**JavaScript Notes**

1/14 and 1/16

Unit 1 – Intro to JavaScript Syntax

* object dot method () OR object dot property
  + object = (noun) a thing you can use (ex: document)
  + properties = (adjectives) descriptions
  + methods / functions = (verbs) do something, have parenthesis (ex: write)
  + parameter / argument = inside parenthesis

<head>

<script>

document.write("Hello Class!");

console.log("Hello Class Console");

alert("Hello Class this is an ALERT!!!!!!!!!");

</script>

</head>

<body>

<p id="para1">This is a para 1</p>

<script>

document.getElementById("para1").innerHTML = "Hello Class! - Inner HTML";

</script>

//order matters because runtime script (script runs as page is being loaded so it needs to be after the para1)

</body>

1/22

* URL = uniform resource locator (web page)
* Document object - <https://www.w3schools.com/jsref/dom_obj_document.asp>
* Browser object model
  + Window
    - Document
    - History
    - Screen
    - Navigator
    - Location
* JavaScript can’t change the HTML or CSS code, can only manipulate the document object

Unit 2 – Variables and Data Types

* Variable - a container that can hold a value
  + Let – defines a new variable
  + Const – defines a new variable
  + Var – old way of defining new variable
* Data types – the information we can store <script>

//define a variable

let firstName = "Erica"; //define a variable and assign a String value

let schoolName = 'DMACC';

let counter = 4; //numeric data type

let price = 1.99; //numerical data type

let newCount = "5"; //string data type of 5 bc quotes

let validForm = true; //boolean data type

let className; //data type 'undefined'

const LANGUAGE = "Javascript"; //constant variable should not be changed

LANGUAGE = "JavaLike"; //should not work, throws console error

//Deprecated variable definition

var oldName = "variable"; //older way of defining variables in Javascript

</script>

<body>

<h1>WDV221 Intro to Javascript</h1>

<h3>Unit 2 - Data Types and Variables</h3>

<script>

document.write("<p>" + firstName + "</p>"); //concatenate the parts

document.write(schoolName);

</script>

</body>

* Concatenation = “join”
* Self-documenting code (use variable and function names that mean something)

1/27

* String objects <https://www.w3schools.com/jsref/jsref_obj_string.asp>

<script>

document.write("<p>" + courseName.length + "</p>");

document.write("<p>" + courseName.toLocaleUpperCase() + "</p>");

document.write("<p>" + courseName.includes("WDV") + "</p>")

alert("The courseName contains WDV? " + courseName.includes("WDV"));

</script>

* Git
  + Add
  + Commit
  + push

1/29

Unit 3 – Introduction to Functions

* function – an action / activity (verbs)
  + do one thing
  + self-contained
  + two or more words when naming it
  + can call them multiple times
* how to call a function

1. use its name in a script
2. call with an event handler <https://www.w3schools.com/tags/ref_eventattributes.asp>
   1. HTML event handler attribute
   2. Attach event handler to an object (field)
   3. Attach an event listener to an object

* Function functionName(parameter) {}

<script>

function sayHello()

{alert("Hello Class");}

function writeName()

{document.write("<p>Erica</p>");}

// How to call a function

// 1. use its name in a script

// 2. call with an event handler

sayHello(); //will call the function

//event handlers used to call functions

//1. HTML event handler attribute

//2. attach event handler to an object (field)

//3. attach event listener to an object

function getName()

//note inName is local scope

{let inName = prompt("Please enter your name.");

console.log(inName);}

</script>

</head>

<body>

<h1>WDV 221 Intro to Javascript</h1>

<h2>Unit 3&4 - Functions</h2>

<h3>Function Examples</h3>

<script>

writeName(); //output my name

</script>

<p>

<button onClick="sayHello()">Say Hello!</button>

</p>

<p>

<button onClick="writeName()">Write my name!</button>

</p>

<p>

<button onClick="getName()">Get name</button>

</p>

</body>

* runtime script / rendered = processes as the page is being loaded (top to bottom)

2/5

Unit 4 – Functions – Parameters and Returns

* parameter – the parentheses after the function name
* return – a value sends the value back to where the function was called
  + display the value
  + pass it to a function
  + assign it to a variable

2/10

2/12

Unit 5 – Event Handlers and Style Object

* Define Event Handlers
  + ‘Listen’ for user actions such as mouse movement and keyboard activity
  + Assigned within HTML they are attributes to an element
  + They can be assigned to an HTML element by Javascript as DOM events
  + Multiple event handlers may be assigned to an element
  + Connect HTML to Javascript
  + Usually used to call functions
  + All HTML elements have event handlers
  + Some event handlers are specific to certain elements or groups of elements
* Examples
  + Window events
    - Onload
    - Onunload
  + Form events
    - Onchange
    - Onsubmit
  + Keyboard events
    - Onkeypress
    - Onkeydown
  + Mouse events
    - Onclick
    - Ondoubleclick
    - Onmouseover

**References:**

W3Schools - HTML <http://www.w3schools.com/tags/ref_eventattributes.asp>

W3Schools – DOM <http://www.w3schools.com/js/js_htmldom_events.asp>

HTML5 Event Handlers <http://www.quackit.com/html_5/tags/html_5_event_handler_content_attributes.cfm>

The HTML example below assigns an event handler to a button element. When the button is clicked it will call the Javascript function called displayValue().

**HTML Example**

<button id=”dispAnswer” onclick=”displayValue()”>Get your answer!</button>

The Javascript example below uses Javascript to assign an onclick event handler that will call the displayValue() function to an element on the page. This is called a dynamic assignment.

**Javascript Example – Dynamic event assignment**

<script>

document.getElementById(“dispAnswer”).onlick = displayValue; //reference call to the function

</script>

2/17

Unit 6 – Math Operators and Textfields

**Introduction of Topic**

Javascript is often used to perform calculations on a web page. Some examples include, sale prices, discounts, volume discounts, shipping amounts, accumulations, etc. Javascript uses most of the same math symbols that you already know and generally follows the same order of operations common to mathematics.

**Increment/Decrement Operators**

There are some new commands in Javascript. The increment and decrement operators are used to add 1 or subtract 1 from a variable. When using these operators anytime the variable is called it will add 1 or subtract 1 from the value of the variable. The increment symbol is ‘++’ and the decrement symbol is ‘- -‘.

Use can use these operators as pre-fix or post-fix operators as needed. A pre-fix operator looks like ++count. When you use this variable it will add 1 to the current value and you will use the new value. A post-fix operator looks like count++. When you use this variable you will use the current value and THEN it will add 1 to the variable.

**Combined Operators**

The combined operators are shortcut commands often used on accumulations equations.

Instead of writing totalSales = totalSales + monthlySales; we can write it using a combined operator as totalSales += monthlySales;. They mean the same thing and work the same way. There are combined operators for most of the math operations. The recommended method is the combined operator, but you are welcome to use either.

**Data Conversion**

Because Javascript is a loosely data typed language we have to very careful when doing calculations. It is easy to get String data into variables, even when it looks like numeric values. When a math operator encounters a String value it will often fail, throw an error or produce an unexpected result. None of which look good on a web page application.

We have two tools that we can use to make sure that numerical values are used in calculations. The parseInt( ) function will convert a value from a String data type to an integer IF the value is numeric. For example parseInt( ‘10’) would turn ‘10’, which is a String, to 10. It would fail when attempting to turn ‘ten’ into a numeric value.

The parseFloat( ) function is used when we want to convert a value from String data to numeric that needs a decimal place. For example parseFloat( “123.45”) would turn “123.45” into 123.45.

**Math Object**

Within the Javascript engine of the browser we have a variety of pre-built objects and functions available to us. You can think of these as tools provided by the browser that we can access. One of these tools is the Math object. This object has a variety of properties (pre-defined variables) and functions (pre-defined actions) that we can use. We will discuss some of these in this project.

Page 1073 of the DHTML book.

**Accessing a specific element from the document object**

The document object is what the browser creates and displays after it reads your HTML code. All the HTML elements become objects and HTML attributes become properties. Each HTML element has a corresponding object within the document. For example each <p> is turned into a p object within the document.

Javascript allows us to access a specific object on the document object. We do this in order to modify or use the object and its contents. In order to do this we have to identify which object we wish to use. To do this we identify the HTML element with an ‘id’ attribute. We can then use this id to access the browser’s version of the HTML element within the document object.

**Keywords and commands**

**order of operations** This is how Javascript determines which math operator to process first when

given more than one.

**post-fix** Use the current value of a variable THEN update the value

**pre-fix** Update the current value of the variable THEN use the value

**parseInt( )** A function from the Javascript Global object that will convert a value to an integer

**parseFloat( )** This will convert a value to a decimal number

**accumulators** Equations where we add a new value to an existing value. Then store the new value in the

variable. Also known as running totals.

**Topic 1 Math operators**

* • Most of these are the same as the math you were taught
* • % Modulus operator. This finds the remainder value from a division process
* • + Is addition when both sides are numeric data types. It is concatenation when either side is a String data type.

***Example:***

 <script>

var remainder = 7%2; //This will divide 7 by 2 and capture the remainder of 1

</script>

***Example:***

<script>

var amount1 = “10”;

var amount2 = 20;

var total = amount1 + amount2; //result will be 1020 because amount1 is a String

</script>

**Topic 2 Increment/Decrement operators**

* • ++ Adds 1 to the current value
* • Subtracts one from the current value
* • Can be pre-fix or post-fix operators
  + o post-fix count++ Use the value of count THEN add 1 to count
  + o pre-fix ++count Add 1 to count THEN use the new value of count

***Example:***

<script>

var count = 0;

count++; // post-fix operator, value of count 1

alert(count++); //displays current value of count 1 and then updates to 2

alert(count++); //displays current value of count 2 and then updates to 3

alert(count); //displays current value of count 3

</script>

**Topic 3 Combined Operators**

* + • +=, - =, \*=, etc.
  + • Shortcuts for writing accumulator equations
  + • Combined operator for most math operations

***Example:***

<script>

var totalSales = 0;

function addSales(inSales)

{

totalSales = totalSales + inSales; //add the inSales amount to the total sales variable

totalSales += inSales; //add the inSales amount to the total sales variable

//BOTH of these equations work the same way.

}

</script>

**Topic 4 Data conversion**

* + • Math operations should only be done on numeric values. This is very true when doing addition.
  + • Javascript does not care what type of data is in a variable, so you have to..
  + • We can convert string data that has numeric value into numeric data types
  + • parseInt( *variable* ) will convert the value of the variable into an integer if it has a numeric value
  + • parseFloat( *variable* ) will convert the value of the variable into a decimal number if it has a numeric value in it

Example:

<script>

var amount1 = ‘10’; //String variable with a numeric value

var amount2 = 4; //numeric variable

var amount3 = ’34.56’;

var total = amount1 + amount2; //This will concatenate because amount1 is String

var total = parseInt( amount1 ) + amount2; //converts ‘10’ to 10 and adds the two numbers

amount1 = parseInt( amount1 ); //converts ‘10’ to 10 and then puts it into the variable

alert( parseInt( amount3 ); //displays 34

alert( parseFloat( amount3 ); //displays 34.56 as a numeric value

</script>

**Topic 5 Javascript Math Object**

* + • Provided by Javascript
  + • Provides pre-defined properties and functions (actions) that we can use without having to code
  + • Most functions return a value that you can use in your scripts
  + • Reference page 1073 DHTML Reference

***Example:***

<script>

var randomNum1 = Math.random( ); //puts a random value from 0-1 into variable

var randomNum2 = Math.random( ) \*10; //puts a random value from 0.00 to 10.000 into variable

var randomNum3 = Math.floor( (Math.random( )\*10) +1 ); //puts 1 to 10 into variable

</script>

<script>

var num1 = Math.round(2.345); //rounds value to nearest integer. 2

var num2 = Math.round(3.68); //rounds value to nearst integer. 4

</script>

**Topic 6 The Input object of the document object**

* + • All text fields are created using the input element in HTML
  + • The browsers reads in the HTML and converts HTML elements into objects within the document object
  + • All input elements become input objects on the document object
  + • We can use Javascript to access the input object to get values from it and put values into it.
  + • Reference page 608 DHTML Reference.
  + • <http://www.w3schools.com/jsref/dom_obj_text.asp>

**Topic 7 Access a specific element using Javascript**

* + • The id attribute in HTML becomes the id property for that object
  + • We can use the id property to locate that element within the document object
  + • document.getElementById(“*id*”) will access the object with the id
  + • The text

***Example:***

<input type=”text” id=”fName” name-“fName” />

<script>

var getFirstName = document.getElementById(“fName”).value; //gets the value from the field

document.getElementById(“fName”).value = “Mark”; //assigns a value to the field

</script>

2/24

* Order of Operations
  + Inside parentheses
  + ++,--          (left to right)
  + \* , / , %      (left to right)
  + +,-              (left to right)
  + =, +=, etc.  (left to right)
* Math operators
  + + , - , \* , / ,
  + Modulus  %
* Increment/Decrement
  + ++  Increment the value by 1
  + --   Decrement the value by 1
  + Post-fix operations: count++        Use the current value of count and then increment by 1
  + Pre-fix operations: ++count          Increment the current value of count then use the new value
* Combined Operators
  + +=, -=, \*= , /=
* totalSales = totalSales + monthlySales; 🡪 totalSales += monthlySales;
* Data conversion
  + parseInt( )              Converts a numeric string to a numeric value
  + parseFloat( )          Converts a numeric string to a numeric decimal value
* Javascript Math Object
  + Properties
* PI
  + Methods
* Math.random( )

<https://www.w3schools.com/jsref/jsref_obj_math.asp>

* Input Object of the DOM
  + This represents the browser's version of the <input> element that has been rendered into the document object
  + Use the .value property of the object to access the content input into a textfield
  + The content of the .value property is ALWAYS a string.  Even if the input content is a number, it is considered a string data type.
* Javascript Error
  + NaN  Not a Number. Encountered when you try to use an operator on a non numeric value.

3/9

Unit 7 – If Statements / Comparisons and Data Evaluation

LOGIC

Sequence – a set of steps in order

Selection – decision making – if statements

if(condition)

{true branch} (required)

else

{false branch} (optional)

Iteration – blah

Count= “5”

if(count==5) true

if(count===5) false (data type is not same)

example

if you have less than $9 you can’t order pizza

if you have greater than or equal to $9 you can get dominos pizza

if you have greater than or equal to $10 you can get pizza hut

if you have greater than or equal to $12 you can get caseys pizza

if you have greater than or equal to $16 you can get wagon wheel pizza

code

if(money<9)

{no pizza}

Else

{if(money>=16)

{get dominos || pizza hut || caseys || wagon wheel}

if(money>=12)

{get dominos || pizza hut || caseys}

if(money>=10)

{get dominos || pizza hut}

if(money>=9)

{get dominoes}}

if(money>=9)

{if(money>=16)

{get dominos || pizza hut || caseys || wagon wheel}

if(money>=12)

{get dominos || pizza hut || caseys}

if(money>=10)

{get dominos || pizza hut}

if(money>=9)

{get dominoes}}

else

{no pizza}

* Comparison Operators
  + ==   Equal
  + === Identical   The values are the same and the data types are the same.
  + != Not Equal.
  + x > y Greater than
  + x < y Less than
  + x >= y  Greater than or equal to
  + x <= y  Less than or equal to
* Logical Operators
  + x && y   And operator
  + x || y     Or operator
  + !x            Not Operator
* Conditional Operator
  + Special assignment statement that is a shortcut if statement
  + variableName = (condition)?value1:value2;
* condition followed by question mark, then an expression to execute if the condition is truthy followed by a colon ( : ), and finally the expression to execute if the condition is false
* var voteable = (age < 18)? “Too young”: “Old enough”;
  + milesRating = (miles < 25000)?”Low Miles”:”Average Miles”;
  + If the number of miles is less than 25000 move the ‘Low Miles’ message to the variable, otherwise move the ‘Average Miles’ message to the variable.
* If Statements
  + Unary or single branch if statement.  True branch is ALWAYS required

if ( ***condition*** )

{ this code processed if the condition is true  }

* + Binary or double branch if statement

if ( **condition** )

{ this code processed if the condition is true  }

else

{  this code processed if the condition is false  }

* + Nested if statement.
  + Additional if statements may be placed within either or both branches of an if statement

if   ( **condition** )

{ this code processed if the condition is true

if ( **condition** )

{ this code processed if both if conditions are true }

else

{ this code process if the first if condition is true and the second is false }

 }

else

{ this code processed if the condition is false }

* Switch Statement
  + Used when your condition has multiple values.
  + A more concise way to display an if statement with multiple blocks of code
  + Can be used in place of nested if…else statements
  + Review Switch statement syntax
* String Object Comparisons
  + .toUpperCase( )
  + .toLowerCase( )
* Javascript
  + .innerHTML( )

3/30

Unit 8 – Testing and Data Validation

TDD

Testing is a skill that all developers require. Many developers learn on their own how to test as they gain experience. This occurs as they attempt to prove that the code they have written will work as expected. You get better with experience.

There are two key thoughts in that statement.  The first is **making sure you know what is expected**. Remember this can change as the project develops or user requirements change.

The second is how do you **ensure that all requirements work as expected** no matter how many changes you make to the application? This process has become more difficult and time consuming as applications become larger and more complex. Formalized processes and tools have been developed to improve the efficiency and accuracy of this process. Test Driven Development is one of those processes or techniques that address this issue.

Test Driven Development (TDD) is a process to ensure that your code meets defined expectations. It not just testing! It is testing based upon requirements in order to ensure that all requirements have been met. If requirements change, then new tests are developed to meet the changing expectations. Once the code is modified ALL tests are processed to ensure the new code works as expected AND that all existing requirements work as expected. In other words "you didn't screw up the rest of the program when you added/fixed the new stuff".

This is a basic overview and implementation of TDD concepts.  There is a lot more to TDD. As you do more TDD please research TDD and expand your working knowledge of TDD.

**Key Terms:**

TDD: Test Driven Development. A formal process for ensuring your application meets all DEFINED expectations.

Unit/Process: the functionality or application process being tested.  Must be well defined, small and can work in isolation.

Test: the smallest situation and expected result that can be defined. Examples of a test.

* "If you enter 'Mark' the process should return 'true'" otherwise it should return 'false'.
* "If you enter all spaces the process should return 'Invalid content', otherwise it will return 'true'.
* "If the entered amount is greater than the available inventory the process should return "Not enough inventory", otherwise it will continue processing the input.
* "If you select Green the background color of the body element should be changed to green".

Test Case: a collection of tests that reflect the expectations of the process being tested

Green/Red: pass/fail indicators.  All Green means all test cases passed successfully

Refactor/Refactoring: changing the application code to meet the requirement and pass a test. This is the term for adding/modifying the code as you code for more tests and expectations.

Regression: testing all previous expectations when adding a new test or updating requirements. Ensures that new changes do not cause problems to other requirements.

Test Runner: a sotware tool/framework used to automate the testing process. Common Javascript testing frameworks include Mocha, Karma, Jasmine, etc.

Assertion: the expected result of a test used as part of the automated testing process. Some Javascript assertion libraries include Chai, etc.

**The 'steps' of using TDD**

* Define the process's expectations/requirements.  What the process/function/application is expected to do.
  + The more detailed the expectations the better.
  + This works best on small well defined processes/functions.
* Design a test to meet a defined requirement.
* Select an input that will FAIL the test.
* Write code that meets the requirement.
* Test the selected input.
  + It should FAIL/RED the test.
* Select an input that will PASS/GREEN the test.
* Test the selected input.
  + It should pass the test.
  + If not then (refactor) update the application code so that it passes.
  + Repeat this process until the input passes

Data Validation

Data validation is required to maintain the integrity of an applications data. In a website application any form that asks for input from the user will be validated.  The intent is to make sure that the information entered on the form is as good and as accurate as you can make it.

The term "Garbage In Garbage Out" is the guiding principle to data validation.  Stop the bad data before it gets any further.  It is much easier and more efficient to prevent bad data than it is to try and 'fix' bad data.

Validation occurs at multiple levels with an application.  The tools and techniques will vary throughout the application. Some things can only be validated at certain points and other things will be validated repeatedly at multiple levels.  However the goal is always the same.  Ensure the quality, accuracy and security of the data being processed into your application.

Data validation occurs at all the following levels of an application:

1. Using the **User Interface** to visually encourage the user to the enter the expected data
   1. Using the placeholder attribute to provide example formatting or content
   2. The size of fields
   3. Providing examples of valid input formatting or content
   4. The use of colors, fonts and spacing to encourage proper usage
2. The use of **HTML elements and attributes** to ensure the user enters the expected data or restricts their ability to enter invalid data.
   1. Dropdown list
   2. Radio buttons
   3. Checkboxes
   4. Default values
   5. Placeholders
3. **HTML5 added numerous attributes** on the input element to enhance data validation in HTML.
   1. required attribute - This attribute requires some content.   <https://www.w3schools.com/tags/att_input_required.asp>
   2. type attribute  -  Additional values were provided for the type attribute for specific types of content.  Use as needed.   <https://www.w3schools.com/tags/att_input_type.asp>
      1. type="date"
      2. type="time"
      3. type="url"
   3. pattern attribute - This attribute allows you to assign a Regular Expression for use as a validation pattern.   <https://www.w3schools.com/tags/att_input_pattern.asp>
      1. Regular expressions are a very powerful tool and one that most developers will use. This topic will be covered in a later course.
4. **Client Side validation with Javascript**
   1. Usually performed as a set of functions when the form is submitted
   2. If all validations are true then the form is submitted
   3. Can change messages on the form as well as visual indicators when an problem is located.
   4. Requires a span element next to the input element for the error message to display. The span element is styles as needed by Javascript
5. **Server Side validation** with server side languages
   1. PHP
   2. Java/JSP
   3. ASP.net
6. **Data validation with SQL** before processing into a database
   1. data types
   2. sanitize data

There are numerous data validation tests. These are common to all languages and implemented as needed.

**Data validations guidelines:**

* Use TDD concepts to create a test plan for coding a validation test.
* Use HTML elements whenever possible to avoid the need to validate user input.
* Use HTML attributes whenever possible to do the validation.  Until you need more advanced techniques.
* Use Javascript to validate user input that cannot be captured by HTML techniques.

**Additional Resources:**

Discusses how to use HTML5 and CSS techniques to enforce data validation as a supplement to Javascript validation:   <https://www.the-art-of-web.com/html/html5-form-validation/>

4/6

Unit 9 – Loops

Whenever you say “I need to do this seven times” you need to use a loop. A LOOP allows you to perform the same action multiple times. The logical name for loops is Repetition. But we call them loops. All programming related languages such Javascript, PHP, Actionscript, etc. have loops.

There are three key pieces of information that you need to remember when working with a loop. If you think about what you need done you can identify the following pieces.

The **Loop Control Variable**: This variable controls the number of times a loop will run. We check the value of this variable to know when to end the loop.

**Check for the Ending Condition**: This is a comparison (IF statement) that ends the loop at the right time. This conditions compares the Loop Control Variable to a value or condition that tells us the loop is finished.

**Update the Loop Control Variable**: Each time through the loop you should update or allow the loop control variable to be updated. If this variable is never changed the loop condition will never end the loop. This results in the ‘endless loop’ effect.

**Loop Control Variable**

There are several ways to use variables to control a loop. Many loops will run a certain number of times. These are called **Counter Controlled** loops. These loops can be based upon the number of rows in an array, the number of records in a row, the number of output lines to display, etc. Counter Controlled loops usually add 1 to a counter variable every time the loop runs. This counter acts as the Loop Control Variable.

Some loops will run until a certain condition is met. These loops are called **Sentinel Controlled** loops. For example: This loop will run until the number of rows is greater than 10. This loop will run until all of the teams have been processed. The loop will run until the customer is done shopping. The loop control variable is usually something besides a number when using a Sentinel Controlled loop.

**Check for the Ending Condition**

You need two pieces of information for this to work. You have to have the Loop Control Variable and you have to have some way of knowing how many times the loop is supposed to run. Sometimes you know the number of times to run, sometimes we base it off of another piece of information such as the length of the array or a response from the user telling us they need to add another item, enter another value, etc.

**Pre-test loops:** These loops will check the end condition BEFORE processing the body of the loop. This ensures that the loop is supposed to run before any processing is done. Most loops are pre-test loops. This includes the FOR and the WHILE loop. It is possible that the loop will not run if the end condition is met before the loop starts.

**Post-test loops:** These loops will check the end condition AFTER processing the body of the loop. This type of loop will always perform the body of the loop at least once before checking the end condition. The DO WHILE loop is an example of this format.

**Update the Loop Control Variable**

The Loop Control Variable is critical to this process. Once you have created the variable you need to initialize it to a beginning value. In many cases this is 0, but it can be other values based upon what the loop needs to do. You MUST update the Loop Control Variable inside the body of the loop. Failure to properly update the Loop Control Variable can create an ‘infinite loop’. This is a loop the never ends because we have written the wrong ending condition or failed to properly update the Loop Control Variable.

For Loop

This is a counter controlled loop. It is best used when you know the number of times you need the loop to run. The For loop has all three pieces required for a loop. The general format of a for loop looks like this.

for (*initialize the loop control variable ; check the end condition ; update the loop control variable* )

{

*body of the loop; //These commands will be processed every time the loop runs*

}

Working example:

for( var i=1 ;i<=5 ; i++)

{

document.write(“<p>Hello</p>”);

}

//Print out the word Hello five times

for( var i=0; i<names.length; i++)

{

document.write(“<p>Team: “ + names[i] + “</p>”);

}

//Print out a list of team names from the names array

While Loop

This is a sentinel controlled loop. It is best used when you need to run a loop until a certain condition is met. If you don’t know how many times it will run then you are using a Sentinel controlled loop.

Example situations: I need to add these numbers until the user is done. I am going to print out the results for each product the user enters. I am going to keep adding to the shopping cart until the user selects “Checkout”. Etc.

Setting up a While loop takes more planning than the For loop. You have to decide on the loop control variable and figure out the initial value. You have to determine when the loop will end. You also have to figure out how to update the loop control variable.

while(check end condition)

{

*body of Loop //this will run as long as the condition is true*

}

Examples:

This loop will run as long as the rowNumber is less than 10.

var rowNumber = 1; //create and initialize loop control variable

while(rowNumber <= 10) //Run the loop as long as the rowNumber is less than or equal to 10

{

Document.write(“<p>Row number: “ + rowNumber + “</p>”);

rowNumber++; //update the loop control variable inside the body of the loop

}

This loop will run as long as the customer has more products to add. When they are done they will type ‘no’ into the prompt and that will change moreProducts from ‘yes’ to ‘no’.

var moreProducts = “yes”; //create and initialize loop control variable LOOP CONTROL VARIABLE

while(moreProducts == “yes”) //Run the loop until condition is false CHECK CONDITION

{

addProducts(); //run this function until the user does not have any more products

moreProducts = prompt(“Do you have more Products? yes/no); //Update loop control

}

4/13

Unit 10 – Arrays

Arrays are a way of storing a lot variables that are very similar or related. Arrays allow the developer to group these variables together and treat them as a group instead of having to work with them as individual variables.

***For example:*** *The following script list four books as individual variables. This works ok if you only have a few books. What happens when you have 100 books? Do you really want to create a variable for each book?*

<script>

var book1 = “Javascript”;

var book2 = “CSS and Styling”;

var book3 = “HTML5 New Way”;

var book4 = “Advanced CSS Styles”;

…

</script>

Instead of storing numerous variables that are very similar we can group them together into an array. This allows us to store all of the information in one variable called books. The books variable has one row for each book stored in the array. We can use as many rows as we need.

<script>

var books = new Array(); //creates an array variable called books

books[0] = “Javascript”; //these lines load the values into rows of the array

books[1] = “CSS and Styling”;

books[2] = “HTML5 New Way”;

books[3] = “Advanced CSS Styles”;

</script>

**Create an Array variable**

In Javascript we have an Array object that defines, describes and provides tools to work with arrays. When we create an array variable we are really making an Array object. So in the example above we are making a **books** variable. Javascript and the developer can treat the variable as an Array object.

You can find more information about the Array object in the Javascript reference. I would recommend the DHTML Definitive Reference and W3Schools as good starting points.

**Row of an array**

Each value stored in an array variable has its own row. This is very similar in layout and structure to a table or a spreadsheet. Each row in this array has a number or index associated with it. In Javascript like most of the object oriented languages the first row starts with **0**.

**Putting values into an array**

In the example shown above the first book was loaded into row 0 or the first row of the books array. The next book was loaded into row 1 of the array. Because we start counting rows with 0 things seem a little off but it will work out.

The Javascript command to create a new uses the var command to create a new variable. You may call an array variable anything you wish. The keyword ‘new’ tells Javascript that you are creating a new Array object and using the Array( ) method to create the object. The new array object will called whatever name you gave the variable.

Once an array variable is created we can load the rows. In general you load the rows of an array starting with row 0 and work your way up. Some arrays can be loaded at the same time they are created. This works well when you have an array that rarely changes and does not have too many values to be stored.

***For example:*** *The two examples shown below create an array variable called products. The first one creates and loads the array in the same command. The second command creates the array and then manually loads a value into each row of the array.*

<script>

var products = new Array(“Pen”, “Paper”,”Pencil”); //creates and loads the array

var products = new Array( ); //create the array

products[0] = “Pen”; //load the first value

products[1] = “Paper”; //load the second value

products[2] = “Pencil”; //load the third value

</script>

**Getting values out of an array**

You can use a row of an array just like a variable. You use the index or row number to indicate which value you wish to access.

<script>

alert(products); //displays all the values stored in the array

alert( products[1] ); //displays Paper because that is the value stored in row 1

</script>

<script>

document.write( products[0] ); //writes the value of “Pen” out to the document object

document.write( “<p>” + products[2] + “</p>” ); //writes out a paragraph with Pencil

</script>

**Using a loop to output an array**

A common use of an array is to store data and then use the array to create content on a page. One of the most useful properties of the Array object is the .length property. This property allows us to know how many rows are in an array. This makes processing the contents of an array with a loop very convenient. Since we know how many rows we can use the for loop to process the rows or values in an array.

***.length A property of the Javascript Array object. It returns the number of rows in an array***

*Given our products array in the example above review these examples.*

<script> **//displays each value in the array as a paragraph on the page.**

for( x=0; x<products.lenght; x++)

{

document.write( “<p>” + products[x] + “</p>”);

}

</script>

<ul>

<script> **//displays each value in the array as a list item within a list.**

for( x=0; x<products.lenght; x++)

{

document.write( “<li>” + products[x] + “</li>”);

}

</script>

</ul>

<select name=”productsList”>

<script> **//displays each value in the array as an option within a select statement.**

for( x=0; x<products.lenght; x++)

{

document.write( “<option>” + products[x] + “</option>”);

}

</script>

</select>

**Parallel arrays**

In many cases we need to group several pieces of information together for each product or service, etc. In this case we can use multiple arrays that work together. One array will hold the name of the product. Another array will hold the prices of the products. The name and the price are stored in different arrays but they are stored in the same row of those arrays.

We have a table of the following information. We need to put the product and its corresponding price in an array.

|  |  |
| --- | --- |
| **Product** | **Price** |
| Pen | 1.29 |
| Paper | .19 |
| Pencil | .67 |

In Javascript we are going to create two arrays. One for the product and one for the price. The price in row 0 will correspond to the product in row 0, etc. The price array is parallel to the product array. In other words the rows of each array work together like in a spreadsheet.

<script>

var products = new Array( ); //create the product array

products[0] = “Pen”; //load Pen

products[1] = “Paper”; //load Paper

products[2] = “Pencil”; //load Price

var price = new Array( ); //create the price array

price[0] = 1.29 //load the price of a Pen

price[1] = .19; //load the price of a Paper

price[2] = .67; //load the price of a Pencil

</script>

Search an array to find a value

When processing arrays we often need to locate a certain value within the array. Or we need to know if the value is already in the array before we add it to the array. This is called searching the array. A loop is used to read through each row in the array. A if statement is used to compare the value of the current row to the desired value. If they match that value is in the array. If not the value is not located within the array.

For example: Using a loop to process each row of the array looking for a value. In this case is ‘Paper’ already in the product array?

<script>

var products = new Array( ); //create the product array

products[0] = “Pen”; //load Pen

products[1] = “Paper”; //load Paper

products[2] = “Pencil”; //load Price

var requestedProduct = prompt(“What product are you looking for?”,””); //Paper

for(x=0; x<product.length; x++)

{

if( product[x] == requestProduct)

{

alert(requestedProduct + “ is already in the Product array”);

alert(“It is located at row “ + x);

}

}

</script>

For example: Find a product and display its price. This questions uses a loop to search through the product array. If it finds the requested product it will use the parallel array price to display the price that corresponds to the product. If nothing is located it will display a ‘Not Found’ message.

<script>

function findPrice()

{

var inProduct = prompt("What product do you want?",""); //product to search for

var foundRow="not"; //local variable

for(x=0; x<products.length; x++)

{

if(inProduct == products[x])

{

foundRow = x; //located the product at row x

break; //stop the loop and continue with next command

}

}//end loop search for product name

if(foundRow == "not") //check to see if we found the product

{

alert("Sorry, " + inProduct + " is not listed in our products");

}

else

{

alert("The price for " + products[foundRow] + " is " + price[foundRow]);

}

}//end findPrice function

findPrice(); //runs the function

</script>

**Modify values in an array**

Once values are in an array they can be accessed, displayed and updated. In order to change the value of the row you located the desired value either by the row number or index or by search through the array looking for the desired value.

Once you have located the row that contains the value you wish to update you can assign a new value, modify the existing value or replace the current value with “” or 0.

<script>

product[1] = “Printer Paper”; //changes “Paper” to “PrinterPaper”

price[1] += .10; //changes .19 to .29 price[1] = price[1]+.10

product[4] = “Stapler”; //Adds new row to product array

price[4]=4.87; //adds stapler price to the parallel price array

</script>

**Unit 11 – Objects**

**4/29/20**

Define an Object?  This is the first step in understanding how to use objects. It is the beginning of a concept called Object Oriented Programming (OOP).

An Object is a thing, something you can use, a container that holds data and functionality. Examples include: your bookbag, your computer, your car, your house, etc. These are things we can see, touch, put things into or take things out of.  These are objects.

In the programming environment most things are objects once you define them and use them in your script or program. The following describes an object and its content in Javascript and most other languages

**Object** - a thing you can use, a container that holds data or functionality

* Objects names are usually nouns. Often times with two or three words to describe the object and its usage
* *Examples* *include*: variables, objects, arrays, strings, etc.

**Properites** - these are pieces of information that describe an **object**.

* property names are often adjectives as they are used to identify the pieces or descriptions of an object
* *Examples include*: name, address, color, size, modelNumber, arrayLength, etc.

**Methods** - this is an activity or functionality that the object can perform or do

* Method names are usually verbs to describe an action.
* Methods could use the content/properties of the object in their process
* Methods could also be used to modify the contents of the object
* *Examples include*: displayName( ), changeColor( ), setFirstName( ), outputValueUpdate( ), etc.

Object Dot Notation is how we write or use and object and its associated properties and methods.

* Camel case is very commonly used in Javascript when writing variables, object, property and method names.
* In camel case the first letter of the name is lowercase.
* The first letter of each following word is capitalized.  This helps make it easier to read and use the names.
* Examples:
  + objectName.propertyName
  + objectName.methodName()
  + student.firstName
  + masterBookList.displayBookISBN( )