CSC 211 – INTRODUCTION TO WEB PROGRAMMING I

PART I – JAVASCRIPT

BY

MUHAMMAD S. ALI

18/02/2020

Programming Constructs

- Conditional expressions are expressions that evaluate to either **true** or **false** based on the result of a comparison between two or more values. You use these expressions in if statements as well as in looping statements.
- **The relational operators**

  Equal to (==), Not equal to (!=), Greater than (>), Less than (<), Greater than or equal to (>=), Less than or equal to (<=)
- A conditional expression uses the relational operators to compare the results of two expressions and return a Boolean value.
- Because floating-point numbers are not stored as exact values, you shouldn't use the equal to (==) or not equal to (!=) operators to compare them.
- The **isNaN()** method tests whether a string can be converted to a number. It returns true if the string is not a number and false if the string is a number.

- **Conditional expressions**
  - ✓lastName **==** "Ali"
  - ✓testScore **==** 10
  - ✓firstName **!=** "Muhammad Salisu"
  - ✓months **!=** 0
  - ✓testScore **>** 100
  - ✓age **<** 18
  - ✓distance **>=** limit
  - ✓stock **<=** reorder_point
  - ✓rate / 100 **>=** 0.1
- **The syntax of the global isNaN() method**
  - ✓isNaN(expression)
  - ✓isNaN(“Ali") // Returns true since “Ali" is not a number
  - ✓isNaN("123.45") // Returns false since "123.45" can be converted to a number

Logical operators

- AND (&&), OR (||), NOT (!)
- A compound conditional expression joins two or more conditional expressions using the logical operators.
- If logical operators are used to join two or more conditional expressions, the sequence in which the operations are performed is determined by the order of precedence of the operators. To clarify or change the order of precedence, you can use parentheses.
- The **AND** and **OR** operators only evaluate the second expression if necessary. As a result, they are known as *short-circuit* operators.
- **The order of precedence for the logical operators**

  1. NOT operator 2. AND operator 3. OR operator
- Examples
  - The AND operator
    - ✓ age > 17 && score < 70
  - The OR operator
    - ✓ isNaN(rate) || rate < 0
  - The NOT operator
    - ✓ !isNaN(age)

How to code control statements

- An **if** statement always has one **if** clause. It can also have one or more **else if** clauses and one **else** clause at the end.
- The statements in a clause are executed when its *condition* is **true**. Otherwise, control passes to the next clause. If none of the conditions in the preceding clauses are true, the statements in the else clause are executed.
- If necessary, you can code one if statement within the if, else if, or else clause of another if statement. This is referred to as **nesting if** statements.
- **The syntax of the if statement**

```
if ( condition-1 ) { statements }
[ else if ( condition-2 ) { statements }
  ...
else if ( condition-n ) { statements } ]
[ else { statements } ]
```

- **An if statement with an else clause**

```
if(age >= 18) {
        alert ("You may vote.");
} else {
        alert ("You are not old enough to vote.");
}
```

- **An if statement with else if and else clauses**

```
if ( isNaN(rate) ) {
        alert ("You did not provide a number for the rate.");
 } else if ( rate < 0 ) {
        alert ("The rate may not be less than zero.");
} else if ( rate > 12 ) {
        alert ("The rate may not be greater than 12.");
} else {
        alert ("The rate is: " + rate + “.”);
}
```

- **An if statement with a compound conditional expression**

```
if ( isNaN(userEntry) || userEntry <= 0 ) {
      alert ("Please enter a valid number greater than zero.");
}
```

- **Two ways to test whether a Boolean variable is true**

```
if ( isValid == true ) { }
if ( isValid ) { }                    // same as isValid == true
```

- **Three ways to test whether a Boolean variable is false**

```
if ( isValid == false ) { }
if ( !isValid == true ) { }
if ( !isValid ) { }                   // same as !isValid == true
```

- **How to get value from confirm box**

```
var answer = confirm("Are you sure you want to do that?");
 if (answer == true) {
    alert("You're sure");
 } else {
    alert("You decided against");
 }
```

How to code while and do while loops

- The *while* statement creates a **while** loop that contains a block of code that is executed while its condition is **true**. This condition is tested at the beginning of the loop, and the loop is skipped if the condition is false.
- The do-while statement creates a **do-while** loop that contains a block of code that is executed while its condition is true. However, its condition is tested at the end of the loop instead of the beginning, so the code in the loop will always be executed at least once.
- If the condition for a while or do-while loop never evaluates to false, the loop never ends. This is known as an infinite loop. You can end an infinite loop by closing the tab or browser window.
- **The syntax of a while loop**

  while (condition) { statements }
- **The syntax of a do-while loop**

  do { statements } while (condition);

- Examples
- **A while loop that adds the numbers from 1 through 5**

```
var sum = 0;
var numberOfLoops = 5;
var counter = 1;
while (counter <= numberOfLoops) {
      sum += counter;
      counter++;
}
alert(sum);
```

- **A do-while loop that adds the numbers from 1 through 5**

```
var sum = 0;
var numberOfLoops = 5;
var counter = 1;
do {
      sumOfNumbers += counter;
      counter++;
}while (counter <= numberOfLoops);
alert(sum);
```

How to code for loops

- The for statement is used when you need to increment or decrement a counter that determines how many times the for loop is executed.
- Within the parentheses of a for statement, you code an expression that initializes a counter (or index) variable, a conditional expression that determines when the loop ends, and an increment expression that indicates how the counter should be incremented or decremented each time through the loop.
- The variable name i is commonly used for the counter in a for loop.
- **The syntax of a for statement**

```
for ( counterInitialization; condition; incrementExpression ) {
        statements
}
```

- Examples
- **A for loop that adds the numbers from 1 through 9**

```
var sum = 0;
var numberOfLoops = 9;
for (var counter = 1; counter <= numberOfLoops; counter++) {
      sum += counter;
}
alert(sum);
```

Demo App

Arrays

- An **array** can store one or more elements. The **length** of an array is the number of elements in the array.
- One way to create an array is to use the new keyword, the name of the object (Array), and an optional length parameter.
- The other way to create an array is to code a set of brackets.
- To refer to the elements in an array, you use an *index* where 0 is the first element, 1 is the second element, and so on.
- One way to add an element to the end of an array is to use the **length** property as the index.
- **The syntax for creating an array Using the new keyword with the Array object name**

  var arrayName = new Array(length);
- **Using an array literal**

  var arrayName = [];

- **The syntax for referring to an element of an array**

  arrayName[index]

- **The syntax for getting the length property of an array**

  arrayName.length

- **How to add values to an array**

```
var totals = [];
totals[0] = 141.95;
totals[1] = 212.25;
totals[2] = 411;
```

- **How to refer to the elements in an array**

```
totals[2]
totals[1]
```

- **How to determine how many elements are in an array**

```
var count = totals.length;
```

- **How to add a value to the end of an array**

```
totals[totals.length] = 135.75;
```

How to access the elements of an array

- **Code that puts the numbers 1 through 10 into an array**

```
var numbers = [];
for (var i = 0; i < 10; i++) {
        numbers[i] = i + 1;
}
```

- **Code that displays the numbers in the array**

```
var numbersString = "";
for (var i = 0; i < numbers.length; i++) {
        numbersString += numbers[i] + " ";
}
alert (numbersString);
```

- **Code that puts four totals in an array**

```
var totals = [];
totals[0] = 141.95;  totals[1] = 212.25;
totals[2] = 411;  totals[3] = 135.75;
```

Demo App

How to use objects to work with data

- The **window** object is the global object, and JavaScript lets you omit the object name and dot operator when referring to the window object.
- The **document** object is the object that lets you work with the *Document Object Model* (DOM) that represents all of the HTML elements of the page.
- Data in a text box is treated as a *string*. Before you can use it in a calculation, you need to use either the **parseInt()** or **parseFloat()** method to convert it to *numeric data*.
- **NaN** is a value that means "Not a Number". It is returned by the parseInt() and parseFloat() methods when the value that is being parsed is not a number.
- The **getElementById()** method is commonly used to get the object for an HTML element.

Window Object

- **Method of the window object that displays a dialog box**
  - ✓ **confirm(string)** - Displays a dialog box that contains the string in the parameter, an OK button, and cancel button. If the user clicks OK, true is returned. If the user clicks Cancel, false is returned.
- **Two methods of the window object for working with numbers**
  - ✓ **parseInt(string)** - Converts the string that is passed to it to an integer data type and returns that value. If it cannot convert the string to an integer, it returns NaN.
  - ✓ **parseFloat(string)** - Converts the string that is passed to it to a decimal data type and returns that value. If it cannot convert the string to a decimal value, it returns NaN.
- **Examples of window methods**
  - ✓ confirm("Are you sure you want to delete it?");
  - ✓ var entryA = prompt("Enter any value", 12345.6789);
  - ✓ entryA = parseInt(entryA);
  - ✓ var entryB = prompt("Enter any value", 12345.6789);
  - ✓ entryB = parseFloat(entryB);

Document Object

- **Some methods of the document object**
  - ✓**getElementById(id)** - Gets the HTML element that has the id that is passed to it and returns that element.
  - ✓**getElementsByName(name)** – returns a list of elements with the given name.
  - ✓**getElementsByTagName(name)** – Returns a list of elements with given tag name.
  - ✓**write(string)** - Writes the string that is passed to it into the document.
  - ✓**writeln(string)** - Writes the string ending with a new line character.
- **Examples of document methods**

```
var rateBox = document.getElementById("rate");
document.write("Today is " + today.toDateString());
var elements = document.geElementsByTagName(“p”);
```

Working with TextField and Number Functions

- When you use the *getElementById()* method to get a text box, the method returns a Textbox object. Then, you can use its value property to get the value in the box.
- When you assign a numeric value to a variable, a *Number* object is created. Then, you can use the Number methods with the variable.
- **One method of the TextField Object**
  - ✓**focus()** - Moves the cursor into the text field.
- **Two properties of the TextField object**
  - ✓**value** - A string that represents the contents of the text field.
  - ✓**disabled** - A boolean value that controls whether the text field is disabled.
- **One method of the Number Object**
  - ✓**toFixed(digits)** - Returns a string representation of the number after it has been rounded to the number of decimal places in the parameter.

- **How to get value from a text field**
  - ✓var email = document.getElementById("email_address");
  - ✓ email = email.value;
- **How round a value to the given number of decimal places**
  - ✓var salary = parseFloat(document.getElementById("salary").value).toFixed(2);
- **How to set focus to a TextField**
  - ✓getElementById("email_address").focus();

How to Create and Use Custom Functions

- There are two types of functions in JS
  - ✓Function declarations
  - ✓Function expressions
- A function declaration is one that is coded with a name and is not assigned to a variable. This is just another way to code functions.
- In contrast to a function expression, a function declaration is hoisted to the top of the scope that contains it. That means that it does not have to be coded before any statements that call it.
- **The syntax for a function declaration function**

```
functionName(parameters) {
    // statements that run when the function is executed
}
```

- **A function declaration with no parameters that doesn't return a value**

```
function showYear() {
    var today = new Date();
    alert( "The year is " + today.getFullYear() );
}
```

- **When and how to use local and global variables**
- The scope of a variable or function determines what code has access to it.
- Variables that are created inside a function are **local** variables, and local variables can only be referred to by the code within the function.
- Variables created outside of functions are **global** variables, and the code in all functions has access to all global variables.
- If you forget to code the **var** keyword in a variable declaration, the JavaScript engine assumes that the variable is global. This can cause debugging problems.
- In general, it is better to pass local variables from one function to another as parameters than it is to use global variables. That will make your code easier to understand with less chance for errors.

When and how to use strict mode

- The strict mode directive goes at the top of a file or function, before any other code.
- When you are coding in strict mode, if you forget to code the *var* keyword in a variable declaration or if you misspell a variable name that has been declared, the JavaScript engine will throw an error.
- Because strict mode became available with ECMAScript 5, it won't work with IE7, IE8, and IE9. However, if you test your applications in strict mode in a modern browser, you will catch all of your omissions of the var keyword so they won't cause debugging problems in older browsers.
- **Best coding practices**
  - ✓ Use local variables whenever possible.
  - ✓ Use the var keyword to declare all variables.
  - ✓ Use strict mode.
  - ✓ Declare the variables that are used in a function at the start of the function.

Demo App