

# Web Programming

**Assigned:** Nov. 22, 2018

**Due:** Nov. 30, 2018 @2355

**Version:** 0.1.1

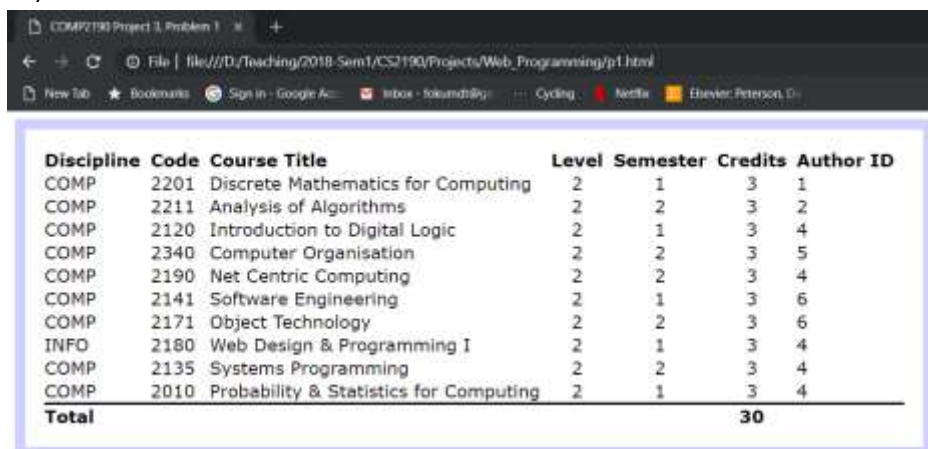
## Problem 1 (10 points)

Create a single HTML document that presents two different appearances, determined by the document's CSS stylesheet. Your HTML file should be called p1.html and the two stylesheets should be called p1a.css and p1b.css. If the HTML file links to p1a.css then it should appear like this ("Version A"), assuming you are running Chrome 70.0 on a Windows 10 machine:



Discipline	Code	Course Title	Level	Semester	Credits	Author ID
COMP	2201	Discrete Mathematics for Computing	2	1	3	1
COMP	2211	Analysis of Algorithms	2	2	3	2
COMP	2120	Introduction to Digital Logic	2	1	3	4
COMP	2340	Computer Organisation	2	2	3	5
COMP	2190	Net Centric Computing	2	2	3	4
COMP	2141	Software Engineering	2	1	3	6
COMP	2171	Object Technology	2	2	3	6
INFO	2180	Web Design & Programming I	2	1	3	4
COMP	2135	Systems Programming	2	2	3	4
COMP	2010	Probability & Statistics for Computing	2	1	3	4

If the HTML file links to p1b.css then it should appear like this in Chrome 70.0 on Windows 10 ("Version B"):



Discipline	Code	Course Title	Level	Semester	Credits	Author ID
COMP	2201	Discrete Mathematics for Computing	2	1	3	1
COMP	2211	Analysis of Algorithms	2	2	3	2
COMP	2120	Introduction to Digital Logic	2	1	3	4
COMP	2340	Computer Organisation	2	2	3	5
COMP	2190	Net Centric Computing	2	2	3	4
COMP	2141	Software Engineering	2	1	3	6
COMP	2171	Object Technology	2	2	3	6
INFO	2180	Web Design & Programming I	2	1	3	4
COMP	2135	Systems Programming	2	2	3	4
COMP	2010	Probability & Statistics for Computing	2	1	3	4
<b>Total</b>					<b>30</b>	

Here are some additional details and requirements for this problem:

- The content should be described in a single HTML file, using a <table> element to display the main table.
- There may not be any formatting information in the HTML file other than class and id attributes.

- The appearance must be generated entirely with CSS style information; you may not use images or JavaScript for this problem.
- The only change that should be required to switch from Version A to Version B is to change the `<link>` element in the header to refer to a different CSS file.
- Your CSS files must appear in a directory called `styles/`.
- Try to duplicate the appearance in the images above exactly ("pixel perfect"). For example:
  - Some of the columns should be centered whereas others are left-justified.
  - The "Total" line appears only in Version B (hint: you may find the `display` attribute useful in producing this effect).
  - The title in the browser title bar should read "CS 2190 Project 3, Problem 1".
  - Both versions use the Verdana font with a size that is 1.1 times the width of an 'm'.
  - The background color for the header row in Version A is #687291.
  - The background colors for the body rows in Version A are #eeff2 and #dfe1e7.
  - The white lines between rows in Version A are 1 pixel wide.
  - The color for the frame around Version B is #d0d0ff.
  - The frame in Version B is 10 pixels wide; there are 20 pixels of empty space on either side of the frame.
  - The horizontal rule above the "Total" line in Version B is 2 pixels wide.
  - Match the paddings and spacings as closely as possible.
- The tables should be fixed-size (i.e. they should not expand and contract as the window size changes).
- Your HTML file must be a valid XHTML document that passes validation at <http://validator.w3.org>, and your CSS files must pass validation at <http://jigsaw.w3.org/css-validator/>
- Note: the border and margin styles are not supported for `<tr>` elements; `<td>` elements support border but not margin.

## Problem 2 (15 points)

Write the XHTML code to create a form with the following capabilities:

- Text widgets to collect the course code, course title, course discipline, level, credits, and semester offered.
- A select one dropdown menu. The options and their values should be:
  - Sara Lambert, value = 1
  - Brad Williams, value = 2
  - Rosalyn Brown, value = 3
  - Emil Powell, value = 4
  - Rene Picard, value = 5
  - Larry Fields, value = 6
- You should also include a hidden field with the value:
 

```
"c7fafc6d1d3ec4671a342b64e319909f"
```

- When the Submit button the form is clicked, a JavaScript validation function must be run. This function should ensure that
  - None of the text fields listed above is empty.
  - That an option was selected from the dropdown menu
  - The course code is a four-digit integer
  - The course discipline is a four-letter string
  - Credits is a number greater than zero and less than or equal to 8
  - The semester offered field is either 1, 2, or 3.
- The title in the browser title bar should read “Course creation form”
- You **must** use a stylesheet for this assignment. Your stylesheet must be called p2.css and it should be saved to the `styles/` directory.
- Your JavaScript must be called `p2.js` and it should be saved to `scripts/` directory.
- The form should use the Verdana font with a size that is 1.1 times the width of an ‘m’.
- If an error is detected in a field during validation, that field should be colored in red. Define a class in your stylesheet to highlight these errors. You should also have another class for fields containing valid data.
- You **must not** rely on HTML5’s field validation commands for this part of the problem.
- Your HTML file must be a valid XHTML document that passes validation at <http://validator.w3.org>, and your CSS files must pass validation at <http://jigsaw.w3.org/css-validator/>

### Problem 3 (35 points)

Write a PHP script that collects the data from the form of Problem 2, above, and stores the information into a MySQL database. Your code should also examine the discipline, course code, and title submitted to verify that they are valid. Valid disciplines are four characters long. A valid course title is any non-empty string, while a valid course code is a string of length 4 consisting only of integers. After each new and valid course is submitted, print the contents of the database to a table that is formatted as in Problem 1-a. The database for this problem should be named CourseMgmtDB. Use the database username “comp2190SA” with the password “2018Sem1”. The CourseMgmtDB should contain a table called Courses. The fields of the table are as follows:

Field	Data type
Discipline	String
Code	String
Title	String
Level	Integer
Credits	Integer
AuthorID	Integer
Semester	Integer
ID	Integer, autonumber

## Problem 4 (Optional: 15 extra-credit points)

If you complete Problems 1—3, you can create a login form for your application.

### Aside: Handling Database Passwords

In general, it is not secure to store passwords directly in a database. Someone who is able to read the database, e.g., a rogue system administrator can easily retrieve all of the passwords for all users. A better approach is to apply a message digest function, e.g., MD5, to each password, and store only the message digest in the database. MD5 takes a string such as a password as input and produces a 32-character string of hex digits (called a message digest) as output. As we have already seen, the output provides a unique ‘fingerprint’ for the input string (there is no known way to produce two different strings with the same digest); second, given a message digest, there is no known way to produce a string that will generate that digest. You will take the password string submitted by each user, invoke PHP’s MD5 function to compute the digest and store only the digest to the database. Once this is done, you can discard the password. With this approach you can make sure that a user enters the correct password when logging in, but if someone reads the digests from the database they cannot use that information to log in.

The approach of the previous paragraph has one remaining flaw. Suppose an attacker gets a copy of the database containing the digests. Since the MD5 function is public, an attacker can employ a fast dictionary attack to guess common passwords. To do this, the attacker takes each word from a dictionary and computes its digest using MD5. Then the attacker checks each digest in the database against the digests in the dictionary (this can be done very quickly by putting all the dictionary digests in a hash table). If any user has chosen a simple dictionary word as their password, the attacker can guess it quickly.

In order to make dictionary attacks more difficult, one must use password salting. When a user sets his/her password, compute a random number and concatenate it with the password before computing the MD5 digest (the `mt_rand()` function will generate a random number). The random number is called a *salt*. Store both the salt and the digest in the database. To check a password during login, retrieve the salt from the database, concatenate it to the password typed by the user, and compute the digest of this string for comparison with the digest in the database. With this approach, an attacker who has gained access to the login database cannot use the simple dictionary attack described above; the digest of a dictionary word would need to include the salt for a particular account, which means that the attacker would need to recompute all of the dictionary digests for every distinct account in the database. This makes dictionary attacks more expensive.

### Main task

Your form for Problem 4 should have a text field for the username, a password entry field, a hidden field, and a submit button. In the PHP to generate this login form, set the value of the hidden field to the MD5 hash of the session ID. Refer to the Users table to get usernames. On the client-side, each user’s password is the string “123456”. To log the user in use the `salt` as described above and refer to the Users table. The Users table is specified below. If the login attempt is unsuccessful, increment the

contents of the `failed_attempts` field. If the user is logged in correctly, update the `last_login` field to the current time. Once the user is logged in, run the same script that you wrote for Problem 3 to list all the courses in the database.

Table 1: Users table

Field	Data type
first_name	String
last_name	String
username	String
email	String
isAdmin	TinyInt(1)
failed_attempts	Integer
last_login	DateTime
password_digest	String
salt	String
ID	Integer, autonumber

## Rules

- **The project is designed to be solved independently.**
- **You may not share submitted code with anyone.** You may discuss the assignment requirements or your solutions away from a computer and without sharing code, but you should not discuss the detailed nature of your solution. If you develop any tests for this assignment, you may share your test code with anyone in the class. Please **do not put any code from this project** in a public repository.
- You may not use external JavaScript libraries such as JQuery.

## What to turn in

1. You will hand in all your XHTML, CSS, Javascript, and PHP files. All your CSS files must appear in a directory called 'styles', your JavaScript and PHP files in a directory called 'scripts'.
2. A separate (typed) document of a page or so, describing the overall program design for Problem 3, a verbal description of "how it works," and design tradeoffs considered and made. Also describe possible improvements and extensions to your program (and sketch how they might be made).
3. A separate description of the tests you ran on your program to convince yourself that it is indeed correct. Also describe any cases for which your program is known not to work correctly.

## Grading

- XHTML and CSS files for Problem 1.
  - Page displays correctly as specified in assignment sheet – 7 points
  - Pass validation – 3 points

- XHTML, CSS and Javascript files for Problem 2.
  - Form displays correctly as specified in assignment sheet – 5 points
  - Validation function works as specified on assignment sheet – 7 points
  - Passes validation – 3 points
- Program listing for Problem 3
  - Works correctly as specified in assignment sheet – 20 points
  - Contains relevant in-line documentation – 3 points
  - Elegant code, i.e., proper decomposition of functionality – 2 points
- Design document
  - Description – 5 points
  - Thoroughness of test cases – 5 points

## Acknowledgements

The text on salting and hashing is based on an assignment developed by John Ousterhout for his CS142 course.