

Arling Rodriguez
MTH 135
December 09, 2013
Final

JavaScript

History

In the 1990s, the World Wide Web was beginning to emerge. At the time, all web pages were created to be static; that is, when you looked at a webpage, you saw it exactly how it was set up to be, there was no way for you to interact with it. Interactivity with web pages required a programming language that would give orders to web pages on what it should do in response to a specific action. In addition to interactivity, there needed to be a language that would allow for responses to be immediate, where a web page did not have to reload to complete a response; in other words, the language had to “be able to run on the same computer as the browser displaying the page.”¹

Netscape Navigator and Internet Explorer were the two browsers that were around at the time. The main goal was to find a way to make it more accessible to non-Java programmers. This language would allow the browser to respond to commands directly without needed to compile a code and without using a plugin; that is, it would make web pages dynamic.² Brendan Eich was hired by Netscape in 1995 to be in charge of designing and implementing such a language.³ Prior to joining the team at Netscape, Eich had worked for seven years at Silicon Graphics and three

¹ Stephen Chapman, *A Brief History of Javascript*, <http://javascript.about.com/od/reference/a/history.htm>, (accessed October 2013).

² Stephen Chapman, *A Brief History of Javascript*, <http://javascript.about.com/od/reference/a/history.htm>, (accessed October 2013).

³ Steve Champeon, JavaScript: How Did We Get Here?, http://www.oreillynet.com/pub/a/javascript/2001/04/06/js_history.html, (April 2001).

years at MicroUnity Systems Engineering. When he was asked to work for Netscape, the company was only about a year old.⁴ With the help of Eich, Netscape was the first to introduce LiveScript, the early name of JavaScript, in 1995. With LiveScript out, people running the latest version of Netscape browser were able to interact with the web pages. Additional names that the language went through were Mocha and LiveWire. Since Netscape had been working with Sun Microsystems, the creators of Java, they both officially announced the creation of JavaScript on December 4, 1995. This language was considered to be a “complement” to HTML and Java.⁵ But it may seem confusing to hear the names Java and JavaScript together. Java does play a part in the making of JavaScript in the sense that some of their code appears similar; however, they are not the same thing.

Java was created by Sun Microsystems. Netscape, which created JavaScript used the name under license, but Sun Microsystems did not create JavaScript. The naming of JavaScript ended up causing a lot of confusion. The truth is that Java and JavaScript are two different things. JavaScript is a scripting language that allows you to alter a web page code fairly easily. It provides interactivity to web pages that simple HTML cannot achieve. You can think of JavaScript as being the part of your browser that controls the look and function of web pages. Java, on the other hand, is a software package that is installed separately. Java allows for running applications and it provides a plugin system that lets applets run in the browser.⁶

⁴ Mary Bellis, *The History of JavaScript*, <http://inventors.about.com/od/jstartinventions/a/JavaScript.htm>, (accessed October 2013).

⁵ Steve Champeon, *JavaScript: How Did We Get Here?*, http://www.oreillynet.com/pub/a/javascript/2001/04/06/js_history.html, (April 2001).

⁶ Paul Ducklin, *Java is not Javascript - tell your friends!*, <http://nakedsecurity.sophos.com/2013/01/16/java-is-not-javascript-tell-your-friends/>, (January 2013).

Upon its release, JavaScript quickly started its own journey. It did not require a compiler and the scripts could be copied and pasted to HTML pages most times with little to no changes made. It is “an object based language that supports built-in objects but no classes or inheritance.”⁷ It supports static objects, which are objects that combine general functions and data constructs; usually these are read only, as well as dynamic objects, which resemble templates. JavaScript can be used to write scripting codes for both browsers and servers.⁸ Browser scripts allow for a more interactive and more responsive web pages.

The success of JavaScript brought about competition for Microsoft. Microsoft attempted to create its own version of JavaScript and they named it Jscript. This language was extremely similar to JavaScript, but they did add new capabilities and it was first introduced with Internet Explorer 3.0 in August 1996.⁹ Soon after this, Netscape submitted JavaScript for standardization and in June 1997 it became ECMA-262; though formally it is named ECMAScript, it is still referred to as JavaScript.¹⁰

With this done, the world was approaching a very important time in history. It was moving away from dial-up and into the broadband era. This allowed web designers to become more intricate with their features on web pages.

⁷ Santosh Kumar, History of JavaScript, <http://www.javascriptstyle.com/history-of-javascript>, (April 2013).

⁸ Santosh Kumar, History of JavaScript, <http://www.javascriptstyle.com/history-of-javascript>, (April 2013).

⁹ Colin Ihrig, The History of JavaScript in a Nutshell, <http://cjihrig.com/blog/the-history-of-javascript-in-a-nutshell/>, (February 2012)

¹⁰ Colin Ihrig, The History of JavaScript in a Nutshell, <http://cjihrig.com/blog/the-history-of-javascript-in-a-nutshell/>, (February 2012)

JavaScript underwent many provisions since it first began. These provisions started with JavaScript 1.0, which is the first version of JavaScript and is supported by Netscape 2.0 and Internet Explorer 3.0. Following that, JavaScript 1.1 was created and introduced in Netscape 3.0 but only some parts were used in Internet Explorer 3.0. Next was JavaScript 1.2 which was supported by Netscape 4.0 and Internet Explorer 4.0. Finally, there was JavaScript 1.3 which was introduced in Netscape 4.06 and is supported by Internet Explorer 4.0 and above.¹¹

Today, JavaScript has become one of the most widely used programming languages. It is mostly used in web development and is normally embedded directly to HTML code.¹² Any website can use JavaScript to manipulate the browser to perform specific tasks and the amazing thing about JavaScript is that it is universal, so any browser that supports JavaScript can use the same code. Though JavaScript may seem simple, the truth is that there are many things that make up this language. In this paper I will cover the Syntax and Semantics of JavaScript, present some program examples in JavaScript, and present an evaluation of JavaScript.

Syntax and Semantics

Javascript has three parts to its specifications. These include: basic syntax, the Document Object Model, and the browser object model. The syntax is the main component of JavaScript; it tells how the code is formed. The syntax for JavaScript is based off of ECMAscript 262 standard. The ECMAscript 262 provides information on how to define variables, how to perform calculations on variables, how to set up loops and functions, and how to define objects. However,

¹¹ JavaScript History, <http://www.scriptingmaster.com/javascript/javascript-history.asp>, (accessed October 2013).

¹² JavaScript History & Information, <http://www.xmluk.org/javascript-history-information.htm>, (accessed October 2013).

it does not explain how the script should come together with both the web page and browser. The Document Object Model, otherwise known as DOM, defines how JavaScript communicates with the web page to extract content from the page for processing, add content to the page, and how to access the style sheet for the page so that the appearance can be changed.¹³ Finally, the Browser Object Model is how the scripts get the information from the browser and pass information back to it. Here, browser writers are able to create their own interface. Many browsers, with the exception of Internet Explorer, have adapted the Firefox way of interfacing to the browser.¹⁴

To create a JavaScript program, the use of `<html>`, `</html>`, `<body>`, `</body>`, `<script>`, and `</script>` are needed. The use of `<html>` and `</html>` indicate the beginning and end of the html code. The use of `<body>` and `</body>` denote the beginning and end of the body of the program. The use of `<script>` and `</script>` are used to indicate the beginning and end of a code. Additionally, you will need to include the language and type of your code. Usually in a JavaScript program the language is javascript and the type text/javascript. A sample of these parts being brought together is the program code that will print out "Hello World!":

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

¹³ Chapman, Stephen. "The Three Parts of JavaScript." *About.com JavaScript*. N.p., n.d. Web. 21 Oct. 2013.

¹⁴ Chapman, Stephen. "The Three Parts of JavaScript." *About.com JavaScript*. N.p., n.d. Web. 21 Oct. 2013.

In the JavaScript language usually semicolons are used in the same way that C, C++, or Java uses them. However, if each statement is written in a separate line, the semicolons can be omitted. Despite having this option available, it is generally a good programming practice to use semicolons.

Comments are also supported by JavaScript and can be written in various ways. For example, anything between `//` and the end of the line will be considered a comment, anything between `/*` and `*/` will also be treated as a comment, `<!--` is seen as an HTML comment opening and `-->` is seen as a comment closing.

JavaScript uses variables in its programs and it requires you to declare a variable before you can use it. In this language, variables are declared by using the **var** keyword. For example, **var** sum or **var** money would be used to declare the variables sum and money. In addition, JavaScript has some rules that need to be followed when naming variables. The rules for naming variables are that you cannot use a reserved word as a variable name, you cannot start a variable name with the numbers 0-9; they must begin with a letter or an underscore. JavaScript is a case-sensitive language. This means that keywords, variables, function names, and any other identifiers within a program must be typed with consistent capitalization of letters. In other words, if you have the variable Sum, SUM, sum, SUM, and sUM, they are all the same word but each one has a different meaning in JavaScript.

JavaScript allows you to work with three data types: numbers, strings, and boolean. Additionally, it has two trivial data types: null and undefined, which defines only a single value. JavaScript also allows for various operators: arithmetic, comparison, logical, assignment, and conditional.

In JavaScript, an expression value can be any type that can be used in JavaScript. The simplest expressions in JavaScript are literals. Examples of literals are:

```
3.9 (a numeric literal)
"Hello!" (a string literal)
false (a boolean literal)
{x:1, y:2} (an object literal)
```

More complicated expressions in JavaScript are combinations of variables, function calls, and other expressions. These can be put together using the operators that are allowed in JavaScript.¹⁵ Another type of expression found in JavaScript is that of a regular expression. This is an object that describes a pattern of characters. They are used to execute pattern-matching and search-and-replace functions on text. The basic syntax of a regular expression is:

```
var patt=/pattern/modifiers;
```

In the previous syntax, pattern specifies the pattern of an expression and modifiers specify if the search is global, case-sensitive, etc..¹⁶

In JavaScript, the equal sign (=) is used to assign values to variables. When this is done, we say that it is an assignment operator. On the left hand of the equal sign is the l-value, which can be variables, array elements, and object properties. On the right hand side of the equal sign is the r-value, which can be an arbitrary value of any type. Putting these together, you have an assignment statement. An example of an assignment statement in JavaScript is as follows:

```
var anInteger = 3;
```

¹⁵ "JavaScript Fundamentals." N.p., n.d. Web. 19 Oct. 2013.

¹⁶ "JavaScript RegExp Object." JavaScript RegExp Object. N.p., n.d. Web. 05 Dec. 2013

JavaScript allows for conditional statements such as the if statement, the if...else statement, the if ...else if...else statement, and the switch statement.¹⁷ I will now show the basic outline of each of these:

If Statement

```
if (condition)
{
    code to be executed if condition is true
}
```

If...else Statement

```
if (condition)
{
    code to be executed if condition is true
}
else
{
    code to be executed if condition is not true
}
```

If...else if...else Statement

```
if (condition1)
{
    code to be executed if condition1 is true
}
else if (condition2)
{
    code to be executed if condition2 is true
}
else
{
    code to be executed if neither condition1 nor
condition 2
    is true
}
```

¹⁷ "JavaScript RegExp Object." JavaScript RegExp Object. N.p., n.d. Web. 05 Dec. 2013

When webpages crash it could be a cause of bad loops constructed in JavaScript. One of the loops allowed in JavaScript is the while loop. This loop will execute over and over again as long as the passed condition evaluates to true and it will stop when it evaluates to false. An example of a while loop is given as follows:¹⁸

```
we want to remove all child nodes from the "body"
element:
```

```
var body = document.body;
while (body.firstChild)
{
    body.removeChild(body.firstChild);
}
```

The statement above is executed as long as the body node has a "firstChild," when there is no child notes left, the condition stops.

An array is a special variable that can hold multiple values at one time. Arrays can hold many values under a single name and they allow you to access values by referring to an index number. JavaScript provides three ways to create an array: regular, condensed, and literal. Suppose you are trying to create an array that holds various car brands. The brands we will be working with in this case are: Saab, Volvo, and BMW. If we were trying to create an array object called myCars using the regular method, it would look something like this:

```
var myCars = new Array();
myCars[0] = "Saab";
myCars[1] = "Volvo";
myCars[2] = "BMW";
```

¹⁸ Padolsey, James. "Looping in JavaScript." *James Padolsey RSS*. N.p., 3 Mar. 2009. Web. 21 Oct. 2013.

In the Condensed method, our array would look like:

```
var myCars = new Array("Saab", "Volvo", "BMW");
```

Finally, if the same array was being created using the literal method it would have this format:

```
var myCars = ["Saab", "Volvo", "BMW"];
```

In JavaScript, all variables are called objects. With this being said, an array is allowed to have variables of different types. This means you can have functions in an array; you can have arrays in an array; you can have objects in an array.

These are just a few of the many things that you can do with JavaScript. In the next section, I will provide a few examples of programs done in JavaScript.

Examples

For this section, I will provide examples of some basic programs in JavaScript. For my programs, I used a website to help me generate and check my codes.¹⁹

In my first example, I will be printing a string. The string I will choose to print is "Hi, my name is Arling Rodriguez." The code for this will look like this:

¹⁹ "JavaScript RegExp Object." JavaScript RegExp Object. N.p., n.d. Web. 05 Dec. 2013

```
<html>
<body>
<script>
document.write("<h1>Hi, my name is Arling Rodriguez.</h1>");
</script>
</body>
</html>
```

The run for this code is:

Result:

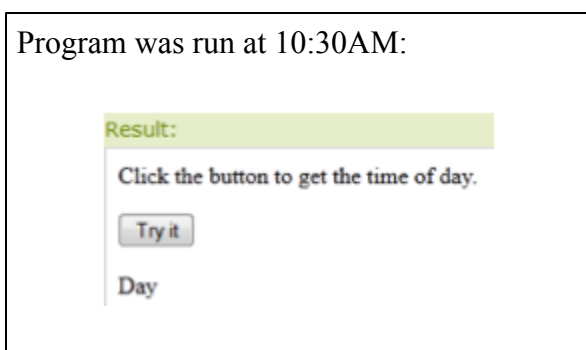
Hi, my name is Arling Rodriguez.

As stated in the syntax and semantics section, we start our JavaScript program with `<html>`, `<body>`, and `<script>`. The `document.write` line tells the program what you want it to print out and we include that in quotation marks so that the program knows to just print it as it is written. Notice that `<h1>` is included before the sentence begins and `</h1>` is included at the end of the sentence; this tells the computer that the letters of this sentence should be made bigger. Finally, we end the program by including `</script>`, `</body>`, and `</html>` to let the computer know that we will not be including any more codes for that specific program.

For my next example, the computer will report either “Day” or “Evening” depending on the time of day when you click the button. This program takes the form of a conditional statement and it would read as follows:

```
<html>
<body>
<p>Click the button to get the time of day.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction()
{
var x="";
var time=new Date().getHours();
if (time<20)
{
x="Day";
}
else
{
x="Evening";
}
document.getElementById("demo").innerHTML=x;
}
</script>
```

The run for this code looks like this:



I have included the button that would appear on a webpage. Since JavaScript is meant to be interactive with the user, the program does not start running until the button is pressed. Once the button is pressed, the program checks the time in military time. Notice that again we use `<html>`, `<body>`, and `<script>` to indicate the beginning of the program and include `</script>`, `</`

body>, </html> to indicate that we have finished the program. Inside the if-then construct, the number of hours are being compared to twenty to see if the hour is less than 20 (before 8pm), then the response you will get will be “Day,” and if the hour is greater than or equal to 20 (8pm and on) then the response you will get will be “Evening.”

In my next program, I will use a function. This function will read in values for three different variables and perform a calculation. The code is as follows:

```
<html>
<body>
<p>This example calls a function which performs a calculation, and returns
the result:</p>
<p id="demo"></p>
<script>
function myFunction(a,b,c)
{
return (a*b) + b - c;
}
document.getElementById("demo").innerHTML=myFunction(7,3,2);
</script>
</body>
</html>
```

The result for this code is:

```
This example calls a function which performs a calculation, and
returns the result:
22
```

What this program does is that it assigns $a = 7$, $b = 3$, and $c = 2$. Then it performs the calculation $(a*b) + b - c$, which in other words is $(7*3) + 3 - 2$. Again, as a standard format for JavaScript, <html>, < body>, <script>, </script>, </body>, </html> are included.

In this example, we will be working with arrays. What this program will do is that it will read in numbers and it will sort them in ascending order. The code for this program will be as follows:

```
<html>
<body>
<p id="demo">Click the button to sort the array.</p>
<button onclick="myFunction()">Try it</button>
<script>
function myFunction()
{
var points = [40,100,1,5,25,10];
points.sort(function(a,b){return a-b});
var x=document.getElementById("demo");
x.innerHTML=points;
}
</script>
</body>
</html>
```

The run for this program is:

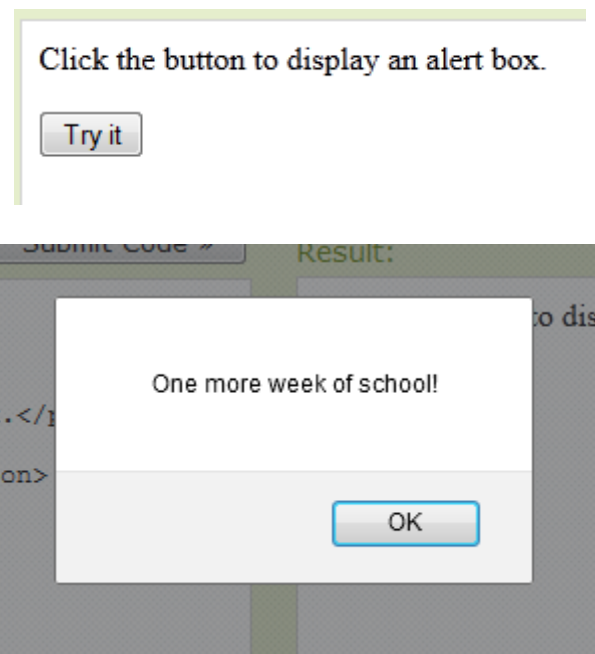
1,5,10,25,40,100

Notice that this program reads in the numbers 40, 100, 1, 5, 25, and 10 to begin with. After being sorted, the output will be 1, 5, 10, 25, 40, and 100, which you can verify is correct.

JavaScript is known to make webpages interactive. Thus, in this example, I will be creating an alert box that will pop up when the user clicks a button. This specific alert box will say: “One more week of school!” The code for this is as follows:

```
<html>
<body>
<p>Click the button to display an alert box.</p>
<button onclick="myFunction()">Try it</button>
<script>
function myFunction()
{
alert("One more week of school!");
}
</script>
</body>
</html>
```

The run for this program is shown in two parts:



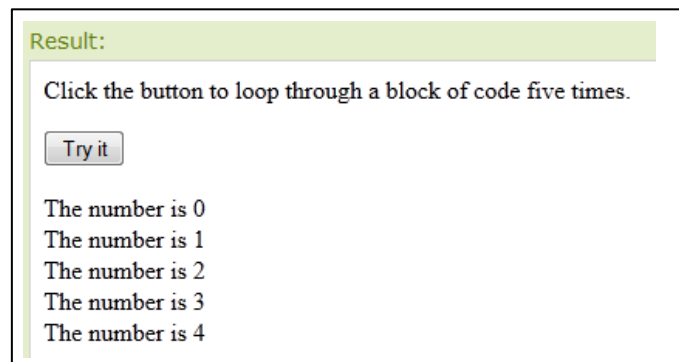
The first picture shows the initial button that the user must press to initiate the alert box.

The second picture will display the alert box with the phrase I mentioned earlier.

This example is for a program that uses a for loop. Here is what the code looks like:

```
<html>
<body>
<p>Click the button to loop through a block of code five times.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction()
{
var x="",i;
for (i=0;i<5;i++)
{
x=x + "The number is " + i + "<br>";
}
document.getElementById("demo").innerHTML=x;
}
</script>
</body>
```

A run for this program is as follows:



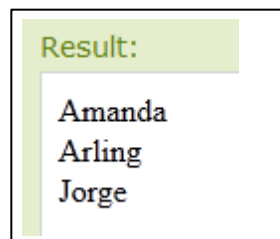
In this example, we are creating a button that the user clicks on to begin the program. The program begins with $i = 0$ and $x = x + i$. Thus, during the first iteration, $x = 0$, in the second, $x = 1$, in the third, $x = 2$, in the fourth, $x = 3$, in the fifth, $x = 4$. The for loop in JavaScript is similar to that of C++. In this specific example we have `for (i = 0; i < 5; i++)`; this means that i is

initialized to be 0 and that the loop will continue while $I < 5$ and that with each iteration, the value of i is going to be increased by 1.

For my final example of JavaScript, I will assign a value to a variable, change it, and display the new value for the variable. The code looks something like this:

```
<html>
<body>
<script>
var firstname;
firstname="Amanda";
document.write(firstname);
document.write("<br>");
firstname="Arling";
document.write(firstname);
document.write("<br>");
firstname="Jorge";
document.write(firstname);
</script>
</body>
</html>
```

The run for this code is:



Result:

Amanda
Arling
Jorge

This final example changes the value of a variable twice. Notice that `firstname` is initially given the value of `Amanda`, which is the first name that is printed. Then the value is changed to `Arling`, which is the second name that is printed, and finally it is changed to `Jorge`, which is the

final name that is printed. The `document.write` option is used to print the lines as in the first example in this section and the `
` tells the program to insert a line break.

These are just a few of the many things that can be done with JavaScript. Additional examples include creating clocks that display on a webpage and changing the colors of a webpage to either be one solid color or constantly changing.

In the last section of this report, I will evaluate JavaScript. Here, I will discuss each of the four criteria: readability, writability, reliability, and cost of JavaScript; I will provide both points that aide to each criteria, as well as things that affect them.

Evaluation

Readability

I think that JavaScript has a readability issue when it comes to the option of using semicolons. I think that by giving the writer the option of either using the semicolon or not causes a big problem because the choice affects the way the program is written. Remember that when a semicolon is used, multiple statements can be written on the same line, but if you are omitting the semicolons, each statement must be in its own line. I feel that this will harm the readability because it will make it more things to remember. Typically, good program writing says to include commas whether you are writing statements in one line or separate. I feel there should just be one way to do it because one might mix up the formats. Another part of the readability in JavaScript is that of the start and end points. I think that this aids greatly to the readability because it clearly lets you see where you start one thing and where it ends. Remember

in the semantics section I talked about `<html>`, `<body>`, and `<script>`. These are typically at the beginning of the program along with others when needed. When it is time to end a part of the program, say the script, you add `</script>`, when it is time to end the body, you include `</body>`, and finally when it is time to end the program, you include `</html>`. I think that this makes it really easy to find where one thing starts and ends.

The fact that JavaScript is case sensitive is a harm to the readability of the program. The reason for this is because case sensitivity allows for the same word to be used over and over again, as long as it has different spelling. For example, `sum`, `SUM`, `Sum`, and `sUM` are all the same word, but each one will have a different meaning in the program. This can become confusing when reading the program because it is hard to remember what each of them stands for. In addition, I feel that the use of comments is something really good. The fact that `/*` is used to indicate the beginning of a comment and `*/` is the end of the comment is really helpful in the reading of the program because it allows us to clarify statements that may not make perfect sense. The comments could be used to indicate at the beginning of the program what its purpose is, as well as explain various steps along the program so that one could follow the code with more ease.

Another aspect of JavaScript that I feel helps the readability is the rules that JavaScript has for variables. I like the fact that a variable name cannot be a reserved word, it cannot start with the numbers 0-9, which means it must begin with either a letter or an underscore. The part where a variable name cannot be a reserved word can be both an advantage and a disadvantage. First off, it is an advantage because it makes sure that a word does not have more than one meaning. However, it is a disadvantage, because I find it impossible to remember all of the

reserved words off the top of your head. However, the fact that variable names have to begin with either a letter or an underscore makes it easier to identify them within a program.

The final issue that I have with the readability of JavaScript involves arrays. Notice that in the syntax and semantics section I mentioned that in JavaScript, everything is considered an object. This allows you to include objects, functions, and arrays in arrays. I believe that this affects the readability because it will make it difficult to tell which object is an object, which is a function, and which is an array. I think that it is not a good idea to mix such things together because it could cause issues.

Writability

My first concern with the writability of JavaScript is the fact that it is a case sensitive language. I feel that this is of great harm because one must constantly be making sure that everything matches up. What this means is that the lowercase and uppercase letters must match up exactly for a variable that is being used more than once. This results in the need for the programmer to remember exactly what they use for the spelling of each variable. For the writability of the program, I think it is helpful that we use `<html>` and `</html>` to indicate the beginning and end of something. This is helpful because they are simple words that correspond with the area you are talking about and the `“/”` makes it easy to tell the difference. Finally, towards the writability of JavaScript, I feel that the fact that JavaScript is somewhat based off of the C language helps users and programmers already know some of the rules. Thus, when they are reading a program, they may already know what some of the statements stand for, which in return, makes it easier for them to write a program in JavaScript.

Reliability

I believe that JavaScript is fairly reliable, considering the fact that it runs on every single computer that is used today. It has come a long way and it has been through a bumpy road, but I feel that it does its job as expected and it is continuously becoming better. The main purpose for the JavaScript language is to make web pages more interactive with the user; well, if you go onto any web page nowadays, you will find that anything you do requires you to interact with the page. You click on a link and it sends you somewhere else, or you type in a question and receive an answer, etc... One of the issues with the reliability of JavaScript is that when you type check, it cannot distinguish between the different types of number. In addition, I believe that having a separate language for web pages makes JavaScript extra reliable. By doing so, you know that it is specifically designed for web pages and that it is going to work well in that setting; just as there are languages that are better for business or math. I feel that this is of harm, because there might be cases where you would only want to use integers. Another issue in the reliability of JavaScript is that an error handling is usually left out. What this means, is that the program will not continue when it reaches this error. I read that there is a way around this, which is by using a “try-catch” statement to solve this issue, but many JavaScript developers seems to omit this from their program. The truth is that a program is bound to reach an error at one point, so people should think ahead about it and include something like this, rather than believe that their program is perfect as is.

Cost

As far as cost goes for JavaScript, I would imagine that it should not be that much money since it is free for all users. Given the fact that it is practically universal, since it is embedded in the browser, I think that the hardware is fairly inexpensive. When it comes to training the programmers, I feel that JavaScript would be fairly easy to use. It is safe to say that everyone has worked with JavaScript before because practically everyone uses the Internet on a daily basis. It is not a complex language, which does make the cost for training decrease and in return makes it less topics for the programmers to get trained in. I tend to believe that overall JavaScript has a really good reliability. Everyone seems to use it for their webpages, and very rarely does it seem that web pages collapse on you. My one concern with this would be that tiny chance that you encounter an error and the program does not know how to handle it because it was not included with the “try-catch” function that I talked about earlier. But other than that, I would tend to believe that all users of JavaScript are happy with what they are getting, and so not many lawsuits are brought forward due to poor reliability.

Summary

I really enjoyed being able to do my own research on a language of my choice. I have done research on JavaScript for the past three months and I feel that I have learned some of the basics, but not all. I was interested in learning JavaScript, because I see it being used everyday when I use the internet. Since JavaScript is somewhat of a universal language, I was able to find a lot of information and tutorials on it. I did pick up on some code structures in these past month, but I do feel that JavaScript could be learned off the internet with a bit more time and

experimentation. I was able to find some websites where I could try out the language and I feel that I will look into that in the future.

Having learned C++, I feel I had the upper hand on reading programming languages. It was interesting to observe how the C based programming language is embedded with JavaScript and become informed of the history of JavaScript. Though I have only covered some of the basics and topics of JavaScript in this paper, I have become encouraged to research some of the more advanced topics in JavaScript and see what other things JavaScript can create. Throughout the process of writing this paper, I realized that JavaScript is a language that is always being altered because technology is getting better day by day and this affects languages as well. The change is easily seen through the presentation of web pages and how elegant and interactive they have become in the past years. Overall, I am happy that I made this language selection and I look forward to reading up on additional topics JavaScript has to offer.

Works Cited

- Bellis, Mary. "The History of JavaScript." *About.com Inventors*. N.p., n.d. Web. 08 Oct. 2013.
- Champeon, Steve. "JavaScript: How Did We Get Here? - O'Reilly Media." *O-Reilly.net*. N.p., 6 Apr. 2001. Web. 08 Oct. 2013.
- Chapman, Stephen. "A Brief History of Javascript." *About.com JavaScript*. About.com, n.d. Web. 04 Oct. 2013.
- Ducklin, Paul. "Java Is Not JavaScript - Tell Your Friends!" *Naked Security*. N.p., 16 Jan. 2013. Web. 05 Oct. 2013.
- Ihrig, Colin. "The History of JavaScript in a Nutshell." *Colin J Ihrigs Blog*. N.p., n.d. Web. 08 Oct. 2013.
- "JavaScript Fundamentals." N.p., n.d. Web. 19 Oct. 2013.
- "JavaScript History." *JavaScript History*. N.p., n.d. Web. 08 Oct. 2013.
- "JavaScript History & Information." *JavaScript History & Information*. N.p., n.d. Web. 08 Oct. 2013.
- "JavaScript RegExp Object." *JavaScript RegExp Object*. N.p., n.d. Web. 05 Dec. 2013.
- Kumar, Santosh. "History Of Javascript." *Java HTML Design*. N.p., n.d. Web. 05 Oct. 2013.
- Padolsey, James. "Looping in JavaScript." *James Padolsey RSS*. N.p., 3 Mar. 2009. Web. 21 Oct. 2013.