Week 8 Take-Home Exercises

1) For loops vs while loops

Convert the following while-loops into for-loops:

While-Loop *For-Loop*

1.

```
i = 0
while i < 10:
    print(i)
    i += 1
```

2.

```
boros = ["bronx", brooklyn",
         "manhattan", "queens",
         "staten island"]
i = 0
while i < len(boros):
    print( boros[i] )
    i += 1
```

3.

```
str = "To be or not to be"
i = 0
num_spaces = 0
while i < len(str):
    if str[i] == ' ':
        num_spaces += 1
    i += 1
print("num spaces found: " + num_spaces)
```

2) Online Grocery Store

Imagine you are starting an online grocery store, and are writing code so that users can search the inventory.

Currently your website just presents users with the whole unsorted list, but users find it hard to navigate.

```
my_inventory_list = [
  "orange juice",
  "apple”,
  "mango sorbet",
  "strawberries (4 oz)",
  "milk",
  "chocolate milk",
  "capn crunch",
  "blueberries (4 oz)",
  "doritos",
  "fritos",
  ...
  "cheetos",
  "raisin bran",
  "asparagus",
]
```

Alphabetic Search

Write the function ‘search_list_by_letter’ that takes two parameters: an inventory list (like the one above) and a letter; and returns a new list containing the items in the inventory that start with that letter.

```
def search_list_by_letter( inventory_list, letter ):
    # Your code here
```

```
# Testing the function – prints a list of items that start with “a”
print( search_by_letter(my_inventory_list, “a”) )
```

Lists of Lists

In order to store more data about each product we can store each product as a list instead of a string:

```
[ <name of product>, <price>, <category>, <# in inventory> ]
```

Orange juice would be stored as the list:

```
item = [ “orange juice”, 4.99, “BEVERAGE”, 10 ]
```

This says that the item’s name is “orange juice”, it costs $4.99, it is a “BEVERAGE”, and there are 10 in the inventory.

Your whole inventory is now represented as a list of lists:

```
my_inventory_list = [
  ["orange juice", 4.99, "BEVERAGE", 10],
  ["apple", 0.79, "FRUIT", 3],
  ["mango sorbet", 3.89, "ICECREAM", 5],
  ["strawberries (4 oz)", 3.99, "FRUIT", 20],
  ["milk", 4.99, "BEVERAGE", 10],
  ["chocolate milk", 6.79, "BEVERAGE", 15],
  ["capn crunch", 3.39, "CEREAL", 5],
  ["blueberries (4 oz)", 3.49, "FRUIT", 1],
  ["doritos", 1.99, "SNACK", 20],
  ["fritos", 1.99, "SNACK", 10],
  ...
  ["cheetos", 1.79, "SNACK", 15],
  ["raisin bran", 3.39, "CEREAL", 2],
  ["orange", 1.49, "FRUIT", 20]
]
```

Example: Searching For Cereal

Many cereal-lovers come to your site looking specifically for cereal. It turns out you have one of the best cereal selections on the web!

Write a function called ‘cereal_in_inventory’ that takes an inventory list as a parameter, and returns a list of cereals from that inventory.

```
def cereal_in_inventory( inventory ):
    cereals_found = []
    for item in inventory:
        if item[2]==”CEREAL”: # the price is stored at index 2 of the item
            cereals_found.append(item)

    return cereals_found

print( cereal_in_inventory( my_inventory ) )
# prints a complete list of cereal products
```

What’s Low in Stock?

Your employees want to more easily see what is low in stock so that they can place orders to refill those items. Write a function called ‘low_items_in_inventory’ that takes an inventory list as a parameter, and returns the items that have a quantity of 3 or less.

```
def low_items_in_inventory( inventory_list ):
    # Your code here
```

```
# Test the function - should print list of items low in inventory
print( low_items_in_inventory( my_inventory_list ) )
```

The Price Search Function

Write a function that takes three parameters: a minimum price, a maximum price, and an inventory list, and returns a list of the items that are within that price range.

```
def filter_inventory_by_price( inventory_list, min_price, max_price ):
    # your code here
```

```
# Test the function - should print list of items costing between $2 and $7
print( filter_inventory_by_price( my_inventory_list, 2, 7 ) )
```