Python

Lab 03: Data Structures

We have demonstrated in lecture how lists, tuples, and dictionaries each provide a valuable way to store information. Through the activities in this lab, you will have the opportunity to explore the capabilities and limitations of these data structures.

Exercise 1: Summing Numbers
Ask the user to input a number greater than 1000. Allow the users input to remain a string. If the number is less than 1000 tell the user so. If the number is greater than 1000, return the type of each digit as well as the sum of all the digits. The output should look like the following:

```
>>>
Enter a number greater than 1000: 5
The string entered is 5
and is not greater than 1000
>>>
>>>
Enter a number greater than 1000: 2222
The string entered is 2222
digit is: 2 and is type: <type 'str'>
digit is: 2 and is type: <type 'str'>
digit is: 2 and is type: <type 'str'>
digit is: 2 and is type: <type 'str'>
Sum of the digits in 2222 is: 8
                                    --
```

Exercise 2: Casting to Lists & Tuples
Given tuple:

stuff = (1, 4, ‘apple’, ‘ziggle’)

- Cast the tuple into a list
- Change 4 into ‘banana’
- Cast the list back into a tuple
- Show the type

The output should look like the following:

```
>>>
stuff is (1, 4, 'apple', 'ziggle')
I cast it to a list - now it is of type <type 'list'>
now stuff is: (1, 'banana', 'apple', 'ziggle')
which is of type <type 'tuple'>
>>>
```

Exercise 3: Tuple Unpacking and Repacking
Given tuple:

test_tuple = (4, 8, "rainbow", "bright")

- Print test_tuple
- Unpack test_tuple into 4 variables
- Reassign the variables and repack the test_tuple so that test_tuple = (16, 512, 'bright rainbow', 'unicorns')

The output should look like the following:

```
>>>
test tuple is: (4, 8, 'rainbow', 'bright')
4
8
rainbow
bright
test tuple is: (16, 512, 'bright rainbow', 'unicorns')
>>>
```

Exercise 4: Dictionary Exercise
Given a dictionary of people and their favorite colors:

people = {'Samidh': 'Blue', 'Lisa': 'Yellow', 'Aaron': 'Purple', 'Jenny': 'Pink' }

- Find out how many people are in the list
- Change Lisa's favorite color
- Add Michael and his favorite color
- Find out if Samidh is included in the list of people
- Remove Aaron and his favorite color
- Sort and print people and their favorite colors, alphabetically by first name

The output should look like the following:

```
>>>
There are 4 people
Samidh is in the list of people!
Jenny; favorite color: Pink
Lisa; favorite color: Green
Michael; favorite color: Red
Samidh; favorite color: Blue
>>>
```

Challenge Exercise: Lists & Queues

1. Messi and Xavi lost their 2011 Champions League trophy while they were out celebrating last night. Now, they have to search through the streets of Barcelona to get it back. Starting from Bus Stop A (Messi's home) they want to check all the bus stops throughout the city.

   An efficient search strategy will maintain one data structure for nodes that have already been visited (`seen`) and another for the nodes to be visited (`to_visit`). You are given a dictionary that maps each bus stop name (strings) to the list of strings representing adjacent bus stops. For example, the bus network in **Figure 1** would be represented with the following adjacency dictionary:

   ```
   adjacency_dict =
   {'A':['B','C'],'B':['A','D','E'],'C':['A','F','E'],'D':['B'],'
   E':['B'],'F':['C']}
   ```

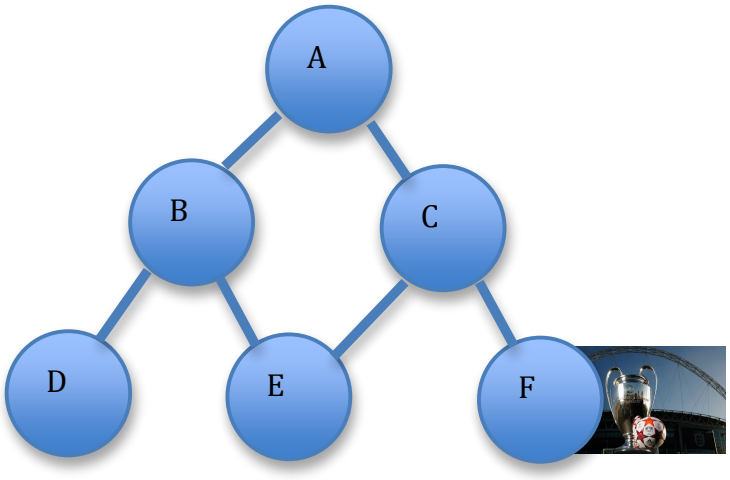

Figure 1: Sample layout of bus stops and connections with trophy at Bus Stop F

Messi and Xavi propose slightly different variations on the following strategy:

At the current bus stop, check for the trophy. If it is there, the quest ends (`return`)! Otherwise, remove the current bus stop from `to_visit`. Get the list of adjacent bus stops from `adjacency_dict`. Add each adjacent bus stop that is not in `seen` to the end of `to_visit`. At this point:

- Messi proposes that they proceed to the first item in `to_visit`.
- Xavi proposes that they proceed to the last item in `to_visit`.

a. What data structure should Messi and Xavi use to store whether or not a node has been `seen` (already visited in the search)? How does this data structure minimize lookup time?

b. Suppose Messi and Xavi decide to use lists to maintain the set of nodes `to_visit`. Whose algorithm will find the trophy fastest, and why?

c. Examine the python documentation on queues. Whose strategy would benefit most from using a queue in their `to_visit` data structure, and why?

Notes
See documentation on queues at:
http://docs.python.org/tutorial/datastructures.html#using-lists-as-queues

Image credit for trophy picture in Figure 1:
Getty Images, http://www.uefa.com/uefachampionsleague/news/newsid=1633420.html