ICS3U Case Study: Inventory System

In this case study, we will examine a simple program that can keep track of a company's inventory of items. Most modern systems use more complicated systems involving databases and graphical interfaces, but the main ideas are the same. The full code is below (sans comments).

```
import math

def main_menu():
    print("[A]dd item")
    print("[D]elete item")
    print("[E]dit item stock")
    print("[S]how all items")
    print("[V]iew item info")
    print("[Q]uit")
    action = input("What would you like to do? ").upper()
    while action not in ("A", "D", "E", "V", "S", "Q"):
        action = input("That is not a valid option, try again: ").upper()
    return action

def get_integer(low=-math.inf, high=math.inf, prompt="Enter a value: "):
   integer = input(prompt)
   while not integer.isdigit() or int(integer) < low or int(integer) > high:
      integer = input("Invalid value, try again: ")
   return int(integer)

def find_item(name, inventory):
    for idx in range(len(inventory)):
        if inventory[idx][0].upper() == name.upper():
            return idx
    return -1

def view_item(name, inventory):
    idx = find_item(name, inventory)
    if idx == -1:
        print("Item not found.")
    else:
        print("{:<15} {}".format("Item:", inventory[idx][0]))
        print("{:<15} {}".format("Product Code:", inventory[idx][1]))
        print("{:<15} {}".format("Stock:", inventory[idx][2]))

def show_all(inventory):
    if len(inventory) == 0:
        print("Inventory is empty.")
    for item in inventory:
        view_item(item[0], inventory)

def add_item(name, inventory):
    if find_item(name, inventory) == -1:
        code = input("Enter product code: ")
        stock = get_integer(low=0, prompt="Enter stock quantity: ")
        inventory.append([name, code, stock])
    else:
        print("Item already exists.")
        view_item(name, inventory)
    return inventory
```

```
def delete_item(name, inventory):
    idx = find_item(name, inventory)
    if idx == -1:
        print("Item not found.")
    else:
        inventory.pop(idx)
    return inventory

def edit_stock(name, inventory):
    idx = find_item(name, inventory)
    if idx == -1:
        print("Item not found.")
    else:
        stock = get_integer(low=0, prompt="Enter new quantity: ")
        inventory[idx][2] = stock
    return inventory

inventory = []
print("Welcome to the ABC Corporation's Inventory System")
while True:
    action = main_menu()
    if action == "Q":
        break
    elif action == "S":
        show_all(inventory)
    else:
        name = input("Enter product name: ")
        if action == "A":
            inventory = add_item(name, inventory)
        elif action == "D":
            inventory = delete_item(name, inventory)
        elif action == "E":
            inventory = edit_stock(name, inventory)
        elif action == "V":
            view_item(name, inventory)
print("Goodbye.")
```

Let's examine the code in more detail. The first two functions, main_menu and get_integer, are similar to ones that we have seen before (see the *Surface Area Case Study* for more information) so we won't examine them in much detail here, other than to say that they display the user actions and obtain an integer from the user respectively.

The `find_item` function is used extensively in the program, in order to avoid duplicate entries and to avoid crashes when trying to remove an item which dies not exist in `inventory`.

```
def find_item(name, inventory):
    for idx in range(len(inventory)):
        if inventory[idx][0].upper() == name.upper():
            return idx
    return -1
```

Given an item's name, the function searches through the `inventory` list one-at-a-time until it either locates the item, or reaches the end of the list. If the item is found, its index is returned to the main program for further action. Otherwise, a value of -1 is returned as a code for the programmer. The search is case-insensitive, using `upper` to match the name to an item.

The `view_item` function displays all information about an item given its name.

```
def view_item(name, inventory):
    idx = find_item(name, inventory)
    if idx == -1:
        print("Item not found.")
    else:
        print("{:<15} {}".format("Item:", inventory[idx][0]))
        print("{:<15} {}".format("Product Code:", inventory[idx][1]))
        print("{:<15} {}".format("Stock:", inventory[idx][2]))
```

First, it calls `find_item` to verify that the item exists in `inventory`. If it does not, a message is displayed. Otherwise, `format` is used to display the information in a nicely-formatted manner.

The `show_all` function uses view_item to display all items in `inventory`. If the list is empty – that is, if `len(inventory)` is zero – then a message is displayed instead.

```
def show_all(inventory):
    if len(inventory) == 0:
        print("Inventory is empty.")
    for item in inventory:
        view_item(item[0], inventory)
```

Adding an item to `inventory` is handled by the `add_item` function.

```
def add_item(name, inventory):
    if find_item(name, inventory) == -1:
        code = input("Enter product code: ")
        stock = get_integer(low=0, prompt="Enter stock quantity: ")
        inventory.append([name, code, stock])
    else:
        print("Item already exists.")
        view_item(name, inventory)
    return inventory
```

If the item does not already exist (indicated when `find_item` returns a value of -1) then the user is prompted to enter the details. Note the call to `get_integer` for `stock`. There is no verification for `code`, but you can add it if you like. The updated `inventory` is returned to the main program.

The `delete_item` function uses `find_item` to locate the index of an item in `inventory`. If the item is not found (`idx` is -1) then a message is displayed. Otherwise, the item is removed using `del`. Like add_item, the updated `inventory` is returned.

```
def delete_item(name, inventory):
    idx = find_item(name, inventory)
    if idx == -1:
        print("Item not found.")
    else:
        del(inventory[idx])
    return inventory
```

The final function, `edit_stock`, allows the user to change an item's quantity.

```
def edit_stock(name, inventory):
    idx = find_item(name, inventory)
    if idx == -1:
        print("Item not found.")
    else:
        stock = get_integer(low=0, prompt="Enter new quantity: ")
        inventory[idx][2] = stock
    return inventory
```

Like most other functions, `find_item` is called to verify that the item exists in `inventory`. If the item exists, the user is promted to enter a new quantity using `get_integer`, and this new value overwrites the previous value. There is no functionality to update an item's name and code, but this could be implemented using a similar technique. The updated inventory list is returned.

The last part to examine is the main program itself. This is largely a loop that runs until the user chooses to quit, and a series of `if` statements that call particular functions based on the user's desired action.

```
inventory = []
print("Welcome to the ABC Corporation's Inventory System")
while True:
    action = main_menu()
    if action == "Q":
        break
    elif action == "S":
        show_all(inventory)
    else:
        name = input("Enter product name: ")
        if action == "A":
            inventory = add_item(name, inventory)
        elif action == "D":
            inventory = delete_item(name, inventory)
        elif action == "E":
            inventory = edit_stock(name, inventory)
        elif action == "V":
            view_item(name, inventory)
print("Goodbye.")
```

At the start of the main program, `inventory` is empty (indicated by an empty list). As the user adds or removes items, `inventory` will grow and shrink as necessary. Unfortunately, there is no code to preserve the items in `inventory` once the program has ended, and the user must enter each time it is run. We can change this by using files to store the information, from which we can read and write; however, this will have to wait until a future lesson.