ATLS 4630/5630: Web Front-End Development
Week 2: JavaScript Basics

JavaScript

JavaScript is the official programming/scripting language of HTML5

- Goes beyond HTML and CSS
- Makes web pages dynamic and adds interactivity
- Runs entirely in the web browser
- Interacts with HTML and CSS
    - Access content of a page
    - Modify content of a page
    - Instructions the browser can follow
    - React to events in the browser

<script> </script> tag tells the browser all code within this tag is in a scripting language
All code within the script tags must be valid JavaScript.
To better organize your files it's best to separate your JavaScript into separate files

- Similar to using an external stylesheet for CSS
- JavaScript files should use a .js file extension
- The src attribute tells the browser where to find the JavaScript file, just like with images

<script src="js/script.js"></script>

https://repl.it/@aileenjp/JavaScript-basics

Variables

Naming variables

- JavaScript is case sensitive
    - FIRSTNAME and firstName are two different variables
- Don't use spaces or punctuation
    - Use capitalization or underscores for multiword names
    - Many of the punctuation characters have special meanings
- Use descriptive names
- Don't use really long names

JavaScript is a loosely typed language

- When defining variables you don't specify a type
- typeOf will return the type of the variable or object
    - does not throw an error if the variable has not been declared.

JavaScript uses dynamic typing to determine the type of the variable

- the type can change over time

Example:

```
var test = 42; //number
console.log(test);
test = "forty two"; //string
console.log(test);
```

Data types https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures

If a value is not provided it will have value of "undefined"

- "undefined" means the variable exists but its value is not defined
- Not the same as "null" which is an empty value that must be assigned

```
console.log(me); //error, not defined

Add var me; above the console.log and now it's undefined.
```

Strings

A string is a series of characters and are always enclosed in quotes
Can use single or double quotes, but the pair must match.
Some text editors will let you insert curly quotes around a string, "like this." JavaScript will not recognize strings surrounded by curly quotes; it only recognizes straight quotes, "like this."

Sometimes a variables' type in JavaScript is not what you expect

- Treats everything as a string unless you tell it otherwise
- It will convert data to a number if it thinks that makes more sense

typeOf will return the type of the variable or object
The isNaN function tells you if something is a number. Returns false if it's a number

- isNaN("four") returns true
- isNaN("4") returns false

JavaScript has functions that handle type conversion

- parseInt(x) converts the parameter to an Integer
- parseFloat(x) converts the parameter to a real number/floating point
- convert to a string using .toString()

Example:

```
//strings
var test2 = 30;
var sum = test + test2;
console.log(sum);
console.log(typeof(sum));
test = 42; //number
sum = test + test2;
console.log(typeof(sum));
```

Scope

Where and how a variable is defined will affect where it can be used. This called the variable's scope.
JavaScript has three ways of declaring variables:

- var - declares a variable with the scope of its current *execution context*
    - A variable that is defined inside a function using the var keyword will have a local scope.
        - Can only be used in that function
        - Function parameters are automatically local variables
        - Always declare variables by defining variables with the 'var' statement so that they have a local scope in a function.

    - Variables created outside a function are global variables and can be used anywhere in your script.
        - Convenient for accessing data throughout a program.
        - Dangerous because it's hard to keep track of where they're being changed.
        - Use global variables sparingly and intentionally.
- let - declares a block-scoped local variable that are limited in scope to the block, statement, or expression on which it is used.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let

    - This is unlike the `var` keyword, which defines a variable globally, or locally to an entire function regardless of block scope
- const - declares a block-scoped constant that can't be changed
    - Constants are block-scoped, like variables defined using let but cannot change through re-assignment, and it can't be redeclared.

A variable that is initialized inside a function without the var keyword will have a global scope and should be avoided.

Example:

```
//var scope
console.log("var");
var hot=90;
var temp=80;
    if (temp<hot){
      var hot=85;
      console.log(hot);
    }
console.log(hot);

//let scope
console.log("let");
let hot2=90;
var temp2=80;
    if (temp2<hot2){
      let hot2=85;
      console.log(hot2);
    }
console.log(hot2);

//constants
const freezing = 32;
console.log(freezing);
freezing = 0;
```

Conditionals

If statements allow us to respond differently based on a condition.
They use Boolean expressions to test a condition

- Every condition will evaluate to either **true** or **false**
- All computer decisions reduce to Boolean decisions
- In JavaScript Boolean decisions are formed using relational operators

Relational operators

= = Abstract Equality

- double equals (= =) will perform a type conversion when comparing two things

= = = Strict Equality

- triple equals (= = =) will do the same comparison as double equals but without type conversion; if the types differ, `false` is returned

(no spaces, I only have them for clarity)
JavaScript equality
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Equality_comparisons_and_sameness

!= Not equal to
< Less than
<= Less than or equal to
> Greater than
>= Greater than or equal to

- Relational operators can be combined with arithmetic operators
  - 5+3 < 4
- Relational operators are always performed last

Logical Operators

! NOT

- The NOT operator, **!**, is used to negate or invert a Boolean value.

|| OR

- Only one needs to be **true** to result in **true**!

&& AND

- Both must be **true** to result in **true**!

if/else statements allow the program to take one action if the given condition is **true** but a different action if the condition is **false**.

Example:

```
//conditionals
var temp3=25;
if (temp3 < 70)
{
  if (temp3 < 30)
  {
      console.log("It's " + temp3 + " degrees outside - " + "wear a
jacket!");
  } //end 30 if
```

```
    else
    {
      console.log("It's " + temp3 + " degrees outside - " + "wear a
sweater!");
    } //end 30 if else
} //end 70 if
  else
  {
      if (temp3 > 85)
    {
        console.log("It's " + temp3 + " degrees outside - " + "wear a
tank top!");
      } //end 85 if
    else
    {
        console.log("It's " + temp3 + " degrees outside - " + "wear
short sleeves");
      } // end 85 if else
  } // end 70 if else
```

Run with values 25, 50, 86, 85

Using Conditionals:

- Plan out what you want your program to do.
- Put your test condition in ( )
- Type both { } for your conditional statements so you don't forget the closing bracket
- Indent code in brackets so it's easier to read
- Remember = = = for testing equality
- No semicolon after the test condition because it's not the end of the if statement

Iteration

JavaScript supports the most common types of loops so our programs can repeat a set of instructions until a given condition changes.

<u>While Loop</u>

While loops repeat a set of instructions

- While the test condition is true the statements in the curly brackets are run
- When it reaches the end curly bracket, JavaScript jumps back to the beginning of the while loop and evaluates the test condition again.
- When the test condition evaluates to false the body of the while loop is skipped and it continues with the rest of the program.
- We must provide something to test before the loop is executed the first time.
- The loop body has a chance of never executing if the test condition is **false** the first time.
- The while loop is excellent for error checking

JavaScript also supports the do-while loop which puts the test condition at the end so the body always executes at least once.

Example:

```
//iteration
var guess;
var result;
var target;
var turns = 0;
target = Math.floor(Math.random() * 100) + 1;
alert("I'm thinking of a number between 1 and 100. Guess my number,
and I'll tell you if it's too high, too low, or correct.  Try to
guess in the fewest turns.");
guess = prompt("What is your guess?");
turns = turns + 1;
while (guess != target)
{
  if (guess > target)
    {
      guess = prompt ("Too high!! Next guess?");
    turns = turns + 1;
    } // end if
    if (guess < target)
    {
      guess = prompt ("Too low!! Next guess?");
    turns = turns + 1;
    } // end if
    if (guess == target)
    {
      console.log("YOU WIN!!! The number was " + target + ". It took
you " + turns + " turns! ");
    } // end if
} // end while
if (turns<= 10)
{
  if (turns <= 5)
  {
    result = "Nicely done!";
  }
  else
    {
      result = "Not bad";
```

```
    }
}
else
  {
    result = "That took a lot of guesses";
  }
console.log(result);
```

JavaScript Math reference https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math

For Loop

The for loop is usually used for repeating a set of instructions a certain number of times.

- the number of repetitions can be calculated

There are three parts within the for loop:

1. Initialize the counter variable
    - Only happens the first time
2. Test condition
    - Boolean expression
3. Update the counter
    - increment/decrement statement
    - Ensures the test condition eventually is false

Initialize counter to 0 or 1

- Will change number of repetitions

If the test condition is false, the loop will never execute
Test condition must be true for the loop to execute
Don't change the counter within the loop
The counter will automatically be changed at the beginning of each loop iteration

Infinite loops

- Iteration will continue as long as the test condition is **true**.
- Something within the loop body ***MUST*** change causing the test condition to become **false**.
- If a **false** condition is never reached, you will have an infinite loop.
- Carefully plan out your program before writing code.
- Initialize your test condition variable(s) before a while loop.
- Make sure the test condition will eventually become false and the loop will end.
- Handle incorrect user input.
- Desk check your program.

Arrays

An array is a data structure that holds a group of values that are related to each other

- An array is an indexed collection of data
- The ***index*** is the reference number used to access the stored data
    - The index is a location, similar to a PO box number.
    - The index always starts at 0
- The ***element*** is the data stored at a given index location, similar to the mail in the PO box.

    - The first element is located at index [0]
    - The last element is located at index [length-1]
- Arrays are ordered
- Array can hold duplicate data
- The square brackets tell JavaScript to create an array
- Create an empty array when the data will be assigned later
- for loops are a great way to iterate through an array because you can calculate how many items are in the array
- The .length property stores the number of items in the array
- The forEach() method executes a function once for each array element https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach

Example:

```
//arrays and for loop
var scores = [];
var num;
num = prompt("How many grades do you have?");
for (i=0; i<num; i++)
{
  scores[i]=prompt("Enter grade #" + parseInt(i+1));
}

console.log("Your grades are:");
var total=0;
scores.forEach(function(score, index, myscores){
  console.log(score);
  total=total+score;
});
console.log("Your average is " + total/num);
```

What type do you think scores is?

```
console.log(typeof(scores));
```

How about the items in scores?

```
console.log(typeof(scores[0]));
```

We want the items in scores to be treated as integers.

```
total=total+parseInt(score);
```

Other array functions
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

- .push() adds an item to the end of the array
- .pop() removes an item from the end of the array
- .shift()removes an item from the front of the array
- .unshift() adds an item to the front of the array

JavaScript also has Sets which are unordered keyed collections of data

- Keyed collections are collections which use keys; these contain elements which are iterable in the order of insertion.
- Cannot have duplicate data

Functions

A function consists of multiple instructions that are grouped together so they can be run together because they perform a specific task

- Enables modularization
- Makes code reusable
- Reduce size and complexity of code
    - Efficiency by avoiding repetition
- Ensures consistency
    - Modifications are only made once
- Functions may:
    - Perform a task
    - Accept data
    - Return a value
    - Or any combination of the three

You can define functions in the <head> section or separate your JavaScript functions into their own file so they can be used in multiple pages.

- JavaScript files should use a .js file extension
- To run a function you call it within the **<body> </body>** portion of an HTML page.
- Functions are not run until they are called in the body portion of an HTML page.

Parameters

Some functions need data to achieve its task.

Ex: The alert() and prompt() functions need data sent to them.

- Parameters in the function header act as variables to hold data sent to the function
- If a function is expecting data you must send the data it needs when you call the function
    - Arguments represent the data sent to a function when it's called
- Arguments and parameters are matched 1-to-1 according to their order

Return values

A function can return data back to where the function is being called from.

Ex: The prompt() function returns what the user entered.

- Return statements return a single value
- Functions without a return statement have a specific purpose that does not require a value to be sent back to the calling program
    - Output
- When calling a function that returns a value you must do something with the returned value
    - assign it to a variable
    - have it be a part of output

Example:

```
//functions
```

```
function FahrToCelsius(tempFahrenheit)
{
  var tempCelsius;
  tempCelsius = (5/9) * (tempFahrenheit - 32);
  return tempCelsius;
}

var tempF;
var tempC;
tempF = prompt("Enter the temperature (in Fahrenheit):");
tempC = FahrToCelsius(tempF);
console.log("You entered " + tempF + " degrees Fahrenheit.");
console.log("That's equivalent to " + tempC + " degrees Celsius.");
```

Comments

As your scripts get larger it's easy to lose track of what all your variables are for, and what different parts of the code do.

When you, or another programmer, returns to your code weeks or months later, comments help you remember what each part of the program does.

Comments are notes only visible in the code, not seen in the browser by the user

Single-line comments

- // this is a single line comment
    - var firstName; // users first name
- The browser ignores everything after the //

Multi-line comments span more than one line

- /* This is a multi-line
- comment */

Good places to use comments

- Variables - what they are for
- Different parts of your script do
- Each individual algorithm
- Anything you want help remembering
- Document the source of your code

Any JavaScript commands in the comments will be ignored

- Also useful to temporarily "comment out" parts of your code that you don't want to run