CSCI 204 – Introduction to Computer Science II

Project 1 – E-Commerce Inventory

Assigned: Wednesday, 8/26/2020
First phase due: Friday, 9/4/2020
Second phase due: 9/18/2020

1 Academic Integrity

The project falls under the Homework and Projects in the course collaboration policy. Make sure you follow these policies. You are able to work with a partner, but must pair program (and not, e.g., "you do this I'll do this").

2 Objective

The main objective of this project is to master the design and implementation of objects and classes with inheritance, and to extend your programs to run through a web interface.

3 Problem Statement

You are just hired by *Samszone*[1], a start-up e-commerce company to help them implement their on-line inventory system. The company plans to sell various items over the web. The types of goods the company is planning to sell can be divided into books, CDs and vinyls, collectables, and eBay items. Each type of the items contains different types of sales information, e.g., a book would have a title and author(s), among other information, while a fashion item would have a product name and manufacturer. The information about these items are stored in a collection of CSV files (Common-Separated Values).

Your tasks are to first design a Python class hierarchy that properly reflects the nature of these items, then you are to write Python programs that can manage these items over the web that allows a user to search for a particular item, to query other information such as total value of the inventory, or to print the information of a collection of item based on their class, e.g., all books or all collectables.

4 Your Tasks

You are given a set of CSV files that contain information for all items for sale. You are also given a collection of test programs to run in Linux command line, or to run within your favorite Python IDE (I suggest Spyder). You will design and implement the programs such that these test programs will run properly. This will be your **first phase** of the project. Once these test programs run properly, which means your class implementation is most likely correct, you will extend your programs to run over the web. This will be your **second phase** of the project. You are also given a set of sample web server-client programs to demonstrate how a set of Python programs can interact over the web.

[1] This fortuitous name comes out of the 204 archives and is not due to Prof. Sam Gutekunst.

4.1 Phase One: Design and Implement the Inventory Class

You are given a set of test programs and data files, available on Moodle. Let's concentrate on the CSV files first: your program will read data from these files and create the appropriate objects to build its database

The following set of CSV files are included. These files will give rise to a total of four general classes of items in the database: Books, CD/Vinyl, Collectibles, and eBay items. The eBay items will be divided into three sub-classes, Electronics, Fashion, and Home/Garden items.

File name	Content
book.csv	A collection of 10 books with title, publishing date, publisher, author, unit price, ISBN, and quantity or count in the database.
cd_vinyl.csv	A collection of 10 CDs or vinyls with title, artist(s), label, ASIN, date, unit price, and quantity.
collectible.csv	A list of seven collectibles with title, unit price, date, owner, and quantity.
electronics.csv	A collection of nine items with name, unit price, date, manufacturer, and quantity.
fashion.csv	Same type of information as in the 'electronics.csv.'
home_garden.csv	Same type of information as in the 'electronics.csv.'

Table 1: Description of the CSV files

These are the data files you will work with. Note that the amount of data is very small. But your programs should be able to work correctly with data of any size: the logic is the same.

You should also see a few Python programs. You will need the programs `simple_web_server.py` and `test_inventory_web.py` in the second phase of the project. For now, first examine the program `test_inventory.py`. From the `test_inventory.py` program you can tell that your Inventory class needs to have the following public interface that consists of the methods that a public program can use.

Method name	Description
Inventory()	Constructor
check_type()	Return the type of object, using isinstance()
compute_inventory()	Compute the total value of all items in the inventory
print_inventory()	Print the information of the inventory
print_category()	Print information by categories of the items
search_item()	Search items whose name contain the given pattern

Table 2: Methods of the Inventory class that are accessible by the public

The text file `output.txt` is the direct result of executing `test_inventory.py`. Your programs, when completed, should generate a similar, if not identical result. Pay careful attention to this file and the output, as this will guide how you create the methods above.

Design your Inventory class, Item class, and subclasses of the Item class

Now that you have a sense how a user may use the class through programs such as `test_inventory.py`, you are asked to design and implement the Inventory class. While object-oriented programs and design are not the concentration of this course (you will see it in CSCI 205), you are asked to follow some basic approaches.

First, draw a diagram of class indicating the relations among the classes and subclasses. Remember the key feature in this design is to minimize the redundancy and to facilitate the future expansion of the class. Follow the pattern you learned in your lab work. Draw the diagram either using a graphics tool such as `xfig` on Linux or Paint on Windows. Figure 1 is the Counter class hierarchy we saw in our lab.

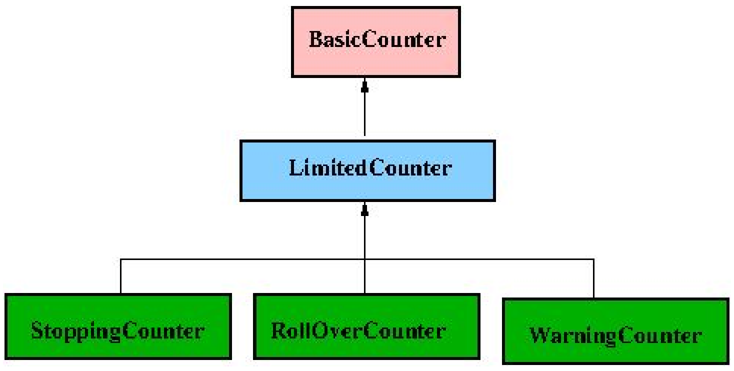


Figure 1: The Counter class hierarchy used in the lab

While the Inventory itself might not need any subclasses, consider the items that are in an inventory. Read the CSV files and their descriptions in Table 1. Identify what are the common features in these items that should be included in a base Item class. Specific features in items such as Book, Collectible and others indicate that they may belong to a subclass of the Item class.

Implement your classes

Once the classes are designed, implement them in Python. You are asked to implement the Inventory class, an Item class, and all subclasses of the Item class. Put all the implementation in one file. Name the file `inventory.py` to make our testing work.

You may have certain degrees of freedom in how to design and implement these classes. You do have to pay attention to the following notes.

1 The Inventory class should build one Python list storing all items. The `__init__` function should read in information from the 6 CSV files listed in Table 1 (possibly using other methods of the Inventory class). The test_inventory.py file will help you determine which attributes to include in Inventory class, and the sample output will guide some of the methods and their output.

2 To read and parse CSV files, use the existing Python package named `csv` (yes, the exact same name as the type of the CSV files.) In the files you copied for this project, you should see one named `csv2list.py` which reads a CSV file and converts the contents into a list of lists. Use the component in this example, you can write a method for your Inventory class to read all the CSV data files.

3 The `__init__` method for your Inventory should not be overwhelmingly long. You may want to add other methods to facilitate a concise `__init__`: perhaps a method that reads in files (e.g. a `read_items` method that takes in a file name and possibly a type), and method(s) that add individual types of items.
4 The CSV files should guide the attributes and `__init__` method for your Item class and its subclasses.
5 Python has a function named `isinstance()` that returns True if a variable belongs to a given class. You should use this function to implement your `check_type()` method.
6 When implementing class methods, make sure they are cohesive and loosely-coupled. Cohesive means only relevant and closely related actions should be put in one method; loosely-coupled means reduce or eliminate inter-dependency between methods. Don't make any methods too long. If an action needs many steps to complete, consider splitting it into multiple methods.
7 You can assume everything is public in the interface for this project. E.g. you do not need to begin attribute names with an underscore, or have separate getter/setter methods.

Test your implementations

Test your implementations often. You should test at each step after completing a method or an object. You can comment out sections of code in `test_inventory.py` to test one component at a time. For example, comment everything out except the constructor part

```
invent = Inventory()
```

that allows you to test if you constructor works. Of course, when completed, you must run the original given test program `test_inventory.py` to show that everything works fine.

Getting Started

Think about how you want to approach building the Inventory, Item, and Item subclass structure. Try to break the task into small, easily testable pieces. You might, for example, start by creating the Item class and subclasses of Item, similar to what you did in Lab, creating `__init__` and `__str__` methods. Test that they work: that you can create and print items of different types. You don't need any extra methods within each Item/Item subclasses beyond the `__init__` and `__str__` methods.

Then think about how to read in data from one of the spreadsheets. How can you read it in, parse it, and create the appropriate objects? Do you need method(s) that help you read the file? Once the `__init__` Inventory class method is working on a single type of item, which methods from Table 2 can you write? Can you, e.g., compute or print the inventory? Do those methods work correctly on the single sheet you are processing? Then what's left to do?

Final Phase 1 Notes:

- 18 of the points on this project (between the two phases) are for good Python style. Please read the course Style Guide handout for details about what we're looking for.

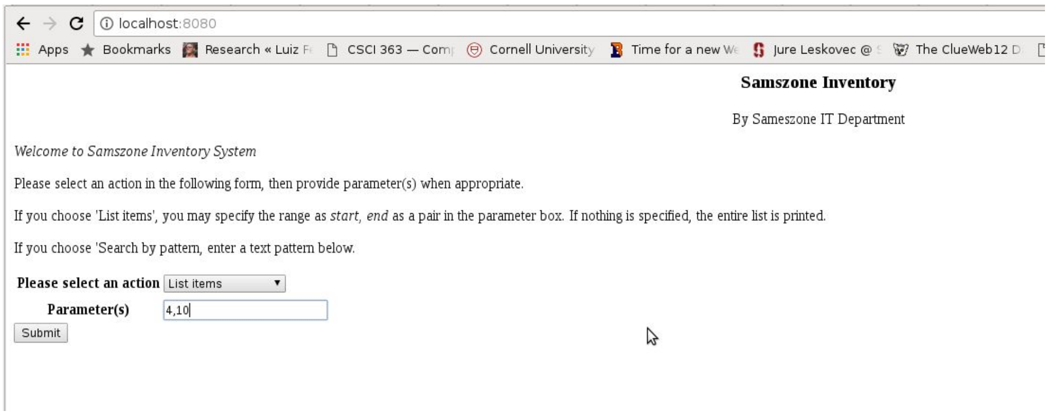
- 9 of the points (between the two phases) are for program quality and features that stand out. Think about what features you can add that will help your work stand out. You might think about adding additional features to your inventory, or later, the design and features of your web interface.

4.2 Phase Two: Make Your Inventory Accessible through the Web

Now that your program works fine in an IDE, you are to make the inventory accessible through the web like any real e-Commerce website! Through a web interface that you will implement, a user is able to issue many of the commands that you implemented in Phase One of the project. At a minimum, an user should be able to:

- List items in your inventory, either all items or by specifying the start and end ID numbers (see Figure 2 and 4). Asking for "4,10" in the Parameter(s) box, like in Figure 2, should display items with IDs 4-9.
- Compute and report the total value of the entire inventory. No parameters are needed. See Figure 4b.
- Search the inventory for a specific phrase (e.g. displaying all items that include "gar" by entering "gar" -- without quotes -- into the Parameter(s) box). See Figure 3.

Figures 2, 3 and 4 show some sample screen shots that illustrate the idea.



Samszone Inventory

By Sameszone IT Department

Welcome to Samszone Inventory System

Please select an action in the following form, then provide parameter(s) when appropriate.

If you choose 'List items', you may specify the range as *start, end* as a pair in the parameter box. If nothing is specified, the entire list is printed.

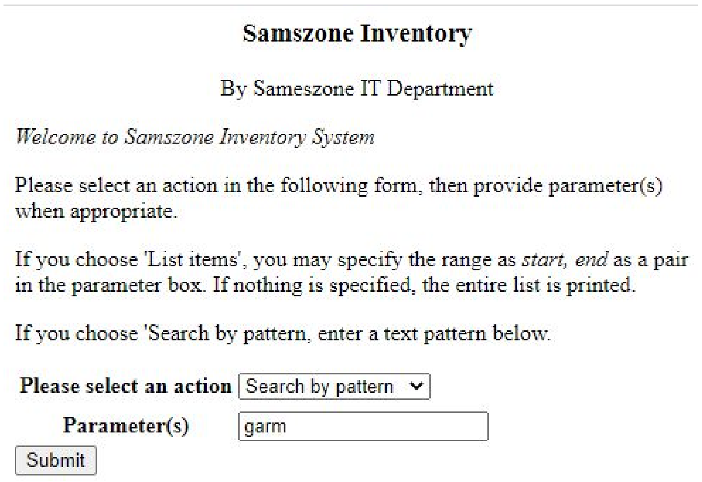
If you choose 'Search by pattern, enter a text pattern below.

Please select an action List items

Parameter(s) 4,10

Submit

Figure 2: Home page of Samszone website with the list option shown


Samszone Inventory

By Sameszone IT Department

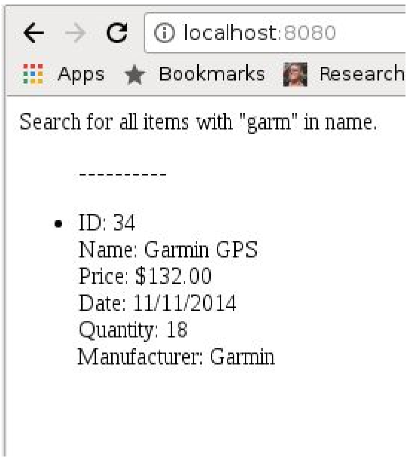
Welcome to Samszone Inventory System

Please select an action in the following form, then provide parameter(s) when appropriate.

If you choose 'List items', you may specify the range as *start, end* as a pair in the parameter box. If nothing is specified, the entire list is printed.

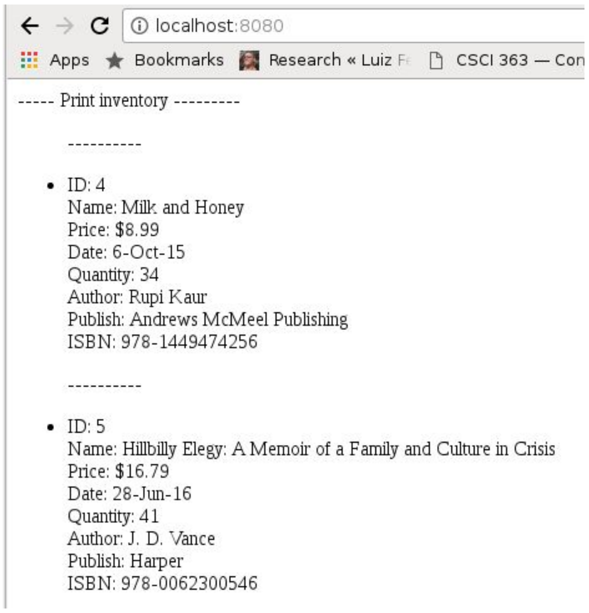
If you choose 'Search by pattern, enter a text pattern below.

Please select an action	Search by pattern
Parameter(s)	garm



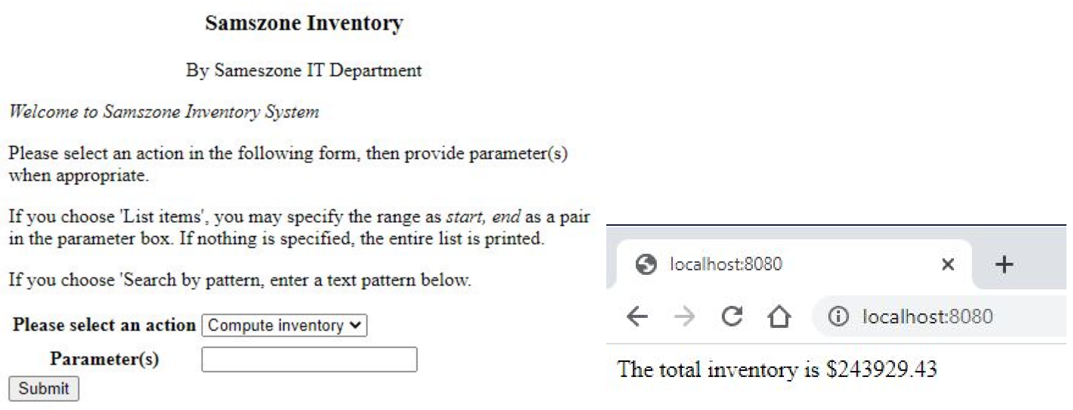
```
Search for all items with "garm" in name.

        ----------

    • ID: 34
      Name: Garmin GPS
      Price: $132.00
      Date: 11/11/2014
      Quantity: 18
      Manufacturer: Garmin
```

Figure 3: Search result for key phrase “garm” in names. Input is on the left, output on the right.



```
----- Print inventory ---------

        ----------

    • ID: 4
      Name: Milk and Honey
      Price: $8.99
      Date: 6-Oct-15
      Quantity: 34
      Author: Rupi Kaur
      Publish: Andrews McMeel Publishing
      ISBN: 978-1449474256

        ----------

    • ID: 5
      Name: Hillbilly Elegy: A Memoir of a Family and Culture in Crisis
      Price: $16.79
      Date: 28-Jun-16
      Quantity: 41
      Author: J. D. Vance
      Publish: Harper
      ISBN: 978-0062300546
```

Figure 4: Partial result of *listing* operation when asked with parameters 4,10


Samszone Inventory

By Sameszone IT Department

Welcome to Samszone Inventory System

Please select an action in the following form, then provide parameter(s) when appropriate.

If you choose 'List items', you may specify the range as *start, end* as a pair in the parameter box. If nothing is specified, the entire list is printed.

If you choose 'Search by pattern, enter a text pattern below.

Please select an action Compute inventory

Parameter(s)

Submit

localhost:8080

The total inventory is $243929.43

Figure 4b: Input and output for compute inventory

How to tackle the problem

Python provides easy-to-use, yet very powerful libraries for programming over the web. The basic idea is that a server program runs on the background (from command line or IDE) that supports the logic of the program. The server program **takes** input from any web browser, **processes** the request, **produces** the results, **formats** the results as a web page, then **sends** the page back to the web browser for display. While the details of web programming are not necessarily the focus of this project, we provide you with a sample Python server program. From the examples, you can figure out what takes place in web programming and you can implement your own logic for your task of creating a website for *Samszone*.

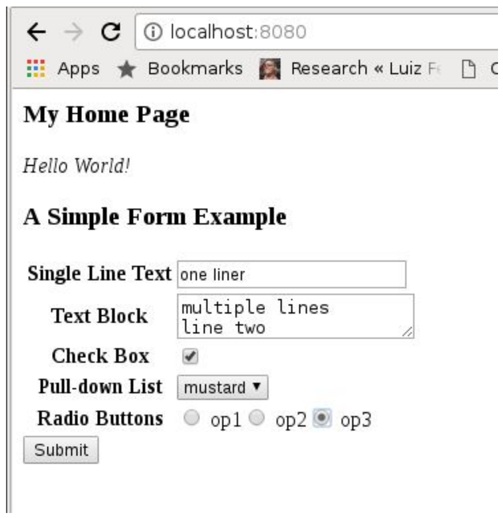
Try out the given server program

Among the files you copied for this project, you should see a Python web server program named `simple_web_server.py`. In addition, you should have two HTML files, `home.html` and `form.html`. Now in Spyder (or your IDE, or the linux command line) run the program `simple_web_server.py.` You should see something similar to the following on your terminal window or your IDE window.

```
[bash abc@host]$ ./simple_web_server.py
Sun Aug 13 10:23:55 2017 Server Starts - :8080
Starting httpd...
```

Figure 5: Start the simple web server from Linux command line

Now go to your favorite browser, and visit the page using the URL http://localhost:8080, you should see a web page similar to the following shown in Figure 6.



My Home Page

Hello World!

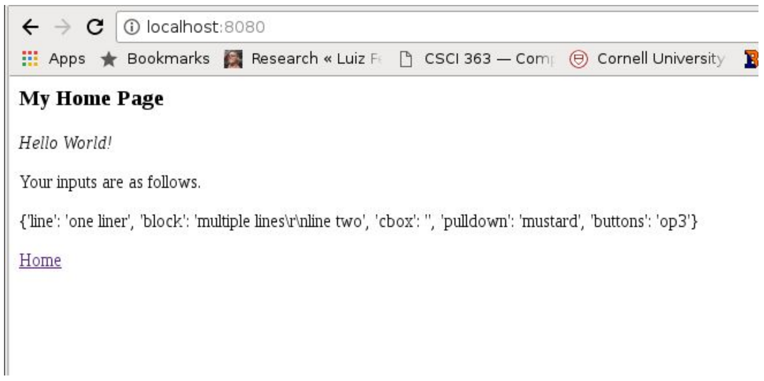
A Simple Form Example

Single Line Text	one liner
Text Block	multiple lines line two
Check Box	✓
Pull-down List	mustard
Radio Buttons	op1 op2 [✓ op3]

Submit

Figure 7: Home page with inputs already typed in

What happened was that the simple web server is running on the computer named `localhost` at the port 8080. The browser, e.g., Google Chrome or Mozilla Firefox, contacts the server through the HTTP protocol that both understand. Figure 7 shows the screen shot after the user (you) has typed some inputs, but not yet clicked the submit button yet. Once the `Submit` button is clicked, the information that was typed is sent to the server that is running on the Linux command line/IDE. After receiving this collection of information, the server can parse and process the information, then sends any responses back to the browser. Figure 8 shows the responses that the `simple_web_server.py` sends back: it echoes what the user typed in.



My Home Page

Hello World!

Your inputs are as follows.

```
{'line': 'one liner', 'block': 'multiple lines\r\nline two', 'cbox': '', 'pulldown': 'mustard', 'buttons': 'op3'}
```

Home

Figure 8: The response page generated by the server

One can see from Figure 8 that the information typed in by the user, such as "`one liner`" in Figure 7 or the click box or the pulldown menu, are sent back by the server and displayed in the web page.

Study how the interaction was generated

Run the program multiple times with different inputs so you have a sense what happens and is displayed. Now let's study the logic behind the program `simple_web_server.py`. Starting from the bottom of the file, you will see that the program starts by the line `run(port=HOSTPORT)` which calls the function **run()**. Basically the server starts running and waits for input from a *client* which can be any program that sends requests to this server. In our case, the client will be any web browser. You can then start a web browser, and point the URL to be something like http://localhost:8080, as in Figure 7. Once you type the information similar to that in Figure 7 and click the `Submit` button, the browser sends all the information you typed to the server through the internet. The server program (`simple_web_server.py`) handles the request from the client, in our case, the browser using the following default functions.

Function `do_GET()`

The function `do_GET()` is automatically called when the user hits the `Return` key with the URL http://localhost:8080 in the browser. Read the code in the function, what the server does is to send back a *home page* to the browser for it to display. The home page is a text in HTML format. The content of the home page is stored in two text files, `home.html` and `form.html`. The functions `generate_home_page()` and `generate_form()` read these two files and store the contents in two variables `home_page` and `form_text` for other functions to use. Note that the content of the `form_text`, when displayed on a browser gives the user a form to fill out. Among other information, the form contains a `Submit` button specified by the `<input type=”submit” />` HTML phrase. When the user click this submit button, the content of the form is sent from the browser to the server. The server will process the information using its `do_POST()` function.

Function do_POST()

The function do_POST() is called when the information sent from the browser contains an HTML phrase `<form name="main" method="post">`. What the function does is to extract the information from the form data, process it, and send the result back to the browser. Read the code `do_POST()` you will find that after reading the content of the form, the function calls another function `convert_code()` to convert HTML data into plain text. When the information is sent from a browser to a web server, some special texts are coded as hexadecimal or other special character to avoid confusion. For example, the space character ‘ ‘ is coded as a ‘+’, and the character ‘+’ itself is coded as hex 2B proceeded by a ‘%’, that is the HTML code ‘%2B’ represents the plus sign ‘+’. See the website at http://www.ascii.cl/htmlcodes.htm for a complete list. In the `simple_web_server.py` program, the function `do_POST()` simply returns the content of the form after being parsed out to the browser.

Your tasks

For this phase of the project, you will largely restrict yourself to modifying do_POST(), the form/home html piles, and perhaps `do_GET()` to make them behave as you needed for the *Samszone* website. In the given do_POST() function, we simply return the parsed data to the browser. The full process involves:

- **Parse out** the data
- **Find out** what the user wants to do (e.g. list the inventory, or compute the inventory value)
- **Perform** these computations

- **Format** the result, and
- **Send** them back to the browser.

We have largely given you the ways to parse out the data and determine what the user wants in do_POST(). You have already written the code to perform the actual tasks in your Phase One code. What you need to do now is to format the data and send back to the browser.

Suggested Approach

This part of the project is reasonably open. I encourage you to think through incremental steps that both will make progress and be easily testable. For example:

- Understand the home and form html files. Modify them so that the user sees a homepage like in Figure 2.
- Figure out how to display simple information in HTML. E.g., by working directly with the home or form file, figure out the syntax for displaying something like "The total inventory is $243929.43." If you are not familiar with HTML code, you can consult any of the information on the web, for example, https://www.w3schools.com/ but in this part of the project, the one you will use most is probably HTML list, which can be found here in this website: https://www.w3schools.com/html/html_lists.asp
- Right now, your Phase One code prints text output to the screen. It will be needed to be revised to return HTML strings. Make a copy of your inventory code (e.g. `inventory_web.py`) and **start** revising the code such that the contents will be formatted as an HTML string and returned to the caller. For example, in your Phase One code, you may have a function called '`print_inventory()`' that prints the inventory to the screen. You now need to revise this function such that it will return the string formatted as HTML text. **First, try revising just the** `compute_inventory()` function in `inventory_web.py`. This function doesn't require any parameters, so it's the easiest to start with.
- After revising the function so that it returns a string formatted as HTML, try to get it working in the web interface.
    - At the start of `simple_web_server.py`, you will need to `import inventory_web.`
    - In the `run` function of `simple_web_server.py` you will want to create your inventory. You can do so by adding the lines:

      ```
      global invent
      inventory = Inventory()
      ```

      at the very start of the `run` function.

- Now work within the `do_POST()` function. Can you create an if statement that recognizes when the user asks for you to compute the inventory? Can you make the website do *something* (really anything noticeable) when this happens?

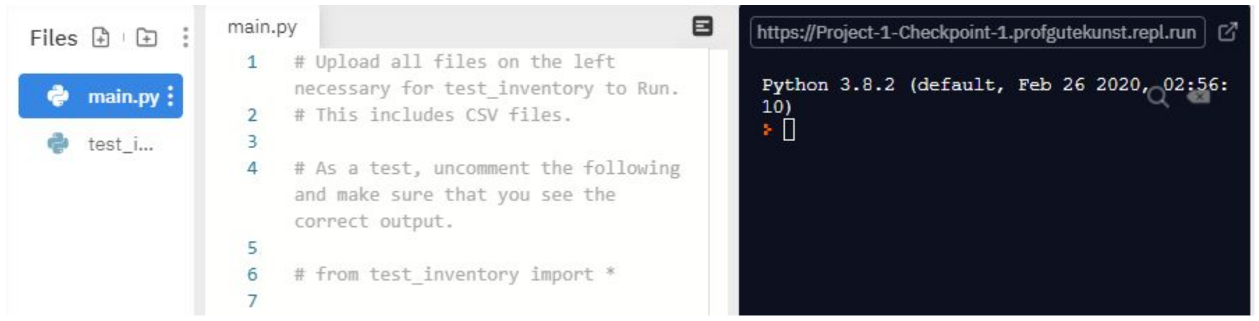
- Once `do_POST()` recognizes that you're asking it to do an inventory calculation, can you make it do the right thing? You'll use something like `inventory.comute_inventory(),` which should now be returning an HTML string.
- If you can do the above, you can do pretty much everything you'll need: from finding out what the user wants, to formatting and sending back the results. Now you should make this work for another two features.
- Think about programming style! It might help to have a `process_data function` that is called in `do_POST(),` so that `do_POST()` isn't overwhelmingly long.
- Think about extra features! Do you want a killer website? To let the user do more things? To check and respond to inputs the user makes? Something else entirely?!?

8 Submission for the Two Phases

Make sure you test all your programs before submission. Note that **we will not grade code that does not compile**. It is to your advantage to work iteratively, building and testing features as you go, so that whatever you submit compiles successfully.

You are required to submit two phases separately by the respective deadlines. All submissions will be to repl.it. In the classroom feature, you should see projects Project 1 Phase 1 and Project 1 Final Submission. These will be used for the first and final phase, respectively. The Final Submission will not go live until after Phase 1 is over.

To submit for Phase 1, in the appropriate repl.it project, you should see something like the following:

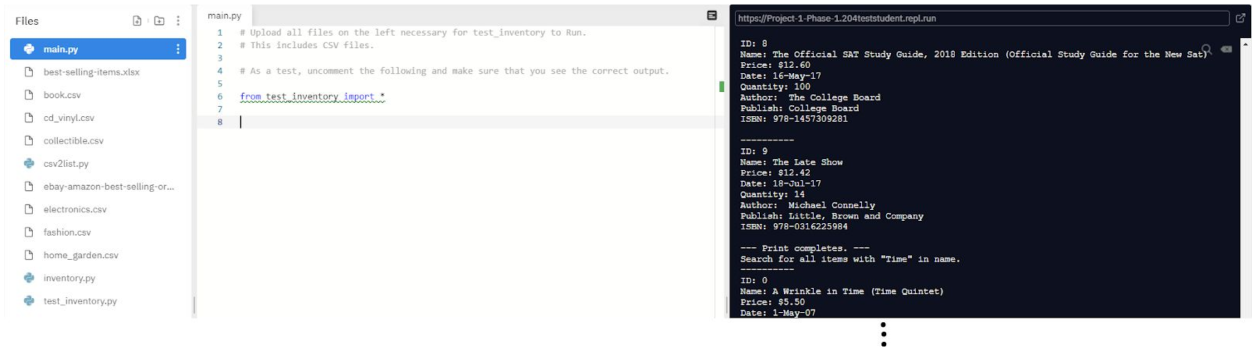


```
# Upload all files on the left necessary for test_inventory to Run.
# This includes CSV files.

# As a test, uncomment the following and make sure that you see the correct output.

# from test_inventory import *
```

```
Python 3.8.2 (default, Feb 26 2020, 02:56:10)
> 
```

Upload all files, including the CSV files, required to make test inventory run. Note: you should only upload to repl.it once your work is complete, and you will not be able to make anything run until all files are uploaded. You should be working in Spyder or your favorite IDE, and only upload + test once complete.

To verify that your work code correctly compiles after submission, uncomment the test in main.py. You should see something like the following:



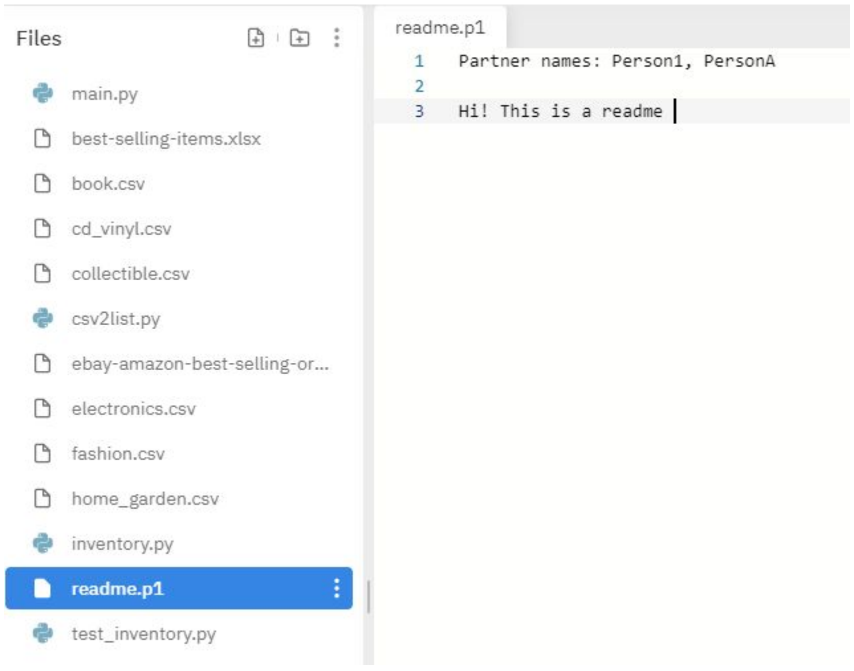
```python
# Upload all files on the left necessary for test_inventory to Run.
# This includes CSV files.

# As a test, uncomment the following and make sure that you see the correct output.

from test_inventory import *

```

```
ID: 8
Name: The Official SAT Study Guide, 2018 Edition (Official Study Guide for the New Sat)
Price: $12.60
Date: 16-May-17
Quantity: 100
Author:  The College Board
Publish: College Board
ISBN: 978-1457309281

----------
ID: 9
Name: The Late Show
Price: $12.42
Date: 18-Jul-17
Quantity: 14
Author:  Michael Connelly
Publish: Little, Brown and Company
ISBN: 978-0316225984

--- Print completes. ---
Search for all items with "Time" in name.
----------
ID: 0
Name: A Wrinkle in Time (Time Quintet)
Price: $5.50
Date: 1-May-07
```

If you do not see such an output, your code does not compile. In Phase 2 submission, you will need to do something similar.

Submission of a README file

Submit a plain text file named README with each of the two phases. Name the file README.p1 for phase one and README.p2 for phase two. The README file should contain a couple of paragraphs describing one or two points in your program that you like most, and one or two points that you had the most challenges. The README files should not be longer than a page. The README for part 1, e.g., should be available like the following. **Make sure that you + your partners names are clearly marked in a comment in the readme, as well as in a comment in the main.py file.**



```
Partner names: Person1, PersonA

Hi! This is a readme
```

9 Grading Rubrics

Normal requirements for programming work apply, including correctness, style, and organization. Specifically the following grading rubrics will be used for the two phases.

Phase One rubrics [55 pts]

[10 pts] Proper design of the Inventory class and Item class along with its subclasses.
[10 pts] Reading CSV file and creating the Python list(s) according.
[23 pts] Correct implementation of the designed classes, including constructors, string methods, and all methods needed to make the programs work.
[10 pts] Using good programming style, including naming, spacing, and others.
[3 pts] Overall program quality, including any features that stand out.

Phase Two rubrics [40 pts]

[18 pts] Implementing the server properly, including handling of forms.
[8 pts] Converting results from plain text to HTML for the server to use.
[8 pts] Using good programming style, including naming, spacing, and others.
[6 pts] Overall program quality, including any features that stand out.

Readmes [5 pts]
[5 pts] Readmes are included, thoughtful, and follow length+naming guidelines.