CSPC57 MIDTERMS

HTML Forms

- enables to build web pages that let users actually enter information and sent it back to the server. It is an area that can contain form elements.

Form Elements

- controls that allow the user to enter information (like text fields, textarea fields, frop-down menus. Radio buttons, checkboxes, command buttons etc.) in a form.
- There could range from a single text box for entering a search string common to all search engines on a web to a complex multipart worksheets that offers powerful submissions capabilities.
- **<form></form>** - to activate the form portion of your html document.

<input> (<input/>)
- it is used to select user information.
- An input field can vary in many ways, depending on the type attribute.
- An input field can be a **text field**, a **checkbox**, a **password field**, a **radio button**, a **button**, and more.

Users <input> types

text
- It is the default, with size used to specify the default size of the text that is created.

password
- a text field with the user input displayed as asterisks or bullets for security.
- **maxlength** is used to specify the maximum number of characters entered in the password

checkbox
- offers a single (ungrouped) checkbox; checked enables to specify whether or not the box should be checked by default.
- **value** specifies the text associated with the checkbox.

hidden
- enables you to send information to the program processing the user input without the user actually seeing it on display.
- Particularly useful in Server Side Scripting.

file
- gives you a way to let users actually submit files to the server.

radio
- displays toggle button; different radio buttons with the same name & value are grouped automatically, so that only one button in the group can be selected.

submit
- which produces a push button on the form that, when clicked, submits the entire form content to the remote server or to execute a client-side script.

reset
- which enables the users to clear the contents of all fields in the form.

image
- which is identical to submit, but instead of button, enables you to specify a graphical image for the submission or enter button.

Attribute	Value	Description
type	button checkbox file hidden image password radio reset submit text	Specifies the type of an input element
checked	checked	Specifies that an input element should be preselected when the page loads (for type="checkbox" or type="radio")
disabled	disabled	Specifies that an input element should be disabled when the page loads
maxlength	number	Specifies the maximum lengths (in characters) or an input field (for type="text" of type="password")
name	name	Specifies the name for an input element
readonly	readonly	Specifies that an input field should be read only (for type="text" of type="password")
size	number	Specifies the width of an input field
src	URL	Specifies the URL to an image to display as a submit button
value	value	Specifies the value of an elements

<select></select>
- is used to create a select list (drop-down list). the **<option>** tag is used to define the available option in the list.

```
<select>
        <option value="volvo">Volvo</option>
        <option value="saab">Saab</option>
        <option value="mercedes">Mercedes</option>
        <option value="audi">Audi</option>
</select>
```

<textarea></textarea>
- defines a multi-line text input control. A text area can hold an unlimited number of chaacters, and the text renders in a fixed-width font (usually courier).

Attribute	Value	Description
cols	number	Specifies the visible width of a text-area
rows	number	Specifies the visible number of rows in a text-area
disabled	disabled	Specifies that a text-area should be disabled
name	name	Specifies the name for a text-area
readonly	readonly	Specifies that a text-area should be read only

```
<textarea rows="2" cols="20">
        At W3Schoools you will find all the Web-building tutorials you need, from basic html to advance XML, SQL, ASP and PHP
</textarea>
```

JavaScript

- JavaScript is a lightweight, interpreted programming language.
- JavaScript was first known as LiveScript.
- It is designed for creating network-centric applications.
- It is complimentary to and integrated with Java.
- JavaScript is very easy to implement because it is integrated with HTML.
- It is open and cross-platform.

Why to learn JavaScript?

- JavaScript is the most popular programming language in the world and that makes it a programmer's great choice.
- JavaScript is everywhere, it comes installed on every modern web browser and so to learn JavaScript you really do not need any special environment setup.
- JavaScript helps you create really beautiful and crazy fast websites.
- JavaScript usage has now extended to mobile app development, desktop app development, and game development.
- Due to high demand, there is tons of job growth and high pay for those who know JavaScript.
- Great thing about JavaScript is that you will find tons of frameworks and Libraries already developed which can be used directly in your software development to reduce your time to market.

Applications of JavaScript Programming

- Client side validation
- Manipulating HTML Pages
- User Notifications
- Back-end Data Loading
- Presentations
- Server Applications

Client-side JavaScript

- Client-side JavaScript is the most common form of the language.
- The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.
- The JavaScript client-side mechanism provides many advantages over traditional CGI server-side scripts.

Advantages of JavaScript

- Less server interaction
- Immediate feedback to the visitors
- Increased interactivity
- Richer interfaces

Limitations of JavaScript

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multi-threading or multiprocessor capabilities.

JavaScript Code

<script> tags
- The script tag alerts the browser program to start interpreting all the text between these tags as a script.

Syntax:

```
<script...>
        JavaScript code
</script>
```

Two Important Attributes of Script Tag

Language
- This attribute specifies what scripting language you are using. Typically, its value will be javascript.

Type
- this attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".

```
<script language = "javascript" type="text/javascript>
        JavaScript code
</script>
```

Hello World using JavaScriot

```
<html>
  <body>
     <script language = “javascript” type=”text/javascript>
          document.write(“Hello World!”)
     </script>
   </body>
</html>
```

JavaScript Output

document.write();

- Is a function that is used to display some strings in the output of HTML web pages.

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
document.write(5 + 6);
</script>

</body>
</html>
```

innerHTML

- gets or sets the HTML or XML markup contained within the element.

```
<!DOCTYPE html>
<html>
<body>

<h1>MY First Web Page</h1>
<p>My first paragraph.</p>

<p id=“demo”></p>

<script>
document.getElementById(“demo”).innerHTML = 5
+ 6;
</script>

</body>
</html>
```

window.alert();

- instructs the browser to display a dialog with an optional message, and to wait until the user dismisses the dialog.

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
window.alert(5 + 6);
</script>

</body>
</html>
```

console.log();

- displays messages or variables in the browser’s console.

```
<!DOCTYPE html>
<html>
<body>

<script>
console.log(5 + 6);
</script>

</body>
</html>
```

Whitespace and Line Breaks

- JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs.
- You can use spaces, tabs and newlines freely in your program and you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

Semicolons are optional

- Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++ and Java.
- JavaScript, however, allows you to omit this semicolon if each of your statements are placed on a separate line. For example, the following code could be written without semicolons.

```
<script language = “javascript” type=”text/javascript>
          var1 = 10
          var 2 = 20
</script>
```

```
<script language = “javascript” type=”text/javascript>
          var1 = 10; var 2 = 20;
</script>
```

Case Sensitivity

- JavaScript is a case-sensitive language.
- Case-sensitive – this means that the language keywords, variables, function names, and any other identifiers must always be typed with consistent capitalization of letters.
- Example: Time and TIME will convey different meanings in JavaScript.

Comments in JavaScript

- Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.
- Any text between the characters /* and */ is treated as a comment.

```
<script language = “javascript” type=”text/javascript>
          // This is a comment. It is similar to comments in C++

          /* This is a multi-line comment in JavaScript
          It is very similar to comments in C Programming*/
</script>
```

JavaScript in External File

- The script tag provides a mechanism to allow you to store JavaScript in an external file and then include it into your HTML files.
- Save as filename.js

```
<html>
  <head>
    <script type="text/javascript" src="filename.js">
        document.write("Hello World!")
    </script>
  </head>
   <body>
      .....
   </body>
</html>
```

JavaScript Datatypes

- JavaScript allows you to work with tree primitive data types.
  - Numbers, e.g. 123,120.50 etc.
  - Strings e.g. "this text string" etc.
  - Boolean e.g. true or false

JavaScript Variables

- Variables can be thought of as named containers.
- Variables are declared with the var keyword.
- Variable initialization – storing a value a variable.

```
<script type="text/javascript">

        var money;
        var name;

</script>
```

```
<script type="text/javascript">

        var money, name;

</script>
```

JavaScript Variables Scope

Global Variables

- it means it can be defines anywhere in your JavaScript code.

Local Variables

- visible only within a function where it is defined.

Operators

Arithmetic Operators

- **Addition (+)** adds two operands
- **Subtraction (-)** subtracts the second operand from the first.
- **Multiplication (*)** multiply both operands
- **Division (/)** divide the numerator by the denominator
- **Modulus (%)** outputs the remainder of an integer division
- **Increment (++)** increases an integer value by one
- **Decrement (--)** decreases an integer value by one

Comparison Operators

- **Equal (==)** checks if the values of two operands are equal or not, if yes, then the condition becomes true.
- **Not Equal (!=)** checks if the values of two operands are equal or not, if the values are not equal, then the condition becomes true.
- **Greater than (>)** checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true.
- **Less than (<)** checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true.
- **Greater than or equal to (>=)** checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true.
- **Less than or equal to (<=)** checks if the value of the left operand is less than or equal to the value of right operand, if yes, then the condition becomes true.

Logical Operators

- **Logical AND (&&)** if both the operands are nom-zero, then the condition becomes true.
- **Logical OR (||)** if any of the two operands are non-zero, then the condition becomes true.
- **Logical NOT (!)** reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false.

Bitwise Operators

- **Bitwise AND (&)** it performs a Boolean AND operation on each bit of its integer arguments.
- **Bitwise OR (|)** it performs a Boolean OR operation on each bit of its integer arguments.
- **Bitwise XOR (^)** it means that either operand one is true or operand two is true, but not both.
- **Bitwise Not (~)** it is unary operator and operates by reversing all the bits in the operand.
- **Left Shift (<<)** it moves all the bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros.
- **Right Shift (>>)** the left operand's value is moved right by the number of bits specified by the right operand.
- **Right Shift with Zero (>>>)** this operator is just like the >> operator, except that the bits shifted in on the left are always zero.

Assignment Operators

- **Simple Assignment (=)** assigns values from the right side operand to the left side operand.
- **Add and Assignment (+=)** it adds the right operand to the left operand and assigns the result to the left operand.
- **Subtract and Assignment (-=)** it subtracts the right operand from the left operand and assigns the result to the left operand.
- **Multiply and Assignment (*=)** it multiplies the right operand with the left operand and assigns the result to the left operand.
- **Divide and Assignment (/=)** it divides the left operand with the right operand and assigns the result to the left operand.

- **Modulus and Assignments (%=)** it takes modulus using two operands and assigns the result to the left operand.

Miscellaneous Operators

- **Conditional Operator (?:)** evaluate an expression for a true or false and then executes one of the two given statements depending upon the result of the evaluation.
- **typeof Operator** – is a unary operator that is placed before its single operand, which can be of any type.

Conditional Statements

If Statement

- The if statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

Syntax:

```
if (expression) {
    Statement(s) to be executed if expression is true
}
```

Example:

```
var age = 20;

    if (age > 18) {
        document.write("<b>Qualifies for driving</b>");
}
```

If...else Statement

- The 'if...else' statement is the next form of control statement that allows JavaScript to execute statement in a more controlled way.

Syntax:

```
if (expression) {
    Statement(s) to be executed if expression is true
}
else
{
    Statement(s) to be executed if expression is false
}
```

Example:

```
var age = 15;

    if (age > 18) {
        document.write("<b>Qualifies for driving</b>");
    }
    else {
        document.write("<b>Does not qualify for driving</b>");
    }
```

If...else if... Statement

- The if...else if... statement is an advanced form of if...else that allows JavaScript to make a correct decision out of several conditions.

Syntax:

```
if (expression 1) {
    Statement(s) to be executed if expression 1 is true
} else if (expression 2) {
    Statement(s) to be executed if expression 2 is true
} else if (expression 3) {
    Statement(s) to be executed if expression 3 is true
} else {
    Statement(s) to be executed if no expression is true
}
```

Example:

```
var book = "maths";
    if (book == "history")
    {
        document.write("<b>History Book</b>");
    }
    else if (book == "maths")
    {
        document.write("<b>Maths Book</b>");
    }
    else if (book == "economics")
    {
        document.write("<b>Economics Book</b>");
    }
    else
    {
        document.write("<b>Unknown Book</b>");
    }
```

Switch Statement

- The objective of a switch statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.

Syntax:

```
switch (expression)
{
    case condition 1: statement(s)
    break;

    case condition 2: statement(s)
    break;

    ...

    case condition n: statement(s)
    break;

    default: statement(s)
}
```

Break Statements

- The break statements indicate the end of a particular case. If they were omitted, the interpreter would continue executing each statement in each of the following cases

Looping Statements

While Loop

- The purpose of a while loop is to execute a statement or code block repeatedly as long as an expression is true.
- Once the expression becomes false, the loop terminates.

Syntax:

```
while (expression) {
    Statement(s) to be executed if expression is true
}
```

Example:

```
<html>
  <body>
     <script type="text/javascript">

          var count = 0;
            document.write("Starting Loop ");

            while (count < 10){
              document.write("Current count: " + count + "<br />");
              count++;
            }

            document.write("Loop stopped!");

       </script>

       <p>Set the variable to different value and then try....</p>
   </body>
</html>
```

Output:

```
Starting Loop
Current count: 0
Current count: 1
Current count: 2
Current count: 3
Current count: 4
Current count: 5
Current count: 6
Current count: 7
Current count: 8
Current count: 9
Loop stopped!
Set the variable to different value and then try....
```

Do...while Loop

- The do...while loop is similar to the while loop except that the condition check happens at the end of the loop.
- This means that the loop will always be executed at least once, even if the condition is false.

Syntax:

```
do {
    Statement(s) to be executed;
} while (expression);
```

Example:

```
<html>
  <body>
     <script type="text/javascript">

          var count = 0;
          document.write("Starting Loop " + "<br />");
          do{
            document.write("Current count: " + count + "<br />");
            count++;
          }
          while (count < 5);
          document.write("Loop stopped!);

       </script>

       <p>Set the variable to different value and then try....</p>
   </body>
</html>
```

Output:

```
Starting Loop
Current count: 0
Current count: 1
Current count: 2
Current count: 3
Current count: 4
Loop stopped!
Set the variable to different value and then try....
```

For Loop

- The '**for**' loop is the most compact of looping. It includes the following three important parts:
- The **loop initialization** where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The **test statement** which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.
- The **iteration statement** where you can increase or decrease your counter.
- You can put all the three parts in a single line separated by semicolons.

Syntax:

```
foe (initialization; test condition; iteration statement) {
    Statement(s) to be executed if test condition is true
}
```

Example:

```
<html>
  <body>
     <script type=”text/javascript”>

          var count;
          document.write(“Starting Loop ” + “<br />”);

          for(count = 0; count < 10; count++) {
            document.write(“Current count: ” + count);
            document.write(“<br /> ”);
          }

          document.write(“Loop stopped!);

     </script>

     <p>Set the variable to different value and then try....</p>
  </body>
</html>
```

Output:

```
Starting Loop
Current count: 0
Current count: 1
Current count: 2
Current count: 3
Current count: 4
Current count: 5
Current count: 6
Current count: 7
Current count: 8
Current count: 9
Loop stopped!
Set the variable to different value and then try
```

JavaScript – Loop Control

The break statement

- The **break** statement, which was briefly introduced with the switch statement, is used to exit a loop early, breaking out of the enclosing curly braces.

Example:

```
<html>
  <body>
     <script type=”text/javascript”>

          var x = 1;
          document.write(“Entering the loop<br /> ”);

          while (x < 20)
          {
            if (x == 5) {
               break; // breaks out of loop completely
            }
            x = x + 1;
            document.write(x + “<br /> ”);
          }
          document.write(“Exiting the loop!<br />”);
     </script>

     <p>Set the variable to different value and then try....</p>
  </body>
</html>
```

Output:

```
Entering the loop
2
3
4
5
Exiting the loop!

Set the variable to different value and then try....
```

Continue Statement

- The **continue** statement tells the interpreter to immediately start the next iteration of the loop and skip the remaining code block.
- When a **continue** statement is encountered, the program flow moves to the loop check expression immediately and if the condition remains true, then it starts the next iteration, otherwise the control comes out of the loop.

Example:

```
<html>
  <body>
     <script type=”text/javascript”>

          var x = 1;
          document.write(“Entering the loop<br /> ”);

          while (x < 10)
          {
            x = x + 1;
            if (x == 5) {
               continue; // skip rest of the loop body
            }
            document.write(x + “<br /> ”);
          }
          document.write(“Exiting the loop!<br />”);
     </script>

     <p>Set the variable to different value and then try....</p>
  </body>
</html>
```

Output:

```
Entering the loop
2
3
4
6
7
8
9
10
Exiting the loop!
Set the variable to different value and then try....
```

Using Labels to Control the Flow

- A **label** is simply an identifier followed by a colon(:) that is applied to a statement or a block of code.
- **Note –** Line breaks are not aloowed between the **‘continue’** or **‘break’** statement and its label name. Also, there should not be any other statement in between a label name and associated loop.

JavaScript Function

Function
- in JavaScript is by using the function keyword, followed by a **unique function name**, a **list of parameters** (that might be empty), and a **statement block surrounded by curly braces.**
- is a group of reusable code which can be called anywhere in your program.
- This eliminates the need of writing the same code again and again and It helps programmers in writing modular codes.
- allow a programmer to divide a big program into a number of small and manageable function.

Syntax:
The basic syntax is shown here.

```
<script type="text/javascript">
<!--
function functionname(parameter-list)
{
  statements
}
//-->
</script>
```

Example:

```
<script type="text/javascript">

        function sayHello() {
            alert("Hello there");
        }
</script>
```

Function Parameters
- we have seen functions **without parameters** but there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters.
- A function can take multiple parameters separated by comma.

Return Statement
- A JavaScript function can have an optional return statement. This is required if you want to return a value from a function.
- This statement should be the last statement in a function.

Output Display Functions

console.log()
- accepts a "**Parameters**," which can be any message, object, or array, and it then returns or outputs the value of the given parameter to the console window.

document.write()
- function is used for inserting any content or JavaScript code in a document.
- removes all existing content from the HTML document and replaces it with the new content added as a parameter.
- this method is mainly employed for testing purposes.

document.getElementById()
- in JavaScript is utilized for returning the Element object, whose "**Id**" gets matched with the provided string.
- The Element Ids are useful for quickly accessing the specified element, and thus they can be used with the ".innerHTML" property to write or display its output.

window.alert()
- method displays output or any information to the user where it shows the added message in an alert box having the "OK" button.
- forces the user to read the displayed message, and the user cannot access the web page until the alert box is closed

JavaScript Errors

Syntax Errors
- also called **parsing errors**, occur at compile time in traditional programming languages and at interpret time in JavaScript.

Runtime Errors
- also called **exceptions**, occur during execution (after compilation/interpretation).

Logical Errors
- can be the most difficult type of errors to track down because these errors are not the result of a syntax or runtime error. They occur when you make a mistake in the logic that drives your script and you do not get the result you expected.

Try...Catch...Finally Statement

- The latest versions of JavaScript added exception handling capabilities. JavaScript implements the try...catch...finally construct as well as the throw operator to handle exceptions.
- You can catch programmer-generated and runtime exceptions, but you cannot catch JavaScript syntax errors.
- The **try block** must be followed by either exactly one **catch block** or one **finally block** (or one of both).
- When an exception occurs in the **ty block**, the exception is placed in **e** and the **catch block** is executed. The optional **finally** block executes unconditionally after **try/catch.**

good luck, CS 1-2!