Lecture 20 – Completing all the slides which were not looked at during lectures 15 to 19

Lecture 19 – slides 16 – 19
Lecture 18 – slides 17
Lecture 16 – slides 12
Lecture 17 – slides 11
Lecture 18 – slides 17
Online shopping – update the quantity

To update the quantity of an item (a string), we need to add/subtract to/from the information at position 3 of the string. The information needs to be converted into an int, the amount added, and, the changed string needs to be assigned to the correct index of the item_list. E.g., the code:

```python
energy_index = 2
update_quantity(items_list, energy_index, 5)
```

changes:

"bc003,V-energy drink,2.75,9"

into:

"bc003,V-energy drink,2.75,14"
Finally, the changed string:

"bc003,V-energy drink,2.75,14"

needs to be assigned to the correct index of the item_list.
Online shopping – updating the quantity of an item on the stock list of items

- When an item is added to (or removed from the shopping cart), its quantity value needs to be updated in the list of stock, `items_list`, e.g., for the following item:

  `'bc003,V-energy drink,2.75,9'`

9 is the quantity, i.e., the number of this item currently in stock.

When item, 'bc003' is bought (added to the shopping cart), the quantity will decrease by one.

Note: in this program we are assuming that the user only buys an item if there is at least one of the item in stock.
Online shopping – updating the quantity of an item on the stock list of items

Complete the update_quantity() function. Note that the quantity should never be less than 0.

```python
def update_quantity(items_list, index, update_amt):
    ...
```

Note: when an element from the items_list is split into a list of its parts (comma separator), the quantity of the item is in position 3 of the list.

```python
def main():
    ...
    elif selection == 2:
        ...
        index = find_item_index(items_list, barcode)
        if index > -1:
            user_item = items_list[index]
            cart_list.append(user_item)
            update_quantity(items_list, index, -1)
    main()
    ...
Complete the function

- Complete the `write_to_file()` function which writes the elements of the two parameter lists (one element from both files per line) to the file (given by the parameter, `filename`). The elements are separated by ": ".

```python
def write_to_file(filename, list1, list2):

def main():
    a_list1 = [2, 4, 5, 6, 8, 1]
    a_list2 = [123, 54, 58, 106, 87, 206]
    filename = "combined_lists.txt"
    write_to_file(filename, a_list1, a_list2)

main()
```

Assume the two files have exactly the same number of elements and that each element is an integer.
Assigning a list object to a variable

- Python lists are **objects**. When an object is assigned to a variable, the reference (the address) is **copied** and stored in the variable.

```
list1 = [1, 2, 3]
list2 = list1
list3 = [1, 2, 3]
print(list1)
print(list2)
print(list3)
print()
print(list1)
print(list2)
print(list3)
```

```
[1, 2, 3]
[1, 2, 3]
[1, 2, 4]
[1, 2, 5]
[1, 2, 5]
[1, 2, 4]
```

Diagram: 
- **list1**
  - 0: 1
  - 1: 2
  - 2: 3
- **list2**
  - 0: 1
  - 1: 2
  - 2: 3
- **list3**
  - 0: 1
  - 1: 2
  - 2: 3
Do the following two sections of code give the same output? If not, what is the difference in output?

**Code A**
```
list1 = [1, 2, 3]
list2 = list1

for index in range(len(list1)):
    list2[index] = list1[index] * 2

print("1.", list1)
print("2.", list2)
```

**Code B**
```
list1 = [1, 2, 3]
list2 = [1, 2, 3]

for index in range(len(list1)):
    list2[index] = list1[index] * 2

print("1.", list1)
print("2.", list2)
```
Lists are "mutable", i.e., the elements in a list object can be updated and adjusted.

```python
list1 = [10, 20, 30, 40, 50]
list2 = [1, 5]  # A
print("1.", list1)
print("2.", list2)
print("3.", list1 is list2)

list2 = list1    # B
print("4.", list1 is list2)

list1[3] = 99
list2[1] = 3    # C
print("5.", list1)
print("6.", list2)
print("7.", list1 is list2)
```

1. [10, 20, 30, 40, 50]
2. [1, 5]
3. False
4. True
5. [10, 3, 30, 99, 50]
6. [10, 3, 30, 99, 50]
7. True
```python
def fiddle_tuples(t1, t2):
    t3 = (t1[1], t2[0])
    t1 = (t3[1], t2[0])
    t2 = t1
    return t3

def main():
    t1 = (3, 5)
    t2 = (4, 7)
    t3 = t2
    t3 = fiddle_tuples(t1, t2)
    print("t1:", t1, "t2:", t2, "t3:", t3)
main()
```