JavaScript notes

Tutorial 9, Carey 6e
There are references to the excerpt of book, *Weaving A Website*. It can be found in this same folder.

1. Introducing JavaScript

a. **How do you edit JavaScript code?** You edit JavaScript just like you edit HTML. That is, you can continue to use Komodo IDE/Edit, Notepad, or your favorite HTML editor. JavaScript gets embedded in the web page HMTL code. You can also place the code in a separate file[1] that you save with a .js extension. The html page is linked to the .js file via a <script> tag, similar to how it is linked to a .css file via the <link> tag. When coding directly into the web page, the JavaScript code is placed in between the script tags, inside the head or body area:

   Example:
   ```
   <head>
   ...
   <script>
   ...JavaScript code goes here ....
   </script>
   </head>
   ```

   Note that you can have more than one set of script tags in the page.

   How do you run JavaScript programs? Simply open the webpage in a browser! And to re-run the program script, click the refresh button. Running a JavaScript program does not require compiling as, say, C++ does. JavaScript is interpreted as opposed to compiled. And we will be coding and executing programs on the client-side, not the server-side. A script is just a program but smaller-sized one.

   What editor do you use? You can still use Komodo IDE, Komodo Edit, Notepad, or your favorite editor. (And you can use Chrome or other browsers for testing.)

2. Try running some sample JavaScript scripts.

a. Find a folder called **sample JS code (demo)** in the same folder as these notes. Open the webpage called **sample_js.html**. (Directions on how to do this are listed right in the web page.) In the first sample, you will be prompted to enter 2 numbers to be added. Run the code. That is, simply open the html page in a browser and see if you can match the code with the appropriate part of the program execution.

b. Investigate some free "cut-n-paste" JavaScript code available online. You can use this for your own web pages. The free code will usually come with a directions telling you where to place it in your HTML code, and what lines you modify to "customize" it for your needs. I have included some JavaScript that I cut and pasted in to my webpages. See two examples in the folders included with this same activities folder, folders called `cut-n-paste JS 1` and `cut-n-paste JS 2`.

   Check out for yourself some free JavaScript code at http://www.dynamicdrive.com that you can cut-n-paste into your webpages. Two other sites are https://www.cssscript.com and https://www.javascriptfreecode.com .

3. A description of JavaScript

a. JavaScript is an object-based scripting language. "When used in conjunction with the Document Object Model (DOM), it produces DHTML (dynamic HTML)." JavaScript is a "free-form" simpler language, making it more ideal for novices. With JavaScript, you can

[1] It can be a simple .txt file that you save with a .js extension instead. Similar to creating a CSS file.

create dynamic content in the web page with *events*. For example, when the user loads the page, this can be made to trigger a pop-up window to appear with a message. Or when he clicks a button, some sort of script is made to run. JavaScript can be used to validate forms on the client-side, thus saving on server-side traffic.

b. Some history of JavaScript. According to Anderson-Freed in her book *Weaving A Website (p174)*, JavaScript was originally meant to be a client-based language to interact with Java applets. The original creator of the language was Netscape. Netscape later joined with Sun MicroSystems (1995) to further its development and changed its name from LiveScript to JavaScript. JavaScript runs in most browsers. JavaScript is useful today, for example, in creating apps.

c. Note that Java and JavaScript are not the same language.

4. **Commenting your code**
   a. For HTML use <!-- --> Example: `<!-- Comments to HTML code go inside here. -->`
   b. For JavaScript use:
      1. /* */
         comments that use these delimiters can continue on multiple lines

         Example:
```
/* Calculate the hours left in the current day. Calculate
   the minutes and seconds left in the current hour */

   var hrsLeft = (daysLeft - Math.floor(daysLeft))*24;
   var minsLeft = (hrsLeft - Math.floor(hrsLeft))*60;
   var secsLeft = (minsLeft - Math.floor(minsLeft))*60;
```

      2. // for one-line, end-of-line comments to JS code.

         Example:
```
hrsLeft = (daysLeft - Math.floor(daysLeft))*24;  // calculate hours left
```

5. The **Assignment statement**
   a. Think of the variable, for example "x", as a name for a location in the computer's memory. You can then place a value, say 5, in that memory location with the following ***assignment statement***:
```
x = 5;
```
      Note the form:
      `x` is the variable
      `=` is the assignment symbol
      `5` is the value assigned to it
      `;` (semicolon) each JavaScript statement ends with a semicolon.

      A diagram of the memory location:

      $\boxed{}$

      The statement `x = 5;` results in:

      x

      $\boxed{5}$

      Further down in the program code: `x = 10;` results in:

      x

      $\boxed{10}$

      (10 overwrites the 5 in the memory location.)

Read the assignment statement Read `x = 10`; as follows,
"`10` is assigned to `x`" or
"x is assigned 10"
DON'T SAY, "`x` equals `10`."

And again, further down in the program code suppose you have: `y = 99;` This results in:

y

$\boxed{99}$

(So now you have 2 memory locations storing different values for later use.)

It is important to distinguish the **assignment** symbol "=" from an ordinary everyday meaning of "=". For JavaScript "=" is the assignment symbol and assigns a value to a memory location. As another example, Pascal (a programming language popular in the 70's) used the symbol ":=" for assignment. In mathematics, "=" stands for "is equal to", but in JavaScript "==" is the "is equal to" symbol. You will use "==" in JavaScript usually to test for the truth of an expression. For example, the expression `A == B`, is read, "A is equal to B", and has Boolean values of either `true` or `false`.

b. The **form** of an assignment statement:

```
Variable_name = value; or
Variable_name = expression;
```

Some other examples:

```
x = 5;
y = y + 1;    (This assumes that y was previously assigned a value, say, y = 4;)
solution = true;
LName = "Smith";
```

Above, the value true is a Boolean and the value "Smith" is a string.

6. **More about variables**
   a. **Conventions for naming JavaScript variables** (Variables are also called identifiers.)
      i. Variable names may consist of letters, numbers, _ (the underscore symbol) or the $ symbol.
      ii. Variable names <u>cannot</u> begin with a number. Variable names <u>cannot</u> contain spaces.
      iii. Variable names are case-sensitive. Example, **sum** and **Sum** are distinct names.
      iv. Meaningful variable names are desirable. (Longer names are better than short, abbreviated names in that respect.) Example: **totalBalance** or **total_balance** is better than **T_bal**.
      v. Some good variable names: **sum**, **S**, **totalCost**, **x**, **Height**, **newBalance**;
      vi. illegal variable names: **5_sum, total balance** (Note that "total balance" has a space in it.)

   b. Four (4) **data types**:
      i. **Number** - examples: `-3, 2.95, 79`
      ii. **String** - examples: `"hello", "the end", "The 'right' way.", 'The "right" way.'`
      iii. **Boolean** - can be just 2 values: `true, false`
      iv. **object** - there are 4 object types:
         1. browser objects
         2. document objects

3. built-in objects such as Math objects (session 9.3) and Date objects (session 9.2 and 9.3)
4. user-defined objects (more advanced)

c. **Declaring variables**: You will mimic most programming languages and *declare* the variables ***explicitly*** before using them. (Yet note that declaring variables explicitly is not a requirement in JavaScript.) Usually, you declare the variables in the first few lines of the program. In JavaScript, the declaration statement begins with the word `var`. Example:

```
var x;
```

You are declaring that you are going to use `x` in your program.

Or you could optionally at the same time assign `x` an initial value ("initialize the variable"):

```
var x = 5;
```

Yet alternately, you can *omit the declaration statement altogether*, and *declare* the variable ***implicitly*** by just using it (assigning it a value) when you need it. Example:

```
x = 5;
```

This is an implicit declaration. Again, JavaScript does not require explicit declaration of variables. In our textbook, the author likes to declare variables explicitly.

In JavaScript, when variables are explicitly declared, no data type needs to be specified, as is the case in many other languages, such as C++. In C++ one has to declare the variable and data type, as in the following C++ declaration:

```
int num = 7;  // int stands for integers
float x;      // float is Real numbers, including decimals
```

JavaScript is referred to as "weakly typed", in this respect.

d. **Declarations** can contain several variables on one line. Notice the placement of the comma and the semicolon.

```
var x, length;
var x = 5, length = 45;
x = 5, y = 10;  <-- implicit declaration
```

e. **Data type conversion** refers to the situation when one variable is first one data type but then later in the same program is a different data type. For example, you assign: `x = 5;` but later in same program you assign `x = "Mary";` . JavaScript can handle this. For most other languages though, this is taboo. Nonetheless you should avoid this. Again, JavaScript is a *weakly typed* language.

7. **Some Input and Output statements**

a. To **output** data, use the **alert** or **document.write** statements:

i. `alert("text goes here but no HTML");`[2] The alert statement displays the text inside the quotes in a pop-up alert window in the browser window.

[2] alert("text goes in here"); or window.alert("text goes in here");

ii. `document.write("text goes here and HTML allowed too");` The document.write statement displays the text (and renders the HTML) that appears inside the quotes.

b. To get **input** from the user, use the **`prompt`** statement**.** The **`prompt`** statement is not in Tutorial 9 but will appear in the teacher's sample code. The prompt statement has form: `prompt("text goes here");`. Once the user enters a value then you can use it or, say, store it in a variable using the assignment statement. For example:

```
x = prompt("Enter a number: " );
```

(The user will see `"Enter a number: "` on the screen. If he enters, say, `8`, then the result is that 8 is stored in memory in the variable x.)

8. **Concatenation (+)** (See Tutorial #9, Session 9.2, p698.) Double meaning (overloading) of the plus symbol "+". The "+" means to **add** but also it means to **concatenate**, or string together. The plus symbol "+" action depends upon the context. So, it will *add* if the operands are numbers but will *string together* if a string is involved. For example, the following 2 lines of code:

```
x = 5;
document.write( "x has the value of:  " +  x);
```

result in the following being displayed in the web page:

`x has the value of: 5`

And the statement `document.write(4 + 5);` displays as**:**

9

9. **Nested quotation marks** (Quotation marks inside of quotation marks) **-** By example, suppose you want to write to the page: *She said, "I need help".* You toggle the double to single, or single to double quotes and express it as follows:

```
document.write('She said "I need help." ' ); or
document.write("She said 'I need help.' ");
```

10. **More about alert(n) and document.write(n)** where **n** is the argument (input) of the method. Below is a summary of what you use as an argument to the alert or document.write methods. Check this code out by experimenting with it in an editor. (Remember to type the code in between `script` tags.)

Assume the following assignments to variables x and LName appearing in arguments below:
x = 8;
LName = Perel		

argument contains:	**alert(argument)**	**document.write(argument)**
text (in quotes)	`alert("Hello there");`	`document.write("Hello there");`
HTML (in quotes)	`no HTML allowed in alert`	`document.write ("<p>This is a paragraph.</p>");`
numbers	`alert(72);`	`document.write(72);`
variables	`alert(LName);` `alert(x);`	`document.write(x);`
math operations	`alert(3 + 4);alert(7*2-x);`	`document.write(7*2 + 1);`
A concatenation of numbers, variables, text	`alert(LName + 77 + "Hello");`	`document.write(LName + 77 + "Hello");`
A concatenation of numbers, variables, text	`alert(8 + 3 + LName);`	`document.write(8 + 3 + LName);`

11. **Tips on programing** and specifically, **object-based** programming
    a. Program code is executed line by line, top to bottom. If a loop or an if-then type of statement is encountered (or other such control structure), this order of execution is altered. (Loops and if-then statements are found in Tutorial 10.) For instance, if an if-then statement is encountered, a statement may be skipped right over. Here is a little explanation. The if statement in JavaScript has the form: `if (condition) command;` If the condition is true, then the command is executed. But if the condition is false, the command is skipped. Let's take an example. In the code:

    ```
    x = 5;
    if (x > 8) document.write("hello");
    ```

    The statement `document.write("hello");` would not be executed, since the condition `5 > 8` is false.

    b. **JavaScript is an object-based language**. One can see evidence of this in the use of the dot separator, the dot in the phrase, for example, **document.write(x).**

    Example1: in **document.write(n);** - the first part before the dot -- document -- is the **object** and the second part after the dot --write(n) -- in the general case, is either a **method** or a **property** of the object. In this case it is a method. write(n) is a method. I can read this from right to left as "perform the write method to the document object, using n as input".

    Example2 (arrays is in Tutorial 10): In **myArray.length,** length is the **property**. myArray, an array, is the **object**. I read this from right to left as "access the length property of the array myArray object".

    c. [Optional] **Reading from *Weaving a Website***. If interested in reading more about object-oriented programming, (OOP), read pp179-187, from an excerpt from the book *Weaving a Website* by Susan Anderson-Freed. If it in this same folder. She explains the object-oriented programming concept. She talks about event handlers on the rest of the pages p187-190. Event handlers are found in Tutorial 11 (Carey 6e).

12. **Actively practice JavaScript programming**
    It is important that you practice JavaScript coding and not just read about it. Try out the exercises from class, in the textbook and wherever you encounter them. Susan Anderson-Freed (*Weaving a Website*) suggests keeping a notebook for programming. Write down program snippets that you have tried and their results. Write down error messages and what they mean, and other things you learn. She suggests that you run the code making some mistakes on purpose to see what kinds of errors you get.

    For example, in trying to understand how a method (function) works, vary it and see what results. If you do not know what a statement does, type it up and run it to see. Keep in mind that if you want to evaluate "x = 3*8;", you will not see the result unless you also output it in an output statement such as "document.write(x);" or "alert(x);".

    Remember all you need is to type inside of the head or the body area of the HTML page:

    ```
    <script>
      some javascript code here
    </script>
    ```

    And add *some JavaScript code* in between. Save it and load the page in a browser to run your JavaScript program.