CSCI N341 : Client-Side Web Programming

Lab Guidelines

Course Goals

The goals of CSCI N341 are:

- To develop skills in algorithmic problem-solving
- To understand the tools of a client-side programmer
- How to create and use variables and objects
- How to create and use conditional statements to form complex decision-making trees
- How to create and use conditional statements to drive looping structures
- How to create and utilize functions
- How to effectively employ JavaScript to create interactive Web pages
- How to effectively manipulate the Document Object Model
- How to increase the efficiency of HTML with Ajax technology
- How to use jQuery
- How to write and use plug-ins
- To develop skills in software engineering

Requirements for ALL Labs

1. Produce all web content to **HTML5 standards**. Validate all files at **W3C Markup Validation Service**.
2. Put your JavaScript in a separate file from your HTML.
3. Use a (document).ready jQuery wrapper for each JavaScript file.
4. Update the **header block comments** for each file.
5. **Comment your code** to make it easier to understand.
6. Utilize the Google-hosted jQuery library. See testFile.html in Web Page Redux for an example.
7. Observe the order of external files (css first, then Google-hosted, then locally hosted files).
8. Check your browser's JS console / developer tools for error-free code.
9. Variables need **descriptive names, lower camelCasing, and comments** describing the purpose of each variable.
10. Use **appropriate prefixes** to describe the data type of your variables.
11. Each **function should have its own comment block**.
12. Post assignment as a link from your class index page labelled with the provided assignment name.
13. Read ALL instructions for every assignment at least once.

Submission Deadlines, Naming Conventions

1. All labs are due as stated in Canvas.
2. You MUST name your lab files and your lab journal files *exactly* as you are directed to name them in lab directions. This requirement is NOT to stifle creativity or individuality. Rather, we don't want to waste time LOOKING for your work

Hosting Server Information

1. In N341, you'll post all of your lab files and journal files to the IUPUI Computer Science Department's Server.
2. Put all of your files into a `/~[username]/public_html/n341/` directory.
3. Be sure that all server content has the proper Unix permissions. Typically, directories will have a "755" mask, and files will have a "644" permission mask. Most SFTP products now support a GUI for setting file properties (WinSCP and Cyberduck, for example, support such a GUI.) If your SFTP program does not have a GUI for changing permissions, you can issue Unix/Linux commands from the server command line:
```
chmod 644 [filename]
```
4. Be sure that ALL files are linked to your n341 Course Page.

Documentation Comments

At the beginning of each of your files (HTML, CSS & JavaScript), be sure that you've included a thorough Header Block Comment, formatted like the following:

```
/**************************************
TITLE: <type the lab title here>
AUTHOR: <type your name here> (<type your initials here>)
CREATE DATE: <type the date you began editing the file>
PURPOSE: <briefly describe the purpose of the program>
LAST MODIFIED ON: <type the date of the last modification>(1)
LAST MODIFIED BY: <type the name & initials of the programmer who last edited>(2)
MODIFICATION HISTORY:(3)
<list the dates of each modification, explain each and give the initials of the programmer who made each modification>
***************************************/
```

(1) - If you didn't have any modifications, this date should match the create date.

(2) - If you didn't have any modifications, you still need to put your name & initials here.

(3) - If you didn't have any modifications, put "No modifications" under *Modification History*.

Example of a well-formed header block comment:

```
/**************************************
TITLE: Hello World
AUTHOR: Paul C. Programmer (PCP)
CREATE DATE: 17 Jun 2019
PURPOSE: To demonstrate simple output using JavaScript.
LAST MODIFIED ON: 19 Jun 2019
LAST MODIFIED BY: Donna C. Developer (DCD)
MODIFICATION HISTORY:
18 Jun 2019 - Changed output message to "Hello Mars!"
  to fix confusion over which planet humans inhabit. (PCP)
19 Jun 2019 - Changed output message back to "Hello World!"
  to return to original intent of the program. (DCD)
***************************************/
```

Please note that HTML, CSS, and JS have different styles of comments.

Notes about Data Types

The var keyword is used to delcare variables, but it does not assign them a type. JavaScript utilizes three main data types:

Data Type	Used For	Prefix	Example
String	text	str	**`strUserName`**
Numeric	integers decimals (float)	int flt	**`intYear`** **`fltPrice`**
Boolean	true/false	bol	**`bolIsFinished`**

Please read Chapter 2 of the text for an explanation of data types.

JavaScript is, technically speaking, a *loosely-typed language*. This means it allows a lot of fluidity among strings, integers and other primitives and will allow you, the programmer, to "get away with" using the same value for different types without converting that value. Such practice leads to sloppy programming and can cause errors. As your teacher, one of my goals for this class is to teach you good programming practices that will extend beyond semester's end. Thus, I want you to treat JavaScript as if it were a more strictly-typed language, like Java or C. When you need to convert a value from one data type to another, be sure to use conversion functions.

In JavaScript, the most common ways to convert data are:

- **`parseInt()`** function converts ***to*** an integer type
- **`parseFloat()`** function converts ***to*** a float type
- **`Object.toString()`** method convert to a string type.

Here's a quick example of conversions (**IMPORTANT NOTE:** JavaScript does not have separate constructors for integers and floats. Instead, we use the `Number()` constructor method for both types of numbers):

```
var strUserAnswer = "";   // string that holds the user's answer to two questions
var intA = 0;   // integer that holds a converted number to be used for addition
var fltB = 0.0;   // integer that holds a converted number to be used for addition
var fltSum = 0.0;   // floating-point that holds a converted number for addition
var strReturnSum = "";   // fltSum converted to a string for output

/* strDefaultText is a string that holds the help text for the prompt window */
var strDefaultText = "TYPE YOUR ANSWER HERE.";

/* Ask user for an integer. User response is returned as a string-type */
strUserAnswer = window.prompt("Enter an integer number:", strDefaultText);

intA = parseInt(strUserAnswer);   // Convert string to an integer using parseInt()

/* Ask user for a decimal number. User response is returned as a string-type */
strUserAnswer = window.prompt("Enter a float number:", strDefaultText);

fltB = parseFloat(strUserAnswer);   // Convert string to an floating-point using parseFloat()
fltSum = intA + fltB;   // Add the values of intA and fltB and store it in fltSum

strReturnSum = fltSum.toString();   // Convert float to a string using the .toString() method
window.alert("The sum is " + strReturnSum + ".");   // Output the sum back to the user
```

DevGuru has more information on:

- **the parseInt() function**
- **the parseFloat() function**
- **Object.toString() method**

Notes on Variables

Name your variables by what they represent. For instance, if you know that a variable will store a user's name (a string value), name your variable **`strUserName`** instead of something like **`stringValue1`**. Also, use prefixes to indicate data type. For example in the variable name **`strUserName`**, the prefix "**`str`**" indicates the variable is a string-type variable). Other good prefixes include **`int`** for integer data types, **`flt`** for floating-point data types and **`bol`** for Boolean data types.

Use lower camelCasing when naming your variables. Since we don't use spaces in variable names, you can capitalize the first letter of subsequent words in a multiple-word variable name: **`strUserName`**

Although it is technically possible to create variables in JavaScript without **using the keyword "var"**, in this class, it will be required. Note: you can use constructor methods to declare variables. What you CANNOT do is utilize the weak typing feature of JavaScript to infer variable declarations.

When you create a variable, precede or follow the variable declaration with a **comment**, explaining the variable, its type and how the variable is going to be used:

```
/* The following variable is a string-type variable that will be
used to store the user's name throughout the program */
var strUserName = "John Q. Student";

var strUserName = "John Q. Student";   // string variable to hold user's name
```

Commenting Functions

You need to comment each function you create (except for the main function). In doing so, be sure to:

- identify the name of the function
- describe the purpose of the function
- identify the parameters it takes (if any)
- describe the value it returns (if any)

Here's an example of a well-formed function comment:

```
/**************************************
NAME: CalcShipCost
PURPOSE:
  Calculates and returns the shipping cost of an object. The cost is based on
  the weight of the object and the distance the object is shipped.
PARAMETERS:
  distance - mileage between two points
  weight - initial weight of object
RETURN VALUE:
  shippingCost - the price of shipping an object of a specific weight from point A to point B
***************************************/
```

HTML5 Envelope

ALL lab work should be hosted from a compliant HTML5 page. Compliance should be tested by passing the validation standards accessible at the **W3C Markup Validation Services Site**.

In this class, the content layer of all pages should be written in HTML. This class is all about client-side programming, so we will not accept work in PHP. If you would like to experience a formal introduction to HTML, please consider taking **CSCI N241: Fundamentals of Web Development**.

Commenting Code

Be sure to include *detailed comments* that "tell the story" of your code. A good guideline to follow is to:

1. Comment each variable as you declare it - noting its purpose.
2. Comment each major structure (decision structures, looping structures, etc.), describing *why* you are using that structure.

Your source code should be easy to read and maintain. To ensure this, be sure to use plenty of appropriate white space in your code and indent the lines of a structure's body. Examples of code readability follow:

GOOD Example of Readability:

```
if(grade >= 70)
{
   printf("Passed\n");
}
else
{
   printf("Failed\n");
}
```

Poor Example of Readability #1 (DO NOT USE!):

```
if(grade >= 70)
{
printf("Passed\n");   // Notice that this line is not indented.
}
else
{
printf("Failed\n");
}
```

Poor Example of Readability #2 (DO NOT USE!):

```
/* This works just like the other two examples, but it is very difficult to read! */
if(grade >= 70){printf("Passed\n");}else{printf("Failed\n");}
```

Important Procedures for All Labs

Here are some general notes for perfection that you should follow for every assignment:

1. Please produce all web content to HTML5 standards.
2. Please **validate** all your files.
3. Be sure to update the header block comments for each file.
4. Be sure to check your browser's console / developer tools for error free code.
5. Test your code in Chrome and Edge at a minimum.
6. Use only your own original code for all labs.
7. Be sure to put your CSS and JavaScript in a separate files from your html.
8. Be sure to read through the lab rubric in Canvas.
9. Submit your lab in Canvas for grading.

Mission Accomplished!

Be sure to keep these important guidelines in mind throughout the semester, and holler if you have any questions!