MIT AITI

Python Software Development

Lab 04: 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. At the conclusion of this lab, you will know how to choose the right data structure for your software development needs!

1. Steven and John have just finished up a long afternoon of football and decide to head to the grocery store. They have a list of groceries that they would like to pick up at the market. Their list, `groceries`, is initialized as follows:

```
groceries = ['bananas','strawberries','apples','bread']
```

   a. They want to celebrate their victory and add champagne to the end of their original grocery list. Write the code to modify `groceries` accordingly.

   b. John decides he doesn't need bread. Write the code to remove this from `groceries`.

   c. The store has 26 aisles, labeled 'a', 'b', 'c', … 'z' from the left side of the store to the 'right' (apples are found in the 'a' aisle, strawberries in the 's' aisle, etc.). What operation could John perform on the list to make it easier for him to find the items he needs in the store? Write the code below. (Hint: we want to *map* from food item to aisle)

2. The store wants to design a catalog of all items in stock and their prices.
   a. What data structure would you choose to store this information and why?

b. Prices at the store are shown in the table below; write code to store this information in the data structure you chose in part (a).

Item	Price
Apples	7.3
Bananas	5.5
Bread	1.0
Carrots	10.0
Champagne	20.90
Strawberries	32.6

c. The price of strawberries goes up in the winter to 63.43; how would you modify the price in your data structure?

d. Soccer players insisted on more protein options for their diets, so the store decided to sell 'Chicken' at a rate of 6.5. Write the code to add this information to the data structure from part (c).

3. The Shoprite CEO has decided that he wants to advertise a list of all the foods that his store sells. He asks the computer scientists at his company to do this for him.

a. Describe the data structure that would best fit this data.

b. Given the data structure chosen in part a, use the data structure from Question 2 to create the collection of items that will be sent to the CEO.

```
in_stock = #your code here
```

Later, before the advertisement runs, the CEO decides that he doesn't want this list of items to change, *ever*. What is another type of sequence in python, besides list, that cannot change?

c. Convert the data structure `in_stock` to the immutable `always_in_stock`.

d. Ok, now the advertisement needs to run! Write code that will print out the message

"Come to shoprite! We always sell:"

And then print out each element of the sequence `always_in_stock`, one per line.

4. Challenge Problem: Combined data structures

Steven and John are outraged at the prices in this store; they want to check around at a few other stores. For example, apples cost SOME_PRICE at the current store and SOME_LOWER_PRICE at another store.

a. What data structure could they use to store different market prices associated with all the items on their grocery list? (Hint: dictionaries can be inside dictionaries, lists inside lists, lists inside dictionaries… etc…)

b. Given a sorted list of prices (e.g., $0.50, $1.25, $1.50), design a function that will insert another price into the list. Maintain the price order without re-sorting the entire list (hint: use binary search).

```
def binary_insert(new_float,some_list_of_floats):
     #modifies the input list to include the new_float

     return
```

c. Write a function that returns the minimum amount of money that Steven and John will have to spend on their grocery list.

```
def min_cost(grocery_list,item_to_price_list_dict):
     #grocery_list is a list of strings (item names)

     #item_to_price_list_dict is a dictionary with key-value
     # pairs as follows: the item name (strings) is the key
     # and the list of prices (floats) at different grocery is
     # the value

    return
```

5. **Challenge Problem: Lists and Queues**
John and Steven lost their phone while they were out drinking champagne last night. Now they have to search through the streets of Accra to get it back. Starting from Bus Stop A (Steven's home) they want to check all the bus stops throughout the city.

An efficient search strategy will maintain one data structure for bus stations (call them nodes) that have already been visited (`seen`) and another for the nodes (bus stations) 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','D','E'],'D':['B','C'],
'E':['B','C','F'],'F':['E']}
```

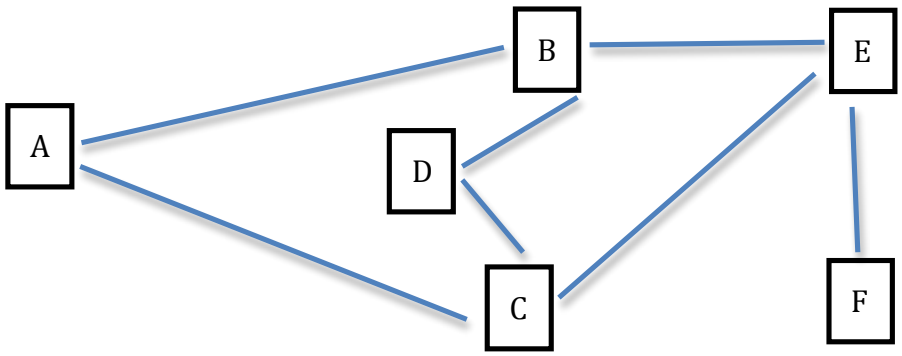

Figure 1: Sample layout of bus stops and connections with phone at Bus Stop F
(John has a very old cell phone)

John and Steven 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:

- John proposes that they proceed to the first item in `to_visit`.
- Steven proposes that they proceed to the last item in `to_visit`.

a. What data structure should John and Steven 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 John and Steven 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?