Assignment 1

Version: 0.9

Due: start of your lab period, week of Feb 15

Worth: 15% of final grade

Late assignments will be assessed a 10% penalty per school day late. Because I must test your program, a "school" day is defined as a day when I am at Seneca and can test your programs (i.e. the days of reading week are not considered school days since I will not be here to test your program). Note that you must be in your proper section for me to test your program.

Incorrect/unsatisfactory assignments submitted for testing will be penalized as described above. Make sure you've thoroughly tested your program before you ask me to test it.

This is an individual assignment. Copying/plagiarism will result in a grade of 0 and be reported to the college. Please see the (many) college resources on copying/plagiarism.

Introduction

In this assignment you will create selected parts of a web-based inventory management system, such as might be used as part of an online store. To prepare for the assignment choose a fictional store name and product range (you cannot use the company name or product type in my examples below). You cannot use the name of a real company.Your web content must meet Seneca's Acceptable Use Policy, so do not include a company theme or products that might be offensive. If you are having trouble thinking of a theme, think of a hobby you have, and make the site about that (e.g. sports equipment, video games, musical instruments, etc.) You can use the names of real products, but any description provided must be written by you (i.e. do not copy and paste a description of a product from a real site selling that product).

The code in this assignment will be built upon in future assignments. Therefore it is important to write your code in a modular fashion, so that you can easily update or modify selected parts.

Note: it is critical you follow the programming requirements below **exactly**. There is a very good reason for this: in the real world it is highly likely you will be working with other programmers, and if you change the smallest thing in the specs you can mess up the other programmers, costing your company thousands (or tens of thousands!) of dollars - or worse (and equally expensive), a website that doesn't work quite right. In addition, it is also highly unlikely you will be the only person ever working on a particular piece of code, and the programmer who is asked to modify your code may refer to the program requirements to determine what your code is doing. If you have deviated from the requirements, then this may create problems that are particularly difficult to track down, again costing your company lots of money.

Program Requirements

Preparation

1. On your removable drive create an **assign1** directory under your **html** directory. Keep all the code for this assignment in this directory (this will make it easier to keep it separate from future assignments).
2. Create a **secret** directory under your home directory. Give the directory permission 701 so that Apache, which runs under a userid that is part of others, can enter this directory.
3. In your secret directory, create a text file named **topsecret**, which contains (**in this order**) MySQL server name, your MySQL username, your MySQL password, and the DB name. We will use this file so that we don't have to hard-code our MySQL infor into our program. Give this file permission 604 so Apache can read the contents of the file.
4. In your DB create a table with the following structure:

```
create table inventory (
id int zerofill not null auto_increment,
itemName varchar(40) not null,
description varchar(2000) not null,
supplierCode varchar(40) not null,
cost decimal(10,2) not null,
price decimal(10,2) not null,
onHand int not null,
reorderPoint int not null,
backOrder enum('y','n') not null,
deleted enum('y','n') not null,
primary key (id)
);
```

Note that **id** is the primary key, and will autoincrement if you store a null value value for that column when adding a new record.

Requirements

You must write the four programs described below that work with the above DB.

Add Program (named add.php)

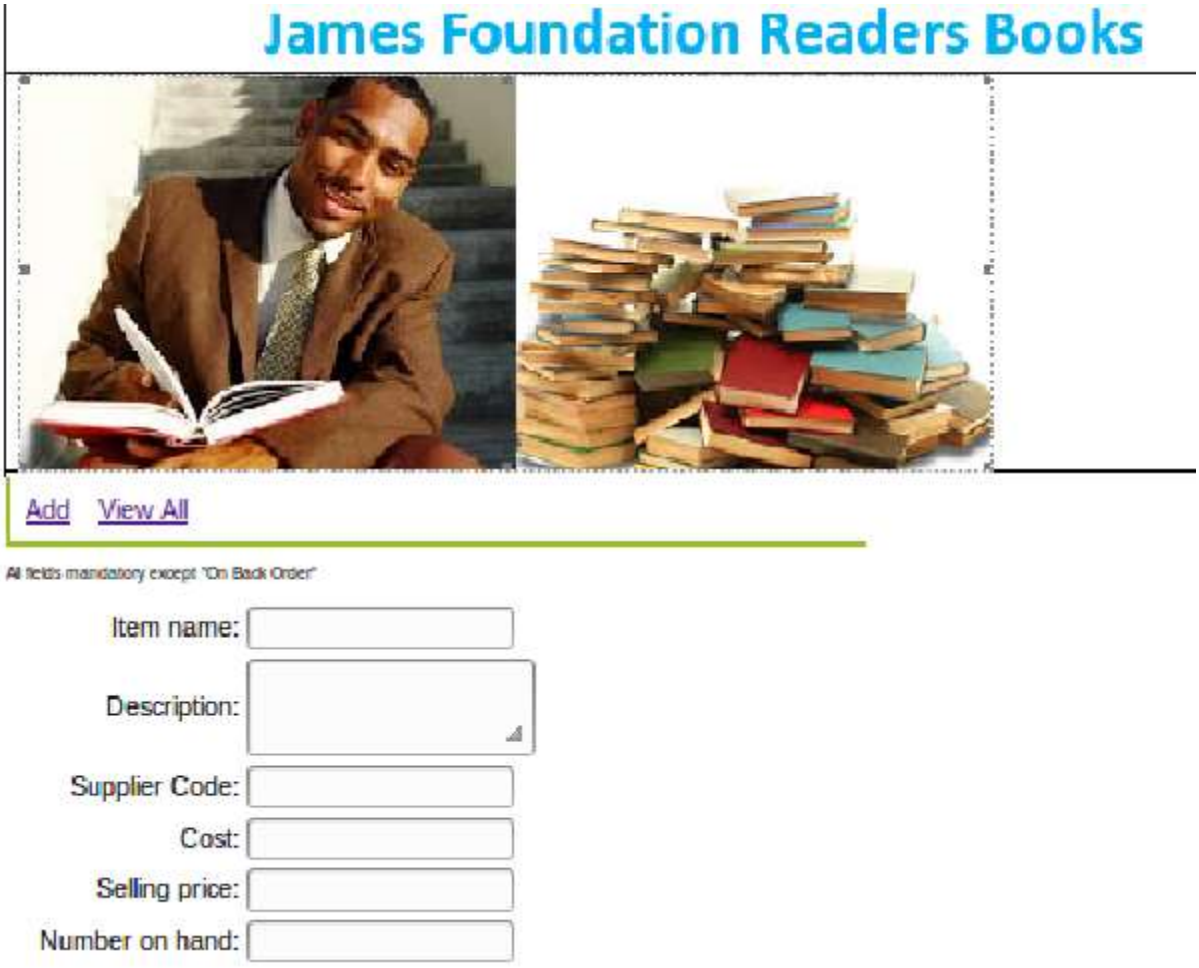
- Displays an empty form when first invoked (not a POST)
- At the top of the page there should be a menu that has the following items: "Add" and "View All"
- When the submit button is clicked, the program validates the input data, using regular expressions, as follows:

name letters, spaces, colon, semi-colon, dash, comma, apostrophe and numeric character (0-9) only - cannot be blank
description letters, digits, periods, commas, apostrophes, dashes and spaces only

(may contain newlines since this is a multiline textarea) - cannot be blank
supplierCode letters, spaces, numeric characters (0-9) and dashes only - cannot be blank
cost monetary amounts only i.e. one or more digits, then a period, then two digits - cannot be blank
price monetary amounts only i.e. one or more digits, then a period, then two digits - cannot be blank
onHand digits only- cannot be blank
reorderPoint digits only- cannot be blank
backOrder - no validation required

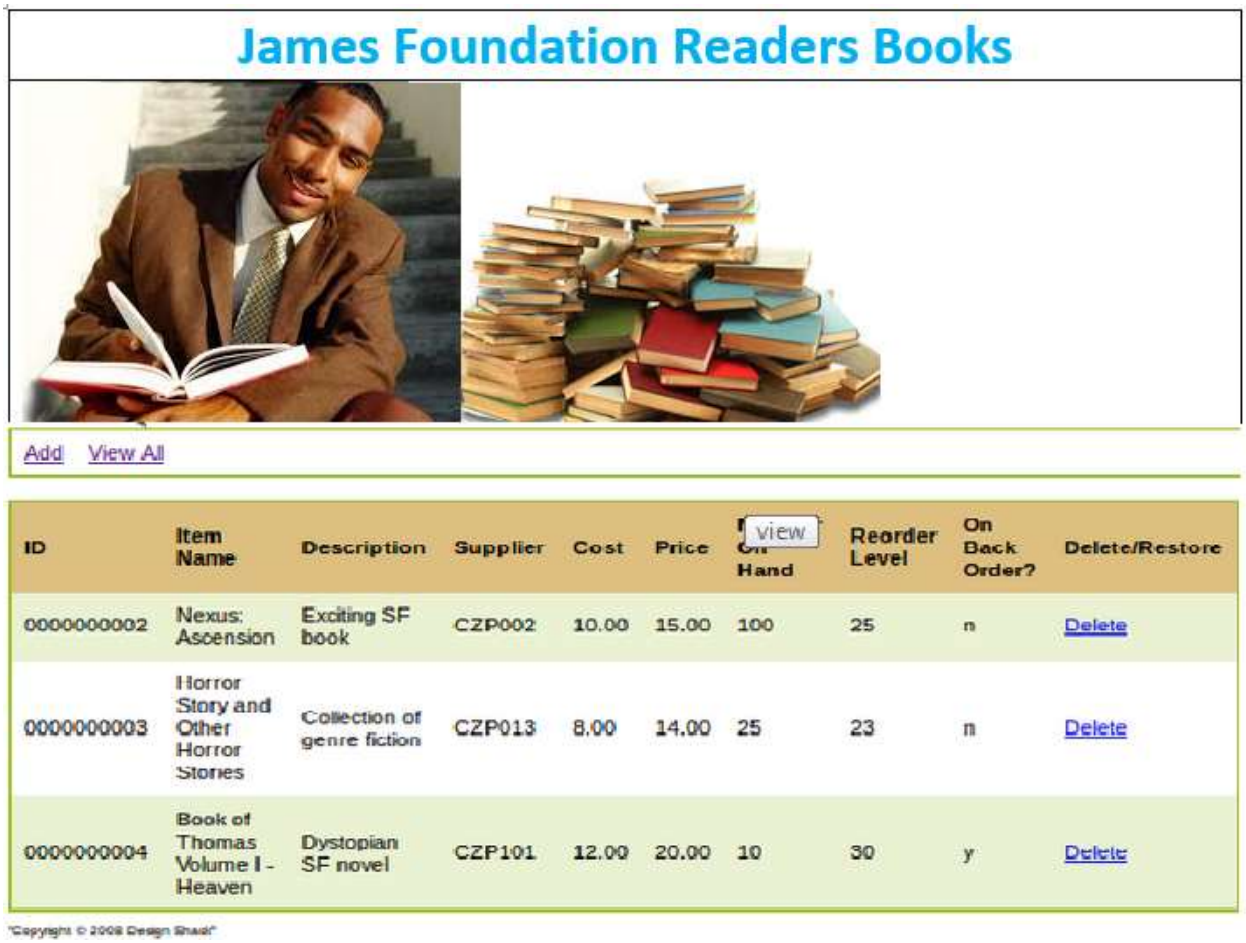
Note 1: all validation must be done on the server side using regular expressions.
Note 2: **all** fields may contain leading and trailing blanks.

- Where it says you are allowed spaces, you are allowed as many spaces as you like, anywhere in the string. Note, however, that a string of all spaces is still a blank field and so means that the user hasn't entered anything (i.e. hasn't entered anything in a mandatory field). This should be flagged as an error. Note also that it is common practice to remove leading and trailing blanks on input strings before storing them since they serve no purpose and may take up space that should really be reserved for other things (like letters!). For example, if you allow 20 characters for a name, and someone types in 20 blanks at the start of the string, this would store a blank string in the DB. In addition, if you store leading blanks in a DB field, then sorted output would not be correct (entries starting with blanks would come before 'A'). So use the `trim( )` function to remove leading and trailing blanks before you store any string data.
- If the data is valid, the program stores the data in your DB and invokes the **view.php** program. (The new item should appear in the list.) Note: you may **not** use buffer control, the HTML meta tag, or javascript to avoid the headers problem when you try to redirect because these are ways to hide bad logic and coding, and there are performance costs. Instead, you should use proper PHP coding techniques.
- If the data is **not** valid, the form is redisplayed with all the fields repopulated with the POSTed values (including checkboxes, radio buttons, dropdowns, etc). Error messages should be displayed **next to the fields in error in red**. Error messages should be as informative as possible without being overly long.
- If the user clicks on the "Add" link, the **add.php** program should be called, and the page should redisplay with all fields blanked/reset. No validation should be done, and nothing should be stored in the DB.
- If the user clicks on the "View" link, the **view.php** program should be invoked, no validation should be done, and nothing should be stored in the DB.



View program (named view.php)

- Displays all records in your DB, one per line, in a tabular format.
- At the top of the page there should be a menu that has the following items: "Add" and "View All"
- At the end of each row there should be a link. If the "deleted" field in the record is 'n', this link should be "Delete". If the "deleted" field in the record is 'y', this link should be "Restore". See the delete.php program for a description of how this field changes. Note also that "deleted" records are displayed in view all.
- When the "Delete" link is clicked, the **delete.php** program should be called.
- When the "Restore" link is clicked the **delete.php** program should be called.
- If the user clicks on the "Add" link, the **add.php** program should be called.
- If the user clicks on the "View" link, the **view.php** program should be invoked.



Delete Program (named delete.php)

- If the item is not already deleted, it should be "deleted" (i.e. the deleted flag should be set to 'y' in the DB, but the item should NOT be physically deleted) and **view.php** should be called. This program does not display a web page - that is done through the call to **view.php**.
- If the item is already deleted, it should be "undeleted" (i.e. the deleted flag should be set to 'n') and **view.php** should be called. This program does not display a web page - that is done through the call to **view.php**.
- Note that this program never does a "hard" delete of a record, it just sets a flag.

Function Library (named a1.lib)

- Contains common code for the above programs. For example, css, headers, footers (copyright notice), and the menu are common to both the add and view pages, and so functions to generate these things should be contained here. There may be more common code as well, like DB access code. There should be NO active code in this file, and you should not drop into HTML mode in this file.

Additional requirements

- **PHP should be embedded within HTML** - not the other way around. This means you should never print/echo or otherwise output static HTML within PHP mode. Therefore, print and echo statements should be used only to output variables.
- **Password protect the directory in which you place your files** - you are developing pages for privileged users and you should not allow other people to access them!
- Each HTML page should have an appropriate title.
- You should put your css in an external file and use the LINK tag to bring it into your web pages. (You do not have to submit your css file.)
- Use realistic data to test your application, not junk. Have at least 5 entries in the inventory table when demonstrating your assignment to me. DO NOT USE THE SAMPLE DATA PROVIDED HERE.Remove any random test data before submitting your assignment.
- Do not try to make your solution look like the images, just make sure the same content is provided and all pages have a consistent "look-and-feel" - i.e. there must be
    - A company name at the top of each html page (logo/image optional)
    - A menu/navigation bar with links to 'add' and 'view'.
    - A copyright notice at the bottom of each page.
- You are not being marked on your design specifically, so don't spend a lot of time on it at the expense of the PHP programming. You can use fonts, colour schemes, and other decorations as you wish, but all HTML output must use the same 'look and feel' – to accomplish this, use include as shown in class.
- You must use your own theme, i.e. company name and choice of products. Make sure that it is not offensive to others. You are not allowed to duplicate the products, company name, or theme shown here, you must create your own.
- You do not need to use graphics or images, but if you do they must be ones you have permission to use, and they must be attributed (i.e. say where you got them and what licence they are used under) in the alt tag of each one. You are not allowed to use the logo or name of an existing company at all.
- Acceptable image sources include:
    - images you take or create yourself, or modify from an acceptable source, such as below
    - images released under an appropriate cc (creative commons) or other applicable open-source license, e.g.
        - http://commons.wikimedia.org/wiki/Main_Page
        - http://www.flickr.com/creativecommons/
        - http://openclipart.org/
- You may use CSS, and Javascript too, but you are not allowed to use Javascript for validation.

Grading

I will test your assignments during the lab period the week before reading week. I will then mark your source code during reading week. The following checklist will be used to mark your assignment. To get a perfect mark, make sure that your assignment meets all of these requirements before you submit it. For each of the points below, marks will be deducted if you fail to meet the requirement. Enough small problems, or one or two major ones, means you will be required to fix your program and resubmit.

	View displays all records, properly formatted
	Delete/Restore work
	Add form displays properly with correct fields

	Add form shows specific error messages if any data invalid
	Add form shows entered data in form if any data invalid
	Adds data to db
	Menu works and is same on Add and View
	Source code – well-commented and efficient (no duplicate code, proper use of functions/includes, no print/ecoh of static HTML, etc)
	Validation works properly – uses correct regular expressions
	Original company and products (proper attribution of images, if used, etc.)
	Other deductions for not following requirements

Submission Guidelines

I will test your assignment during class the day it is due. If you are not prepared for testing on the day designated, then you will have to arrange for an out of class demo on another day.

I will also be grading your source code. So **after** I have successfully tested your program, you should immediately submit your four source code files (add.php, view.php, delete.php and library.php) through the link below on Moodle. Do NOT change your source code. Do NOT zip your files into one file. Submit them as four separate source files. You do not have to submit your css file.

All assignments must contain the student declaration below, and comments at the beginning of each source file with the following information:

Subject Code and Section (eg. IPC144A, OOP244B, etc.)
Student Name
Date Submitted

```
Student Declaration

I/we declare that the attached assignment is my/our own work in accordance
with Seneca Academic Policy. No part of this assignment has been copied
manually or electronically from any other source (including web sites) or
distributed to other students.

Name ---------------------------------

Student ID -------------------------
```

All assignments must contain a statement(s) of the nature of the problem being solved (i.e. Purpose of the assignment). Assignments should be **well commented**, and must be **properly indented** according to the guidelines established in class (i.e. a minimum of 3 spaces of indenting is required for each new code block). Indentation must be consistent (same number of spaces), and correct. See the code in the text or on the web site for examples.

You must have an appropriate program structure (i.e. broken into proper sized functions). Deductions will be made if you do not have proper comments, spacing and indentation in your source code. Here is a resource for typical coding conventions that I suggest you follow:http://www.phpdeveloper.org.uk/articles/php-coding-guidelines/. Here are additional requirements:

- **Make sure to include a comment at the start of your program** identifying yourself, the course, the assignment, etc. This should be followed immediately by the **Student Honesty Statement**.
- **Put a comment above each function** to identify what the parameters represent, what the return value (if any) represents, and a one line statement of what the function does.
- **Do not do more than one thing in a function** - that is, a function should do one thing, and do it well. For example, you might want to write a function to print out your web page. But you don't want to write a function that prints out your web page and processes it.
- **Put comments within functions**, whenever you are doing something that would not be self-evident to someone reading your program. Don't put a comment on every line - too many are as bad as not enough.
- **Read and understand the specifications**. If you do not understand the specifications, ask me for clarification. If you do not implement something required, you will lose marks, even if you didn't understand the requirement - i.e. it is your job to seek clarification.
- **Do not change the specifications**. For example, if you print something out and the assignment does not tell you to print it out, you are changing the specification and will lose marks.
- **Get rid of unnecessary (and confusing) duplication**. For code, you can do this by factoring out common code and putting it into a function. In a regex, you can always delete {1} because it simply means the character in front of it repeats exactly once, but they always do by default! I also noticed several people were including parentheses in regexes, but they served no purpose. For example, '([0-9]{1})' is the same as '[0-9]'. Why make it look more complicated than need be?
- **Indent the same number of spaces. Always.**
- Always tab, or manually space, but **don't mix the two** because tab stops change in different editors and mean your code won't line up nicely.
- Develop your code anywhere you like, but **make sure your code looks nice in gedit**. That's what I use to check it.
- If you develop your code in Windows, don't submit it without testing as above. Transferring source files from Windows to Linux requires transferring them in ASCII mode - and dragging and dropping often transfers files in binary mode. If transferred in binary mode, you will get extra characters in your file and your program formatting will look terrible.
- **Do not put multiple returns in a function**. If you have them now, refactor your code to get rid of them. The exception to this would be calling the exit function where you want the program to terminate. You may have one return, and multiple exits.
- **Always check to see if external resources have been allocated properly** - i.e. check that a file opens properly, or that a DB connection is working. If it is not allocated properly, you should print a meaningful error message and terminate your program.
- **Do not leave external resources "open" when your program terminates** (i.e. close all files, DB, etc)
- **Do not leave external resources open for extended periods of time**. If, for example, you open a file, and leave it open for a long time, you can lose info written to it if there is a power failure because a lot of output is "buffered" when your program "writes", and is not actually written to the physical device until some time later. (You can also "flush" buffers to force physical writes.)
- **Do not open and close a resource every time you want to write something to it** if you are doing so in a tight loop. Open it before the loop, and close it after the loop.
- **Use only boolean flags** - i.e. flags that have only two states, true or false. Multi-state flags can create all sorts of subtle logic problems.
- **Do not condition while loops to be infinite loops** unless they are supposed to be infinite loops. The condition in the while should show us what we are testing - however, having "while (True)" doesn't really tell us anything about the loop. This also leads to breaking out of the loop in funny, non-conventional ways, and frequently is accompanied by numerous flags. If you are going to

stop a loop based on an input value, do a pre-read and then read again at the end of your loop, rather than putting a big If inside your loop. Or use a do-while.