Out[]: 8

In []: values[2]

Out[]: 3

Predict, try, discuss, and explain the output for this input:

In []: values[2] = '3' *# Note the quotes!*

In []: values[2] == 3

In []: **\_** # 10. *(Discuss and explain.)*

\_the string ‘3’ and the integer 3 are different variable types, so it will return false

1. You can add onto a list with another list or append a new element. Predict, try, discuss, and explain the output for this input.

In []: values = values + [4, 5]

In []: values

In []: **\_** # 11a. *(Discuss and explain.)*

*adds 4 and 5 to the end of the list*

In []: values.append([6, 7])

In []: values

In []: **\_** # 11b. *(Discuss and explain.)*

*adds [6, 7] as a single value in the list*

If you are looking for a leap ahead, look at the official *Python* tutorial #5 at <https://docs.python.org/2/tutorial/datastructures.html>.

1. You cannot add onto a list using anything other than a list. Try this and explain:

In []: values = values + 5

In []: **\_** # 12. *(Explain why this doesn’t work.)*

\_ it doesn’t work because there are two different variable types

1. There is a shorthand for adding something to a variable and putting the result back in the variable.

In []: values += [5,6] *# values = values + [5,6]*

Hey! There’s the typing saved!

In []: values

Out[]: ['a', 'b', '3', 4, 5, 6]

This shorthand works for many variable types and operations. Predict, try, discuss, and explain the output for this input:

In []: a = 6

In []: a \*= 3

In []: a

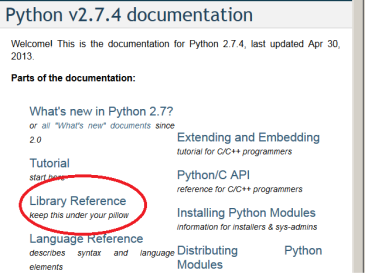
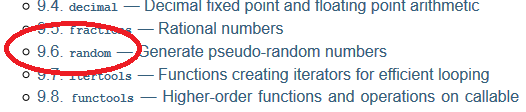
In []: **\_** # 13. *(Discuss and explain.)*

**Part III: Lists and the random module**

1. For a game, you might want a program to pick at random from a list. *Python* doesn’t have a built-in way to get random or pseudo-random numbers. Pseudo-random numbers are unpredictable but not actually random. The computer can use the system clock and some fancy arithmetic to generate a pseudo-random number.

Fortunately, someone already wrote the code for generating pseudo-random numbers and shared it. You only need to import it from a library included with Canopy. The module containing these programs is called “random”. It has many functions, as you can see in the “Library Reference” at [docs.python.org/2.7](http://docs.python.org/2.7).

Use the browser’s “find” [Ctrl+F] function to search for the keyword “random”. That will get you what you are looking for quickly. You’ll find documentation on the functions in the random module by following the link in category 9 as shown below. Or, just do a Google search for “python random” and choose the link to the official site.

In []: **import random**

Here are the three functions we want from the random module. Try each one several times to view random output. (Use the up arrow to enter them again.)

In []: random.choice(values) *# choose from a list*

In []: random.randint(5,8) *# choose int from [5,6,7,8]*

In []: random.uniform(5,8) *# choose float between 5 and 8*

1. Define a function roll\_two\_dice() that simulates rolling two six–sided dice and returns the total. Pair program, strategizing first.

In []: roll\_two\_dice()

Out[]: 7

**\_def rollTwoDice():**

**return random.randint(1,6) + random.randint(1,6)**

1. Define a function guess\_letter() that will pick one letter randomly from the alphabet.

**\_def guessLetter():**

**letters = "abcdefghijklmnopqrstuvwxyz"**

**n = random.randint(0,25)**

**return letters[n]**

Conclusion

1. Consider a string, tuple, and list of characters.

In []: a = 'abcde'

In []: b = ('a', 'b', 'c', 'd', 'e')

In []: c = ['a', 'b', 'c', 'd', 'e']

The values of a[3], b[3], and c[3] are all the same. In what ways are a, b, and c different?

A is a string, b is a tuple so it cannot be modified or mutated, and c is a list so it can be.

1. Why do computer programming languages almost always have a variety of variable types? Why can't everything be represented with an integer?

Some types are faster or easier to work with, or are more useful for some conditions.

