

Web Scripting 1

Day 02

Agenda

- Assignment 1 Walkthrough
- Progress check-in
- Data types
- Functions
- Assignment 02

Assignment 01

Walkthrough

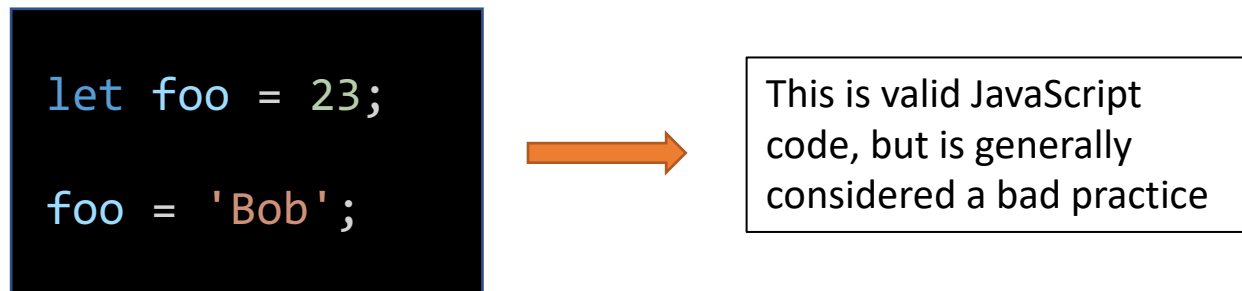
Progress Check-In

- At this time you should have a rough idea of the following concepts
 - What is JavaScript
 - How to attach a JavaScript file to an HTML file
 - What is a variable
 - Window Methods
 - Alert
 - Prompt
 - Confirm

Data Types

Data Types and Variables in JavaScript

- JavaScript is dynamically typed¹, which means developers do not need to declare a data type when creating variables
- JavaScript will automatically set the data type of a variable based on the value stored in a variable
- Variables in JavaScript can change their data type dynamically. A variable can start out as a string and change to a number. Though this is often considered a bad practice, it is still valid code and will not cause an error



1. <https://en.wikipedia.org/wiki/JavaScript>

Variable Data Types

- Data stored in variables are often associated with a type of data
 - JavaScript has three main data types (plus a couple of others)
 - Strings
 - A series of character strung together (usually words wrapped in quotes ("", ", or `"))
 - 'Hello World' -> string data type
 - Numbers
 - In JavaScript all numbers are numbers. JavaScript does not differentiate between integers (whole numbers) and floating point numbers (numbers with decimals)
 - 12 -> number data type
 - 1.5789 -> number data type
 - Booleans
 - Can have two values:
 - true -> boolean data type
 - false -> boolean data type

String Data Type

- A string is a series of characters "strung" together.
- In JavaScript, strings are always wrapped in quotes (" , ' , `)
- There is no difference between using single quotes and double quotes
 - The string 'Hello World' and "Hello World" are functionally similar in JavaScript
- In the upcoming slides we will discuss template strings (` `)

```
const user = 'Matthew';
```



The variable "user" is a string data type

Escaping Special Characters

- If we have a string that contains a special character that would otherwise confuse the JavaScript parser we can use the "\" to escape that special character

```
const someText = 'The Terminator said "I'll be back!";
```

← BAD

We can use the "\" character to escape the "'" character that was causing the error above

← The "'" will confuse the parser and cause an error. Notice the strange colour of the characters in the string. This is one way your text editor tries to help you with this error

```
const someText = 'The Terminator said "I\'ll be back!";
```

← GOOD

String Concatenation


- To concatenate a string means to combine 2 or more strings together to form a single string
- In JavaScript, one method to combine a string is to use the " + " character
- **JavaScript does not know grammar**
 - In example 1, JavaScript will combine the "firstname" and "lastname" variables without any spaces between the strings
 - You need to add an " " (empty space) in between the strings to create a properly formatted full name string with spaces between the names. This is what we did in example 2

Example 1 (No space between words)

```
const firstname = 'John';  
const lastname = 'Smith';  
const fullname = firstName + lastName;  
// fullname -> JohnSmith
```

Example 2 (with a space character (' ') manually added between the "firstname" and "lastname" variables)

```
const firstname = 'John';  
const lastname = 'Smith';  
const fullname = firstName + ' ' + lastName;  
// fullname -> John Smith
```



An empty space character was manually inserted between the two strings

Template Strings

- Template strings are strings wrapped using the back tick character (`)
 - The back tick character is usually located beside the "1" character on a US English keyboard


The back tick character



Template Strings - Features

- Template strings allow you to insert variables directly inside a string using the "\${yourVariable}" syntax

```
const firstname = 'John';  
const lastname = 'Smith';  
const fullname = `The user's name is ${firstname} ${lastname}.`;   
// fullname -> ...is John Smith.
```



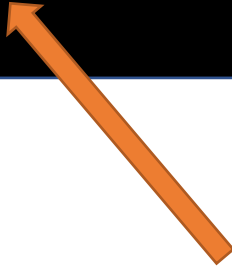
Variables inserted directly
inside a template string

White space is honored
inside template strings

Template Strings - Features

- Template strings allow you to run code directly inside the string

```
const subTotal = 23.57;  
const taxRate = 0.05;  
const output = `Your total including tax is ${subTotal + (subTotal * taxRate).toFixed(2)}`;  
// output -> Your total including tax is 24.75
```



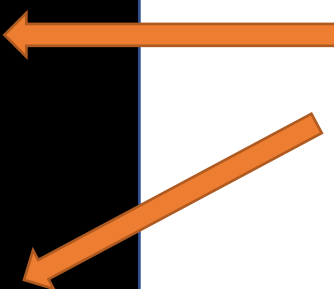
Running JavaScript code
directly inside a template
string

Template Strings - Features

- White space is honored inside template strings
- This is handy if you want to insert dynamically created HTML into the DOM and wish to maintain proper indentation

```
const dynamicHTML =  
  `

  
    <li>Foo</li>  
    <li>Bar</li>  
  </ul>`;  
/* dynamicHTML ->  
  <ul>  
    <li>Foo</li>  
    <li>Bar</li>  
  </ul>  
*/
```



All your indentation and spacing is preserved when using template strings

Number Data Type

- In JavaScript all numbers (integers and floating point decimal numbers) are assigned the number data type
 - 27 -> number data type
 - 32.7823 -> number data type
- Most of the usual operations you may be familiar with from math or other programming languages are available in JavaScript
 - + (add)
 - - (subtract)
 - * (multiplication)
 - / (divide)
 - % (modulus)

What is the Modulus Operator (%)?

- The modulus operator returns the remainder of a division between two numbers
 - $17 \% 5 \rightarrow$ returns 2 (17 divided by 5 equals 3 with a remainder of 2)
- If the dividend (the first number) is smaller than the divisor (the second number) then the modulus returns the dividend
 - $5 \% 17 \rightarrow$ returns 5
 - $3 \% 5 \rightarrow$ returns 3
 - $2 \% 9 \rightarrow$ returns 2

The Order of Operations

- JavaScript follows the order of operations
- In Canada we learn the acronym BEDMAS
 - B -> Brackets -> $(5 + 2)$
 - E -> Exponents -> eg: 5^2 (5 x 5 or 5 squared)
 - D -> Division -> $10 \div 5$
 - M -> Multiplication -> 10×5
 - A -> Add -> $5 + 3$
 - S -> Subtract -> $7 - 2$
- JavaScript follows BEDMAS
- Make sure you write your equations correctly to avoid unpredictable results

How you write your equation will determine the result. Be sure to follow proper order of operations to get the intended result



```
const total = 5 + 5 * 3;  
// total -> 20
```

or...

```
const total = (5 + 5) * 3;  
// total -> 30
```

Beware of String Numbers in JavaScript

- A string number is a string with a number as its value
 - '23' -> since this "number" is wrapped in quotes it is considered to have a string data type
- With JavaScript that runs in a browser we often get data from the HTML document from form fields or prompt boxes
 - Almost all data your script receives from the HTML document is a string, even if it is a number, it is always a string data type
 - This even applies to form fields with an input type of "number". The data received from number inputs are still strings

Beware of String Numbers in JavaScript

- JavaScript will not always automatically convert string numbers to numbers
- Be especially cautious when adding two string numbers together in JavaScript as you will get unpredictable results

```
const formInput1 = '23';  
const formInput2 = '7';  
const total = formInput1 + formInput2;  
// total -> 237
```



Since the variables "formInput1" and "formInput2" are strings, JavaScript will simply concatenate the strings together and assign the "total" variable a value of "237" (literally "23"+"7" -> 237). We will look at solutions to this problem in the next few of slides

String to Number Conversion

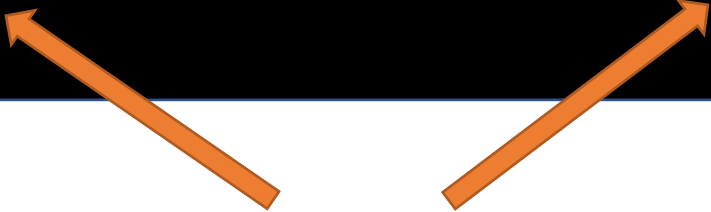
- Dealing with string numbers is a common problem in JavaScript that runs in the browser
- JavaScript provides several ways to convert a string number to a number data type
- On the next slides we will look at three options

parseFloat() and parseInt()

- parseFloat() will convert a floating point (decimal number) and an integer string number to a number
- parseInt() will convert an integer number (whole number) into a number
 - If you pass in a floating point number into parseInt() then JavaScript will simply remove the decimals and return an integer
 - Use parseInt() if you are sure you will be dealing with a whole number

parseFloat()

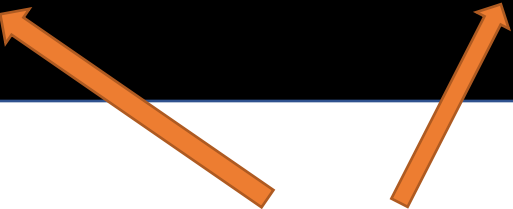
```
const strNum1 = '23.571';  
const strNum2 = '7.258';  
const total = parseFloat(strNum1) + parseFloat(strNum2);  
// total -> 30.829
```



We convert our string numbers to numbers using "parseFloat()" which then allows us to do a proper "addition" operation and get the proper result. Since our numbers have decimals we use the "parseFloat()" method. The parseFloat() method also works with integers, but if you are 100% positive you will be dealing with whole numbers than use parseInt() instead

parseInt()

```
const strNum1 = '23';  
const strNum2 = '7';  
const total = parseInt(strNum1) + parseInt(strNum2);  
// total -> 30
```



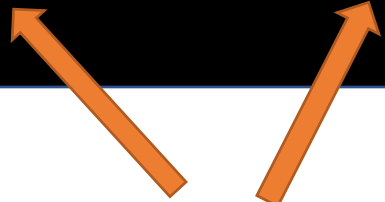
We convert our string numbers to numbers using "parseInt()" which then allows us to do a proper "addition" operation and get the proper result. Since our numbers are whole numbers and do not contain decimals we use the "parseInt()" method

Type Conversion with Operators

- JavaScript will convert your string numbers to numbers if you try to multiply them or do a few other mathematical operations on the string number
- We can use this to our advantage to convert our string numbers to numbers

Type Conversion by Multiplying by 1

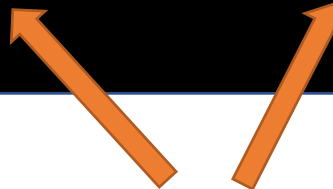
```
const strNum1 = '23';  
const strNum2 = '7';  
const total = strNum1*1 + strNum2*1;  
// total -> 30
```



Multiplying our string numbers by one will force JavaScript to convert our string numbers to numbers which will then allows us to perform our addition operation on our variables and get the correct result

Type Conversion by Prepending "+"

```
const strNum1 = '23';  
const strNum2 = '7';  
const total = +strNum1 + +strNum2;  
// total -> 30
```



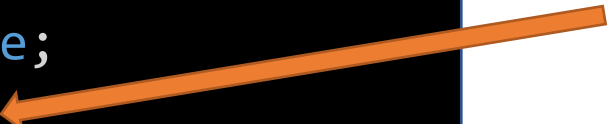
Placing a "+" character before our numbers will force JavaScript to convert our string numbers to numbers which will then allows us to perform our addition operation on our variables and get the correct result

Boolean Data Type

- Computers at their core are just a series of electronic switches that either let electricity flow (on or true) or prevent the flow of electricity (off or false)
- Most computer languages have a Boolean data type that can store one of two values
 - true (on)
 - false (off)
- Boolean values are never written with quotes
- We often use Booleans with conditional logic in our scripts
 - We will explore conditional statements in a future lesson

Boolean Data Type Example

```
let userCanVote = true;  
  
if(userCanVote == true){  
    console.log('User can vote');  
}else {  
    console.log('User can\'t vote');  
}
```



Booleans combined with a conditional statement provide an easy way to direct your script in different directions. In this example, since the variable "userCanVote" has a Boolean value of true the script will output "User can vote" in the console.

Functions

Functions

- A function is a collection of statements that you can call once or many times throughout your script
- Functions can take parameters that affect their output
 - Parameters are declared between the "()" of a function

Defining the function

```
// Function definition
function sayName(user){
    alert(`Hello ${user}`);
}
```



Calling the function and changing its output by passing in values

```
// Calling the function
// with different values
// for the user parameter
sayName('John'); // Alerts -> Hello John!
sayName('Melanie'); // Alerts -> Hello Melanie!
sayName('Sarah'); // Alerts -> Hello Sarah!
```

JavaScript Functions

- Today's class will be a brief introduction to JavaScript functions
- We will build on our introductory knowledge of functions in future classes

JavaScript Function Definitions

- JavaScript functions can be defined in multiple ways
 - Two main ways to define functions in JavaScript are:
 - Function expressions
 - Function declarations
 - Both of the above ways of declaring functions are functionally similar with one important difference, which is explained in an upcoming slide

```
// Function expression
const cube = function(num){
    return num * num * num;
}
```

```
// Function declaration
function cube(num){
    return num * num * num;
}
```


Function Expressions vs Declarations

- One is not really better than the other
- One major difference between function expressions and declarations is hoisting
 - JavaScript will hoist function declarations to the top of the script or the top of the parent function
 - This means that function declarations can be available anywhere in the current scope of the script regardless of the location of the function in the scope
 - Function expressions are like regular variables, in that they must be defined first before calling them

Function Declarations are Hoisted

```
// Since function declarations  
// are hoisted they are available  
// in our scripts before we define them  
cube(3); // 27  
cube(5) // 125  
cube(7) // 343
```

This is valid because function declarations are hoisted by JavaScript and are therefore available to be called before they are defined in our script

```
// Function declarations are hoisted  
// to the top of the script by  
// JavaScript  
function cube(num){  
    return num * num * num;  
}
```

This function declaration is hoisted to the top of our script by JavaScript

Function Expressions Must Be Declared First

*** Note ***: This is bad code!!!

```
// Since function expressions  
// are NOT hoisted the following  
// code will cause an error  
cube(3); // 27
```

```
// Function expressions  
// not hoisted and must be  
// defined before calling  
// them  
const cube = function(num){  
    return num * num * num;  
}
```

This will cause an error because we are calling the "cube()" function before defining our function expression

This function expression is **NOT** hoisted

Function Expressions Must Be Declared First

```
// Define you function expressions before calling them
const cube = function(num){
    return num * num * num;
}

// This code is valid since we defined our function
// expression before calling it here
cube(3); // 27
```

Personal Opinion Here

- I prefer to write my functions near the bottom of my scripts, so I prefer to use the function declaration syntax
- Using function declaration syntax means my functions are available anywhere in the current scope of the script
- This is a personal choice

```
// Calling the cube() function
cube(3); // 27

// I prefer the function declaration syntax
function cube(num){
    return num * num * num;
}
```

Using Return in a function

- One way to get processed data out of a function is through a "return" statement
- Once JavaScript hits a "return" statement in the function, the function stops and any code located after the return statement will not run
- You can return any type of data or no data at all

Using return in a function

```
const colours = ['red', 'green', 'blue'];
// The htmlList variable will be set to whatever data
// the convertArrayToHTMLList() function returns
const htmlList = convertArrayToHTMLList(colours, 'ul');
// htmlList ->
// <ul>
//   <li>red</li>
//   <li>green</li>
//   <li>blue</li>
// </ul>

function convertArrayToHTMLList(arr, listType){
  let html = `<${listType}>`;
  arr.forEach(function(item){
    html += `<li>${item}</li>`;
  });
  html += `</${listType}>`;
  // This will return the html
  // variable
  return html;
}
```

Using Multiple Returns in a function

- You can have more than one return statement in a function
- JavaScript will stop the function when it hits the first return statement
- We can leverage this behaviour by writing code that has a fail-fast feature that will stop a function “if” a condition is or is not met
 - This can make our code easier to read and prevent our code from becoming unwieldy due to nested “if” statements

```
function sayHello(username){  
  
    // Fail-Fast  
    //  
    // If a name was not passed into the  
    // function, we end the function  
    // with a return statement  
    if(username === undefined){  
        return;  
    }  
  
    // If the code passes the fail-fast  
    // code block, then this code  
    // will run  
    return `Hello ${username}!`;   
  
}
```