

CSCI 204 – Introduction to Computer Science II

Project 1 – E-Commerce Inventory

Assigned: Monday, 09/03/2018

First phase due: 09/14/2018

Second phase due: 09/21/2018

1 Academic Integrity

The project falls under the Homework and Projects in the course collaboration policy. Make sure you follow these policies.

2 Objective

The main objective of this project is to master the design and implementation of objects and classes with inheritance.

3 Problem Statement

You are just hired by *Samszone*, 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 the 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 IDLE. 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 program to demonstrate how a set of Python programs can interact over the web.

4.1 Phase One: Design and Implement the Inventory Class

You are given a set of test programs and data files. You can copy the files from this link, or from the following Linux directory using the command, assuming you are in your working project directory

```
cp -r ~csci204/2018-fall/student/project/p1/ .
```

If you are not in the proper working directory yet, please make and change into the directory by the following commands before issuing the copy command above.

```
cd ~/csci204
mkdir p1
cd p1
```

You should see a set of files copied to your directory. Let's concentrate on the CSV files first.

The following set of CSV files are given, followed by a brief description. Note that we have a total of four general classes of items in the database, Books, CD/Vinyl, Collectibles, and eBay items. The eBay items are 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, data, owner, and quantity.
electronics.csv	A collection of nine items with name, unit price, data, 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 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. Examine the program `test_inventory.py` first. From the `test_inventory.py` program you can tell that your Inventory class need to have the following public interface that is the methods that a public program can use.

Method name	Description
<code>init ()</code>	Constructor
<code>check_type()</code>	Return the type of object, using <code>isinstance()</code>
<code>compute_inventory()</code>	Compute the total value of the inventory
<code>print_inventory()</code>	Print the information of the inventory
<code>print_category()</code>	Print information by categories of the items
<code>search_item()</code>	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.

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.

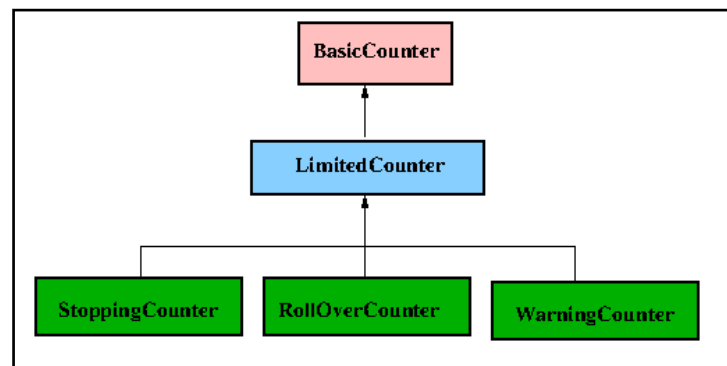


Figure 1: The Counter class hierarchy used in the lab

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 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 and grading easier.

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

- 1 All inventory items should be maintained as one Python list within the Inventory class.
- 2 While you can read any CSV files and parse them into Python lists using Python file structure we learned before, it may be easier to 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 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.

- 4 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.

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

4.2 Phase Two: Make Your Inventory Accessible through the Web

Now that your program works fine at the command line, or through the IDLE, 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 all the commands that you implemented in Phase One of the project. For example, one can query the total value of the inventory, search the names of the items by name, or print a category of items. Figures 2, 3 and 4 show some sample screen shots that illustrate the idea.

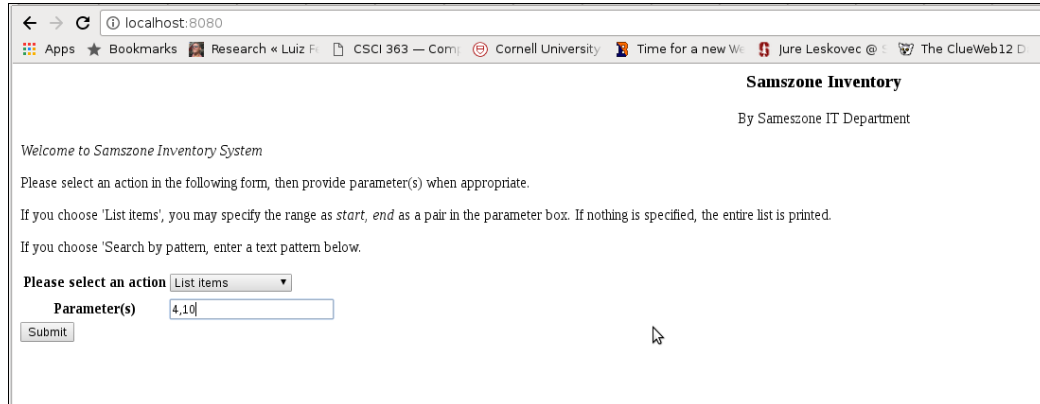


Figure 2: Home page of Samszone website with the list option shown

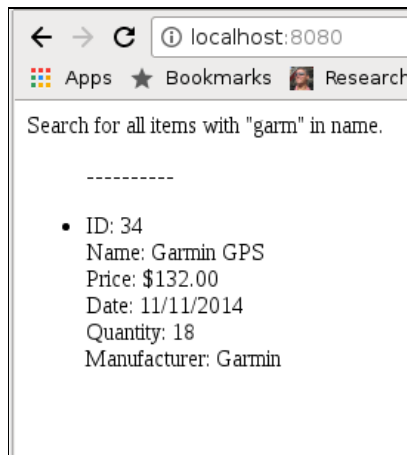
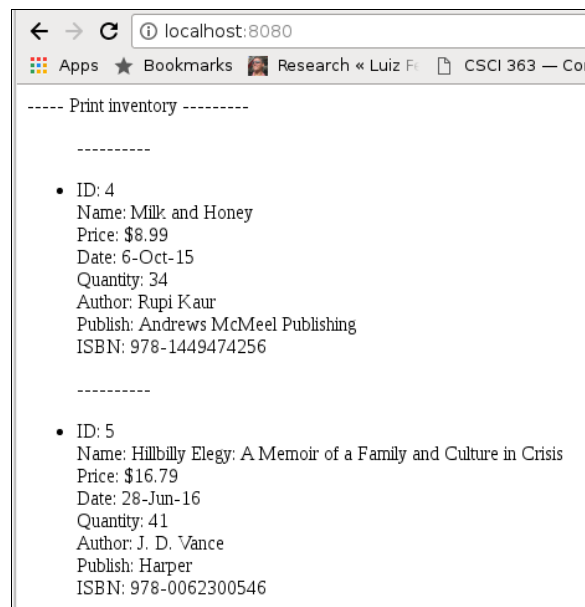


Figure 3: Search result for key phrase “garm” in names

Figure 4: Partial result of *listing* operation

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 IDLE) 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** page back to the web browser for display. While the details of web programming is not necessary the focus of this project, we provide you with a sample Python server program. From the examples, you can easily figure out what takes place in web programming and you can implement your own logic for your task of creating a website for *Samszone*.

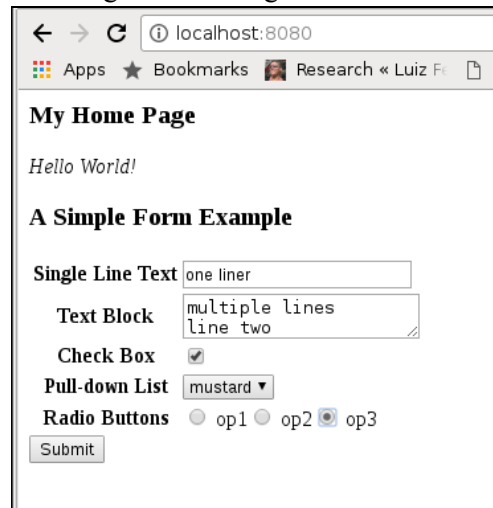
Try out the given server program

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 at the Linux command line, or in your IDLE, run the program `simple_web_server.py`. You should see something similar to the following on your terminal window or your IDLE 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.



← → ↻ ⓘ localhost:8080

Apps ★ Bookmarks Research « Luiz F

My Home Page

Hello World!

A Simple Form Example

Single Line Text

Text Block

Check Box ☒

Pull-down List

Radio Buttons ☐ op1 ☐ op2 ☒ op3

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. The figure shows the screen shot that the user (you) has typed the some inputs, but not yet click 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. 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 simply echo what the user typed in.

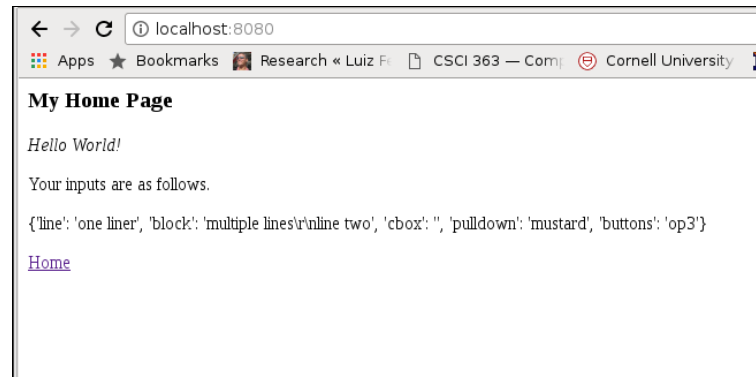


Figure 8: The response page generated by the server

One can see from the figure that the information typed in by the user such as “one liner” 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. 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>, see 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 preceded 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

Your work will modify the functions `do_GET()` and `do_POST()` 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. What you need to do is to **parse out** the data, **find out** what the user wants to do, for example, list the inventory, or compute the inventory value, **perform** these computation, **format** the result, and **send** them back to the browser. We have given you the ways to parse out the data and determine what the user wants. You have 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.

An easier way of accomplishing these tasks is to make a copy of your Phase One code. Revise the code such that instead of printing the results to the screen as Phase One required, you now revise 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. Your `do_POST()` function then can return this formatted text directly to the browser.

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

5 Submission for the Two Phases

Make sure you test all your programs before submission. **You are required to submit two phases separately by the respective deadlines.** All submissions are to the course Moodle site. You must submit all the relevant files in a zipped file such that your programs will run in their own folder. Create the zipped file using the following command at your Linux command line, assuming you have followed instructions in the project description in creating the project folder named `p1` under your `csci204` directory.

```
cd ~/csci204/  
zip -r p1.zip p1/
```

The above commands creates a file named `p1.zip` in your `csci204` directory. You can submit the zip file to the course Moodle site. In Phase One submission, you must submit all files that will make

the test program `test_inventory.py` run directly from its own folder. In Phase Two, you must submit all relevant files that will make the program `samszone_web_server.py` correctly.

Submission of a README file

Submit a plain text file named README for 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.

6 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 [60 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.
- [25 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.
- [5 pts] Overall program quality, including any features that stand out.

Phase Two rubrics [40 pts]

- [20 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.
- [4 pts] Overall program quality, including any features that stand out.