COMP 1950

Day 11

Intro to JavaScript

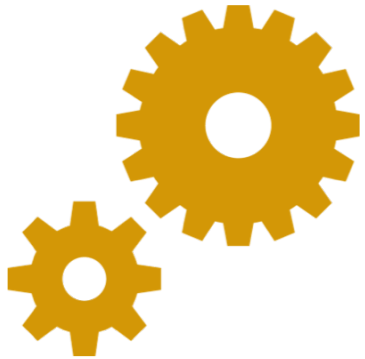
What is Front-End Web Scripting

- A web site can be divided into 3 main areas
  1. Content (HTML)
  2. Style (CSS)
  3. Behaviour (JavaScript)
- Web scripting deals with the behaviours of our web page

Content =
HTML

Style =
CSS

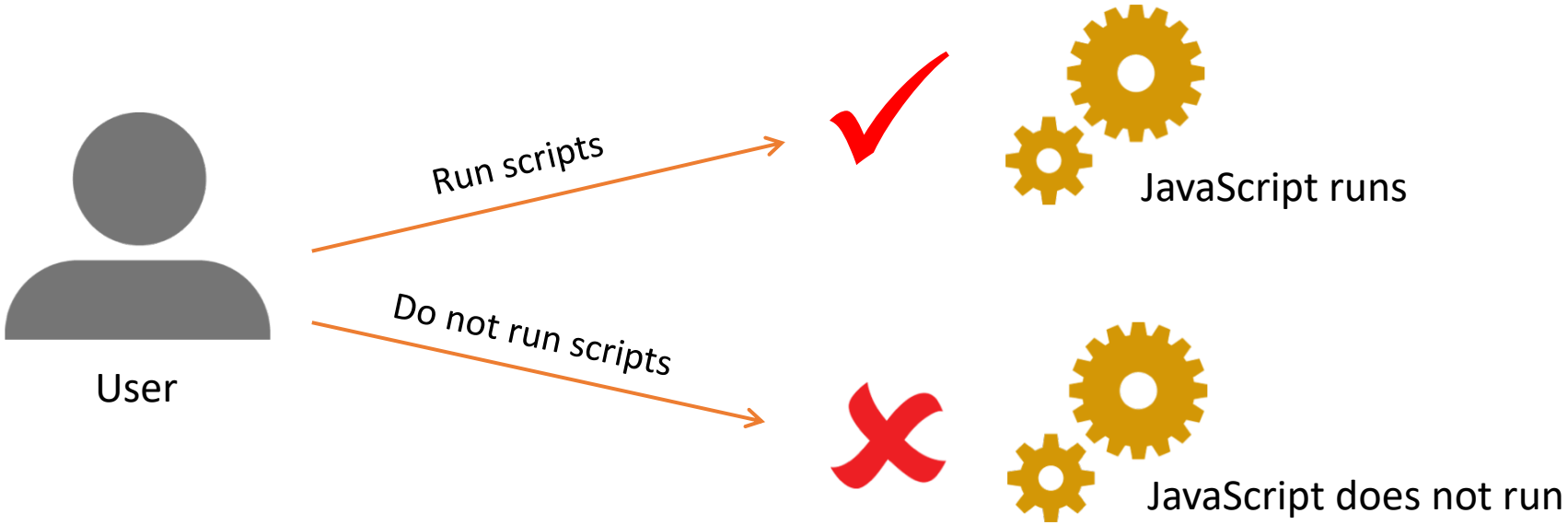
Behaviour =
JavaScript



What is Web Scripting

- Web scripting can mean either client side scripting (front-end scripting) or server side scripting (back-end scripting)
- Web scripting for this class deals with client side scripting or front-end scripting
  - Client side scripting means the code is executed on the end-users computer
    - Desktop computer
    - Laptop computer
    - Mobile phone
    - Tablet
  - With client side scripting the end-user usually has a choice whether or not to run your scripts or not

Users Can Turn off Front End Scripts



What is JavaScript

- JavaScript is not Java
- JavaScript is an Object-Oriented scripting language
- JavaScript is one implementation of a specification called ECMAScript
- JavaScript is the only client side scripting language that is broadly supported by all the major web browsers
- It is primarily used in a web browser to create enhanced user interfaces and add interactivity to web sites

History of JavaScript

- Developed by Brendan Eich at Netscape around 1995 – 1996
- Originally called LiveScript
- Name was changed to JavaScript to leverage the popularity of the Java programming language which is mostly unrelated to JavaScript
- The use of the word “Java” in the name for JavaScript had some negative implications
  - At the time “Java” not JavaScript was used to add enhancements to web sites and “Java” required a user to install a plug-in in order for these enhancements to run
  - Users confused Java with JavaScript
- JavaScript ran in a browser natively without requiring a plug-in

History of JavaScript

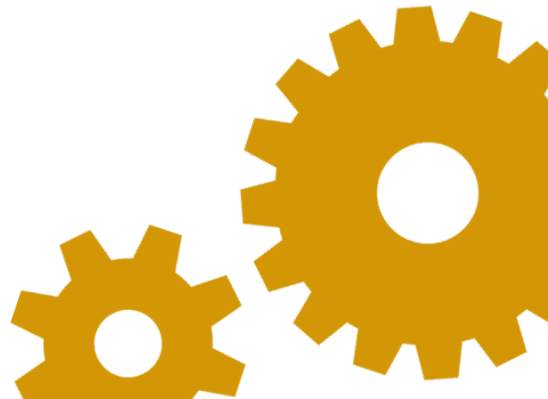
- In 1996 Microsoft released IE 3.0 in 1996 which included Jscript
  - JScript was Microsoft's version of JavaScript
- JScript and JavaScript were similar but not entirely the same
- Microsoft and Netscape and the European Computer Manufacturers Asscociation (ECMA) created a language specification called ECMAScript
- Currently all modern browsers support some version of the ECMAScript Standard

The Limits of JavaScript

- A user can not be forced to use JavaScript
- Data from a user can not be relied upon
- JavaScript code on one web site normally can not interact with or access JavaScript code on another web site
- JavaScript that runs in a browser can not access the user's file system

Development Environments

- JavaScript usually runs in a web browser
- Like HTML or CSS a simple text editor is all you need to create a JavaScript script (Don't use a Word Processor)
- External JavaScript files should be saved with the extension “.js”
- JavaScript can be tested locally (with some issues with Internet Explorer) or tested either on a local testing server or a remote web server
- There is no compile step for running native JavaScript in the browser. The plain text JavaScript file is parsed by the browser

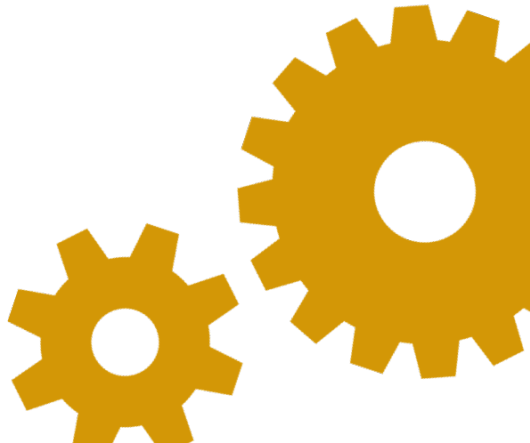


Common Uses of JavaScript

- Display a message box
- Select list navigation
- Edit and validate form information
- Image Rollovers
- Status Messages
- Display Current Date
- Calculations

Adding JavaScript to a Web Site

- JavaScript can be added to a web page in three ways
  - Inline inside HTML tags
  - Using embedded scripts inside the HTML file
  - Linking to external JavaScript files (preferred for most situations)



Inline JavaScript

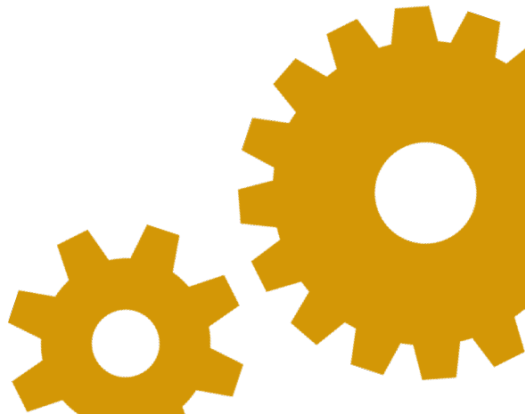
- Adding JavaScript inside an HTML tag
    - This is the least preferred method
    - Sometimes we have to use this method, but if possible try to use either embedded scripts or an external HTML file

```
<h1 onclick="helloWorld()">Click Me</h2>
```

This runs a custom JavaScript function called "helloWorld" when the user clicks on the `<h1>` tag. We will discuss more about JavaScript functions later on

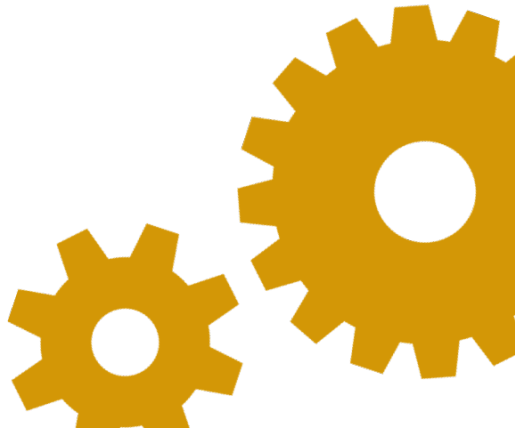
Embedding JavaScript

- Embedding JavaScript scripts inside an HTML file
  - This is similar to embedding CSS styles into an HTML file
  - You use the `<script></script>` tag and place your JavaScript scripts inside the `<script>` tags
  - Embedded scripts can be placed anywhere inside the `<head>` section or the `<body>` section of an HTML document
  - Usually the scripts are included either at the bottom of the `<head>` section or at the bottom of the `<body>` section
  - If you place your scripts before the closing body tag then you gain the benefit of not having your scripts block the HTML rendering. Also, your HTML loads first, giving your JavaScript script full access to the HTML



Adding JavaScript to a Web Site

- The current best practice is to include your JavaScript scripts at the end of the `<body>` element, right before the closing `</body>` tag



Adding JavaScript to a Web Site

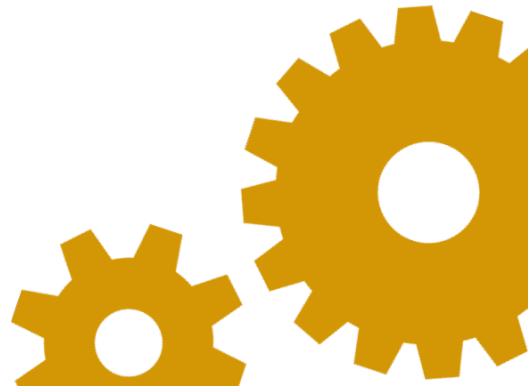
- Syntax for embedding JavaScript scripts inside an HTML5 page

```
<script>
      /* JavaScript Code goes inside the script tags */
      alert(“Hello World”);
</script>
```

Using An External Script File

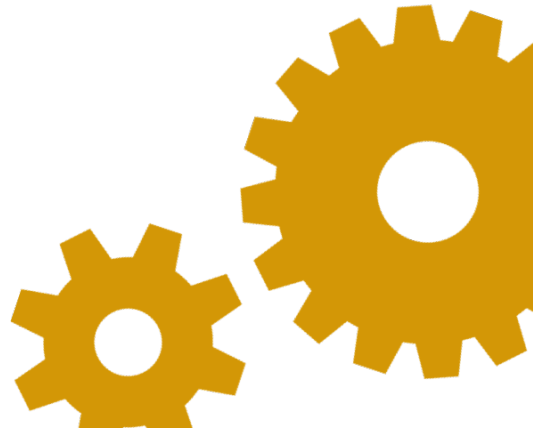
- Syntax for adding an external JavaScript file to an HTML5 page
- Using an external JavaScript file is the preferred method in most cases

```
<script src="some-script.js"></script>
```



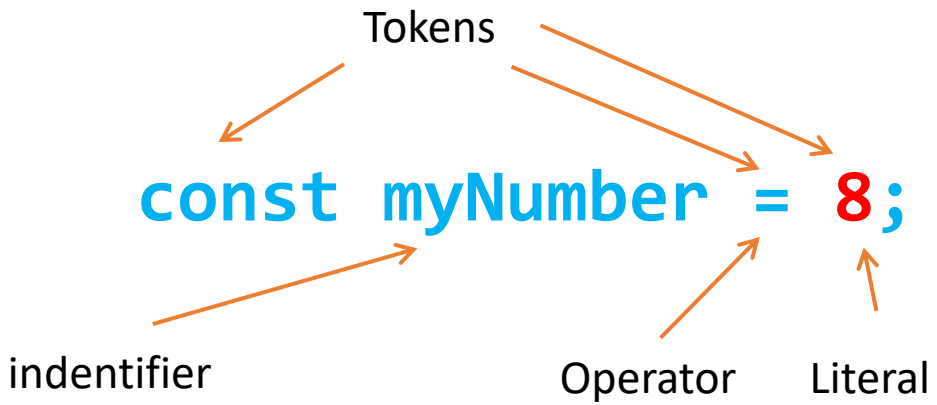
JavaScript Syntax

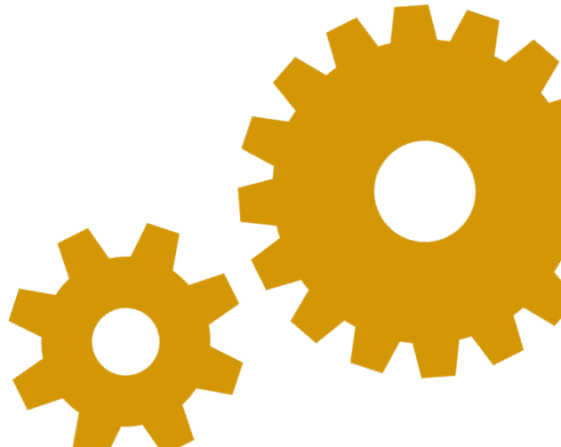
- JavaScript is case sensitive
- JavaScript ignores extra whitespace within statements
- Each JavaScript statement ends with a semicolon
- JavaScript uses dot (.) notation to access properties and methods of an object (more on this later)



JavaScript Syntax

- A JavaScript script consists of statements which can consist of the following parts:
  - Tokens
  - Operators
  - Identifiers
  - Literals
- A JavaScript statement:



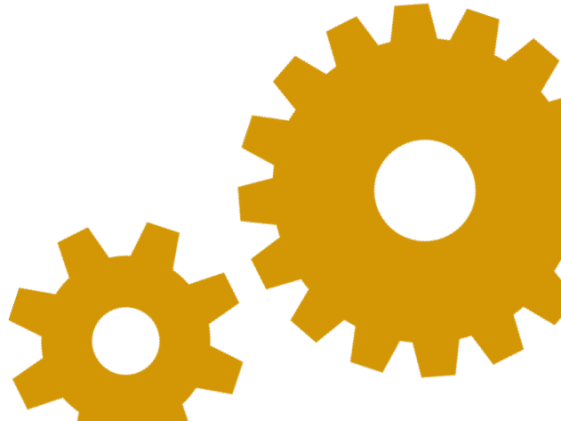


JavaScript Syntax

- Identifiers are labels you give to variables, functions, arrays and objects. We will discuss all of these later
  - Rules for creating identifiers
    - Identifiers can only contain letters, numbers, the underscore, and the dollar sign.
    - Identifiers can't start with a number
    - Identifiers are case-sensitive
    - Identifiers can be any length
    - Identifiers can't be the same as reserved words (word reserved by JavaScript)
    - Avoid using global properties and methods as identifiers

```
const myNumber = 8;
```

indentifier

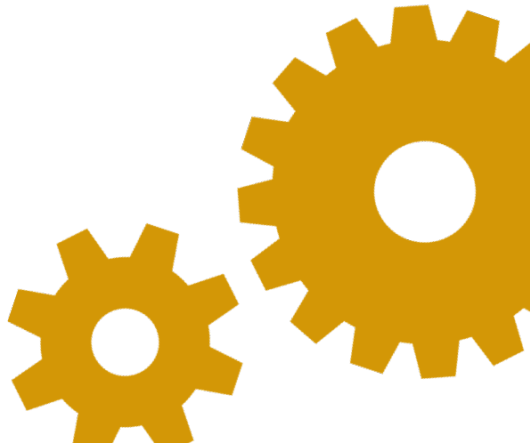


JavaScript Syntax

- Valid identifiers in JavaScript
  - subtotal
  - index_1
  - $
  - taxRate
  - calculate_click
  - $log

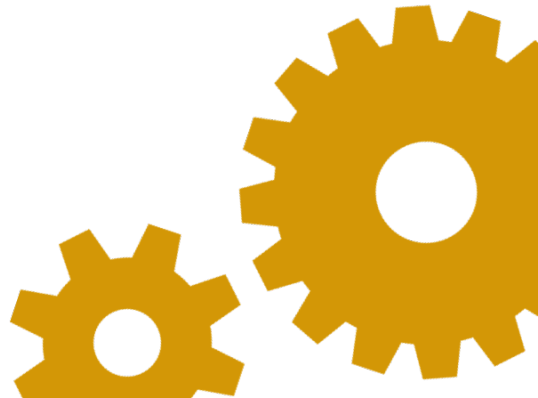
```
const myNumber = 8;
```

indentifier



JavaScript Syntax

- Camel casing versus underscore notation
  - Camel casing means uppercasing separate words in an identifiers name with the exception in most cases of the first letter
    - Example:
      - The name “my first variable” would be camel cased to “myFirstVariable”
  - Underscore notation means to separate words in an indentifiers name with “_”
    - Use lower case letters if you are using underscore notation
    - Example:
      - The name “my first variable” would be written using underscore notation as “my_first_variable”
  - Either method works fine
  - Stick with one method throughout your JavaScript code
  - Most JavaScript developers use camel case

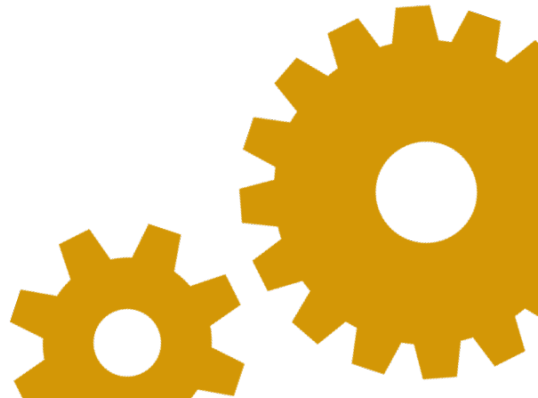


JavaScript Syntax

```
myFirstVariable  vs  my_first_variable
```

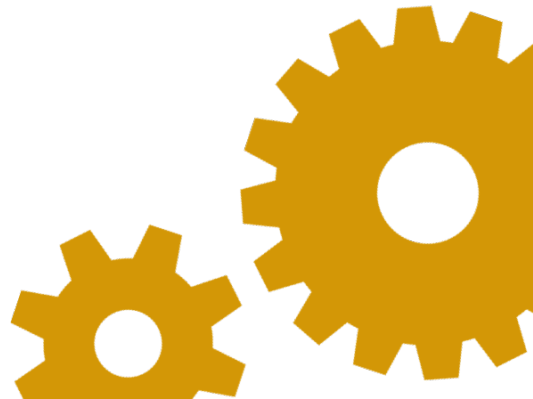
Camel Case Notation

Underscore Notation



JavaScript Syntax

- Naming recommendations for identifiers
  - Use meaningful names for identifiers
    - Names that describe what the identifier is or what the identifier does
  - Be consistent with your identifier names
    - If you start using camel case notation then continue using camel case notation throughout your script
  - If you are using underscore notation then use lowercase for all letters

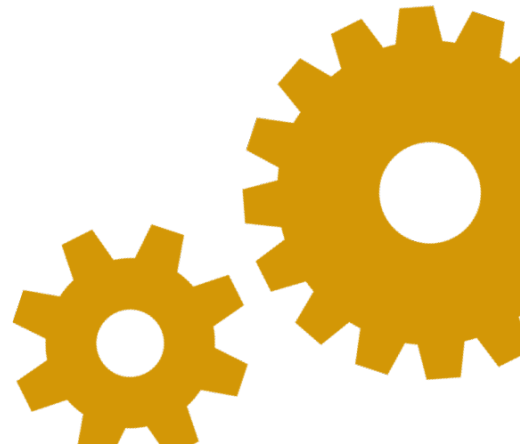


JavaScript Syntax

- Comments can be added to any JavaScript script
  - Similar to HTML and CSS comments, they are not read by the script but are placed inside the script as notes for other developers or yourself
  - There are two ways to write JavaScript comments
    - `//` for single line comments
    - `/* some comments */` for single line or multiline comments
      - Everything inside the `/* */` is a comment
  - Examples of JavaScript comments:

```
// Single line comments – used for shorter comments

/* Multiline JavaScript comment –
   Used for longer comments */
```

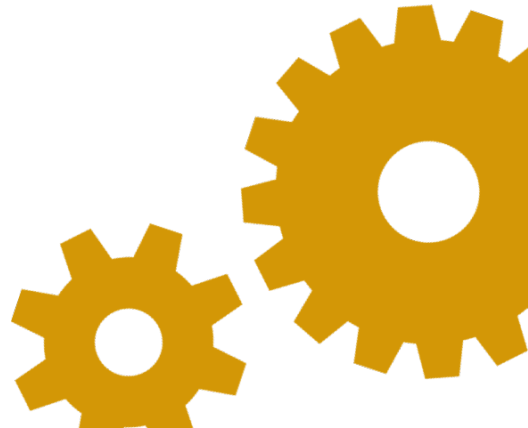


Comments are Public

- Remember, like HTML and CSS code, JS code is delivered to the browser as is, so any savvy user can view your JS code by opening developer tools in a browser
- This means your comments are public, so do not put passwords or other private information in them
- Use comments for:
  - To provide additional information for what your script is doing
  - To give credit to another developer
  - To provide version and copyright information about your script

JavaScript Syntax

- JavaScript mostly ignores white space
  - Use spaces and indenting to help make your code more readable
  - Using spaces and indenting is not required in order for a web browser to read your JavaScript but having nicely written code with good spacing and indenting helps make your code more readable



JavaScript Syntax

- JavaScript code examples with one example using very little white space and the other example using good indenting and good use of white space

Very little white space

```
function
cubeme(incomingNum){if(incomingNum==1
){return"What are you
doing?";}else{return
Math.pow(incomingNum,3);}}
var theNum=2;var
finalNum=cubeme(theNum);if(isNaN(fina
lNum)){alert("You should know that 1
to any power is
1.");}else{alert("When cubed,
"+theNum+" is "+finalNum);}
```

Good use of white space

```
function cubeme(incomingNum) {
   if (incomingNum == 1) {
      return "What are you doing?";
   } else {
      return Math.pow(incomingNum,3);
   }
}

var theNum = 2;
var finalNum = cubeme(theNum);

if (isNaN(finalNum)) {
   alert("You should know that 1 to any power is 1.");
} else {
   alert("When cubed, " + theNum + " is " + finalNum);
}
```

Above code samples from the book “JavaScript Step by Step – Second Edition” by Steve Suehring

Objects

- An object in browser based JavaScript is a thing or entity on the web page or in your code.
- The following are all objects in JavaScript
  - Browser window – the window object
  - The HTML document - the document object
  - Submit button – An HTML element object
  - Div element – An HTML element object
  - Any HTML element on a page can be a JavaScript object
- In addition to built in JavaScript objects you can create your own custom objects

Object Properties

- A property is a characteristic or attribute of an object
- A property can be set (written to) or retrieved (read from)
  - The "title" property of the "document" object sets or gets the title of the document
    - Set the title of the document
      - document.title = "Foo Corp Home Page";
        - Notice the dot (.) notation for accessing the property of an object

Object Methods

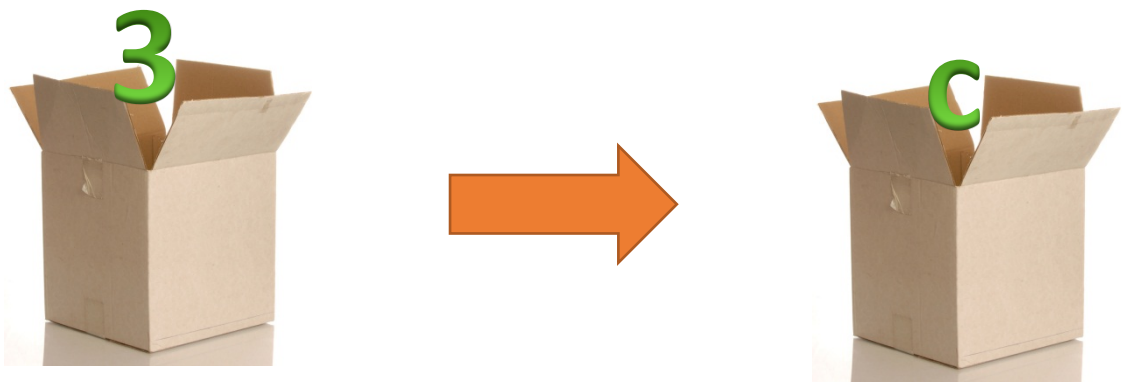
- A method is an action or something that an object can do
- You can sometimes recognize methods by the "()" following the method name
  - Some methods of the document object
    - document.write("Hello World");
      - This method writes the text "Hello World" to the HTML page (the document)
    - document.getElementById("foo");
      - Selects an element on the HTML page with an "ID" value of "foo"
  - Did you notice the text between the "()". This is how you pass in data into a method (a method is really another name for a function). More on functions later

Variables

- A variable is basically a storage location that stores information of a known or unknown value
  - source: http://en.wikipedia.org/wiki/Variable_(computer_science)
- Think of a variable as a storage box with a label that represents the variable name
- The storage box or variable usually stores a single piece of data but not always
- The value or data stored in some variable types can change

Variables

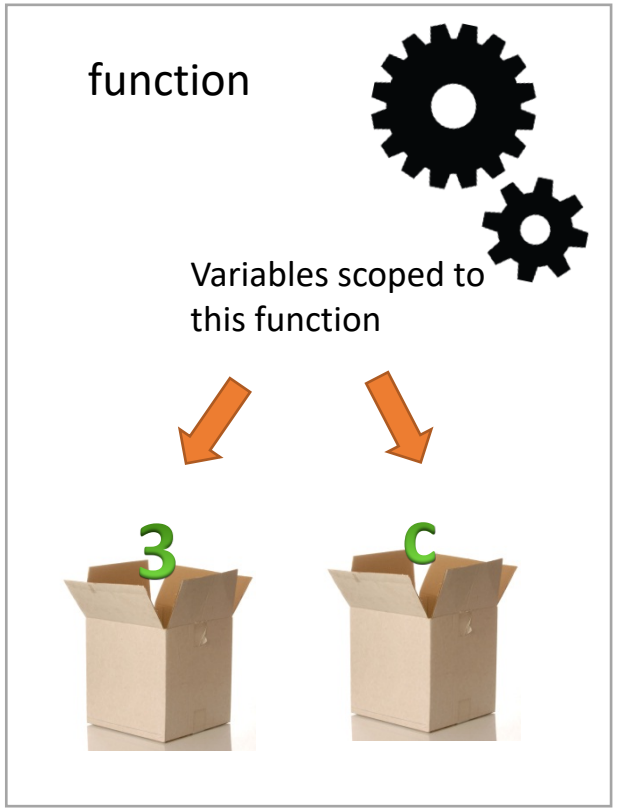
- Unlike some other programming languages, variables in JavaScript can change from one data type to another by being given a new value of a different data type
    - For example at one point a variable can be given a value of “17” which is a number, and then later the same variable can be given a value “zebra” which is a string



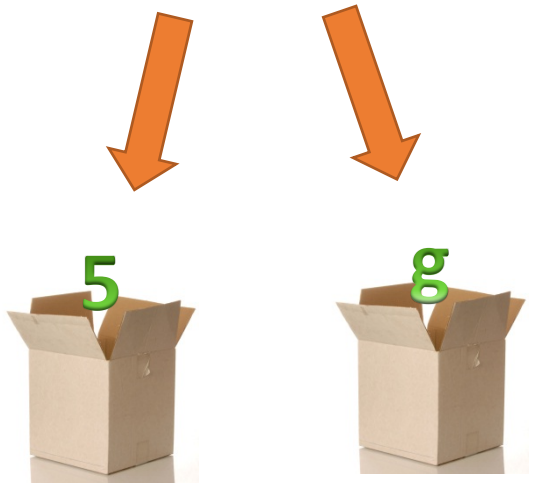
Variable Scope

- A variable’s scope determines what has access to the variable
  - Variables created outside of a function or a code block usually have a global scope and can be accessed anywhere in the script
  - Variables created inside a function are not accessible to code that is outside that function
  - Some variables created inside a code block are not accessible outside that code block

Variable Scope



These variables have a global scope and are available everywhere inside a script



Types of Variables

- Modern JavaScript has three types of variables
  - var
  - let
  - const

Variables set with "var"

- var
  - Legacy syntax. You will see var used a lot in code demos from a few years ago
  - Variables set with the "var" keyword can have their values change over the duration of the script
  - Variables set with "var" are globally scoped unless set inside a function

```
var bar = true;
```

Variables set with "let"

- let
  - Similar to "var" but with one key difference
  - "let" variables can have their values change over the duration of the script
  - They differ from "var" in that they are block scoped, meaning if they are set inside "{}" they are scoped to that code block. So if a "let" variable is declared inside an "if" statement it is locked to that block. This is different from "var"

```
let someValue = “zebra”;
```

Variables set with "const"

- const
  - Variables set with the "const" keyword can not have their values change. Once set they are locked to that value for the duration of that script or function call

```
const foo = 23;
```

When to use var, let and const

- If the variables value will change throughout the running of the script use the “`let`” keyword followed by the variable name
- For variables that will not have their values change throughout the running of the script or function call use the “`const`” keyword
- For legacy scripts or scripts that will run on older browsers use the “`var`” keyword

```
let someValue = “zebra”;
const foo = 23;
var bar = true;
```

JavaScript Debugging

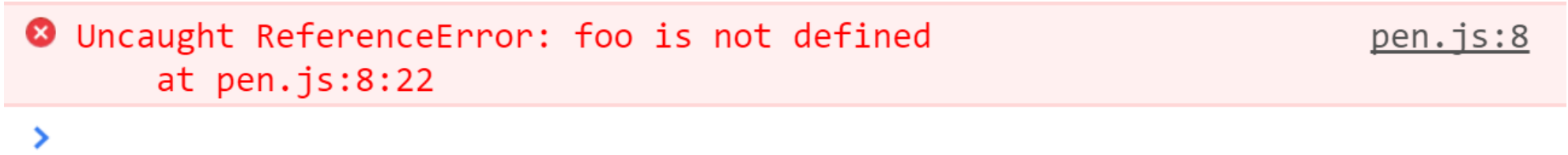
- Unlike other languages, there is no compiler to check your code before running in the browser.
- Unlike other languages JavaScript is designed to fail silently. Meaning your HTML and CSS will load fine and your JavaScript code will fail silently in the background (with no error messages displayed to the user). This is by design so that a page will still load even if there is an error in your JavaScript
- Due to the above points your best tool for debugging JavaScript is the developer console found in most browsers

Developer Tools - A JavaScript Developers Best Friend

- Most modern browsers have some sort of developer tools
- You have been using these tools to debug HTML and CSS, now you can use the same tools for debugging JavaScript
- To access the developer tools (or dev tools) in Chrome you can either:
  - Press F12 (Windows) / Ctrl+Shift+I / Cmd+Option+I (Mac)
  - Right click on the browser window and select "inspect"
- Once in developer tools in Chrome you will want to select the "Console" tab

The Console

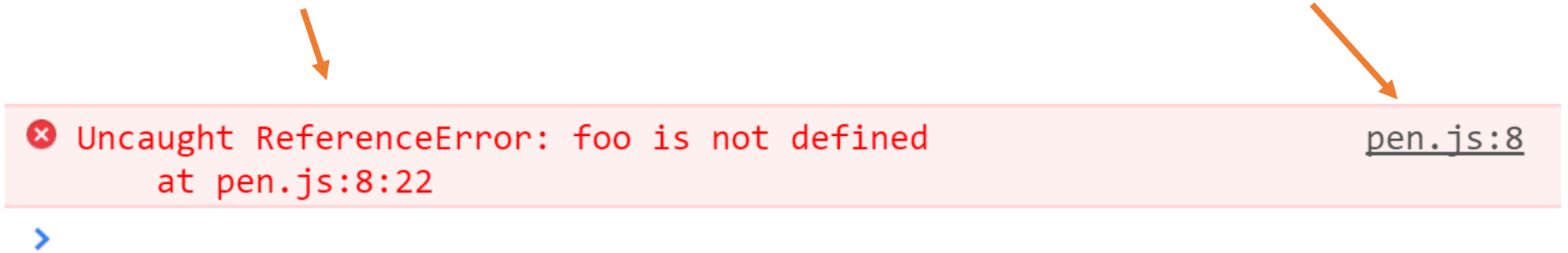
- The console is where all your JavaScript errors will be displayed
- The errors can be cryptic at first, but they will usually tell you a line number where the error occurred (this may not be the original cause of the error, but it will point you in the right direction)
- The errors will tell you want went wrong. For example if you called a variable "foo" that did not exist you would get this error in the console:



Console Error Breakdown

This is the error. In this case it is a "ReferenceError" meaning you have used an undefined identifier (variable name in this case)

This is the script name that caused the error along with the line number that triggered the error. This may not be the original cause of the error but it will point you in the right direction



Creating your Own Console Messages

- Outputting variable values or data to the console is very helpful for debugging or developing a script
- We can output custom messages to the console using the "console.log()" method in our scripts
- Simply place the "console.log()" statement wherever you want data to be outputted to the console in your script
- Pass in a message that you want displayed in the console by putting any data between the "()"
- You can pass in a message, a number, a variable, a combination of a message and a variable. Almost any type of data can be passed into "console.log()"

console.log() Messages

Pass in a Message

```
console.log('Hello Console');
```



```
Hello Console!
>
```

Pass in a Variable

```
const foo = 23;
console.log(foo);
```

```
23
>
```

Pass in a Message with a Variable

```
const foo = 23;
console.log('The value of foo is: ' + foo);
```



```
The value of foo is: 23
>
```

Window Object

- The window is a special object in JavaScript that has some built-in methods
- The window represents the browser window in JavaScript
- Since the "window" object is assumed you can sometimes omit the "window" object reference when calling some "window" methods
- Some common methods on the window object are:
    - alert(‘some string to display in a pop-up window’);
    - prompt(‘some question text’ , ‘optional default text’);
    - confirm(‘text to ask user to confirm something’);

Alert Method

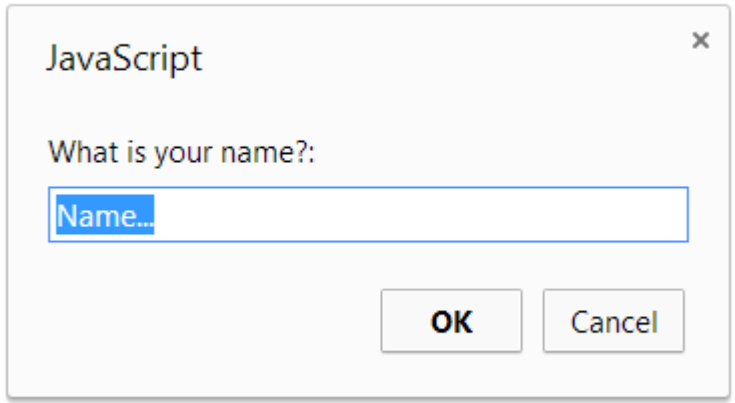
- Displays a pop-up message in the browser when the "alert" message is called
- The message to be displayed is passed into the "()"
- The pop-up window can not be styled or positioned. It will look different from browser to browser
- Not commonly used in production JavaScript but can be handy when developing a script to "alert" a variable value

```
alert(‘Hello World’);
```

Prompt Method

- The prompt method displays a pop-up box that asks the user to enter in some data.
- the prompt method returns the data that the user entered into the form. This data is often stored in a variable for later use.

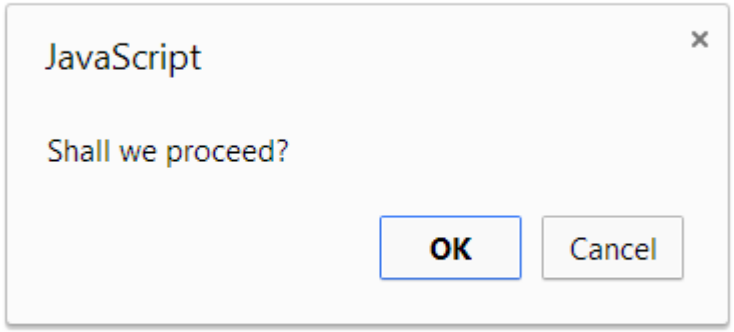
```
let theName = prompt('What is your name', 'Name…');
```



Confirm Method

- The confirm method displays a pop-up window that asks the user to confirm a statement
  - The user can either click on “OK” which would return “true” or “Cancel” which would return “false”
  - The confirm method returns a Boolean value meaning it only returns the values of “true” or “false”

```
let userConfirm = confirm('Shall we proceed?');
```



Conditional Statements

- Conditional Statements allow your script to go in different directions depending if a statement is true or false
- In JavaScript, one form of a conditional statement is the "if/else if/else" or "if/else" statement

```
if (condition) {
    … commands to execute if condition is true
}
else {
   … commands to execute if condition is false
}
```

Comparison Operators

Operator	Description	Example	Sample values of quantity that would result in true
= =	Double equals sign (equivalent) "is exactly equal to"	quantity = = 10	10
>	Greater than	quantity > 10	11, 12 (but not 10)
> =	Greater than or equal to	quantity > = 10	10, 11, 12
<	Less than	quantity < 10	8, 9 (but not 10)
< =	Less than or equal to	quantity < = 10	8, 9, 10
! =	Not equal to	quantity ! = 10	8, 9, 11 (but not 10)

Functions

- A function is a block of one or more JavaScript statements with a specific purpose, which can be run when needed.
- A function can be reused and called multiple times in your script

```
function doSomething()  {
   ... JavaScript statements …
}
```

Using Functions

- You can define functions anywhere in the same scope in your script and they are available anywhere else in the same scope or in child scopes
- To run or "call" a function you simply write the functions name followed by "()"
  - For example to call the function "addNumbers" you would write:
    - addNumbers();
- Inside the "()" you can pass in arguments to a function by putting values inside the "()". Each argument should be separated by a ","

Getting Data out of a Function

- Some functions just perform a task such as outputting a name to the screen, they do not "return" any data
- If you need to get data out of a function use the "return" key word followed by the data that you want to pass out of the function

Call the "addNumbers" function and store its returned data in a variable called "total"

```
let total = addNumbers(2, 3);
console.log(total) // outputs 5
```

The addNumbers function. Notice the return statement at the end of the function

```
function addNumbers(num1, num2){
    return num1 + num2;
}
```

In Class Exercise

- Create and attach a JavaScript file to an HTML document
- Write the JavaScript that does the following:
  - Create a function that does the following
    - Asks the user (via a prompt box) their age
    - If the user is over 18 display a message (via an alert box) that says
      - "You are permitted to vote in BC."
    - If the user is under 18 display a message (via an alert box) that says
      - "Sorry you can not vote in BC."
  - Call the function created in the above step anywhere in your script

Data Types

- A data type is a way to classify data and to determine what the possible values of data of a certain type can be

Data Types

Common JavaScript Data Types

Data Type	Possible Values
Strings	Any text
Numbers	Any number
Booleans	true, false

Determining Data Type

- To determine the data type of a value or a variable in JavaScript use the "typeof" keyword followed by a value or a variable

```
const a = 23;
const b = 'cat';
const c = true;

console.log(typeof a); // outputs "number"
console.log(typeof b); // outputs "string"
console.log(typeof c); // outputs "Boolean"
```

Strings

- A string is a finite sequence of symbols that are chosen from a set called an alphabet.
    - source: http://en.wikipedia.org/wiki/String_(computer_science)
- More commonly, strings are data that are words or letters made up from an alphabet
    - An example of strings:
        - “Hello World”
        - “C”

Strings

- Strings and JavaScript
  - To create a variable with a data type of a string, simply create a variable and set the variable's value as a value surrounded by double or single quotes
    - Example:
      - `const someString = "This variable is storing a string";`
  - Unlike some other computer languages, JavaScript sets the data type of the variable based on what the value of the variable is
  - Also in JavaScript a variable's data type can change if it's value and data type changes

Strings

- Quotes inside strings
  - How do you store a string such as these:
    - The man said “Hello”
    - I’m the person that said “Hello”

Strings

- If you wish to store a quote inside a string of text than you have a few options:
  - Option 01:
    - Alternate your quote types between double and single quotes
      - `const someString = ‘The man said “Hello”’;`
    - This will not work with the second example string:
      - `const anotherString = ‘I’m the person that said “Hello”;`
        - This will fail because the apostrophe in “I’m” will cause the initial opening single quote to close after the letter “I”
        - Use option 02 or option 03 for strings which contain various quotes and apostrophes
  - Option 02:
    - Use the “\” escape character to escape the next character in the string
    - Using the “\” escape character tells JavaScript to ignore any special meaning that the next character may have
      - `const yetAnotherString = ‘I\’m the person that said “Hello”’;`
  - Option 03:
    - Use Template Strings (more on template strings in upcoming slides). Template strings use back ticks (``) to store strings
      - `const aString = \`I’m the person that said “Hello”\`;`

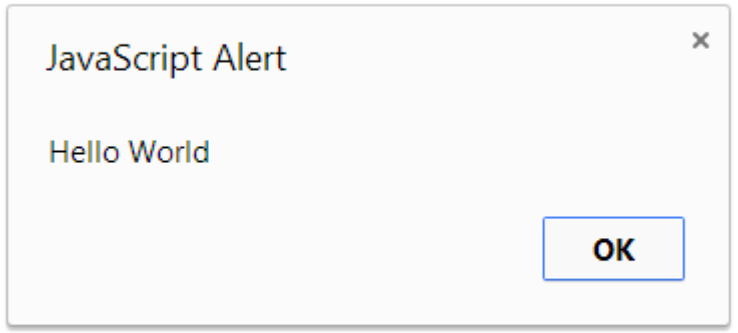
Strings

- Concatenating string values
  - To concatenate string values means to combine two or more strings together to form a single larger string
    - To combine strings together you use the “+”

```
let a = ‘Hello’;

let b = ‘World’;

alert(a + ‘ ’ + b);
```



This empty string adds a space between the two words

Template Strings

- Another type of string which is new to JavaScript is a template string
- Template strings are formed by wrapping text in backticks (``)
- Template strings allow for variables to be placed directly inside the string by using the *${variableName}* syntax
- Below is an example of a template screen that has a variable inside the string

```
const output = `The country is ${country}`;
```

- Expressions can also be resolved inside template strings
- Below is an example of an expression being resolved inside a template string

```
const sum = `12 + 6 = ${12 + 6}`;
```

Numbers

- A number is pretty much just what it says, it is a number
  - The following are numbers
    - 2
    - 0
    - -3
    - 0.23589
    - -0.12
    - 6.82
    - -8.276
  - Unlike other computer languages integers (whole numbers) and floating point (numbers with decimals points) do not have special or separate data types

19 **19.5**

Arithmetic Operators

Operator	Description	Example	Value of Quantity
=	assign	quantity = 10	10
+	addition	quantity = 10 + 6	16
-	subtraction	quantity = 10 - 6	4
*	multiplication	quantity = 10 * 2	20
/	division	quantity = 10 / 2	5

Booleans

- Variables that have a data type of a Boolean can only store the values of true or false
- Often used in conditional statements
  - Example:

```
let x = true;

if (x == true){
   // do something
};
```

true **false**

Strings that Look Like Numbers

- Often in JavaScript we are retrieving data from our users via a form
- When data is returned from a form to JavaScript the data is always in the form of a string
  - For example if the user enters their age in a form then submits the form the data that is returned is "43" (or whatever age they entered). Notice the quotes around the number. This number looks like a number but is actually a string
  - The problems arise when we try to perform some math functions on the number using the "+" operator. If we try to add the number to 2 to the string number "43" we would get "432" and not 45 as JavaScript thinks "43" is a string so concatenates 2 on to the end of the string

Converting String Numbers to Numbers

- Due to the common problem mentioned in the previous slide, there are numerous ways to convert a string number to a number
  - Option 1
    - Simply multiply the string number by 1. JavaScript will convert the string number to a number to perform the multiplication. Since you are multiplying by 1 the value does not change
      - const valueFromFormInput = "43";
      - const num1 = valueFromFormInput * 1 // converts to the number 43
  - Option 2
    - Use parseFloat() for decimal numbers or parseInt() for whole numbers
      - const valueFromFormInput = "52.345"
      - const num2 = parsefloat(valueFromFormInput) // converts to the number 52.345
  - Option 3
    - Use the Number() method
      - const valueFromFormInput = "0.235"
      - const num3 = Number(valueFromFormInput) // converts to the number 0.235

Selecting HTML Elements With JavaScript

Selecting HTML Elements with JavaScript

- One of the primary purposes of JavaScript is to interact, update, delete and modify HTML elements on the a web page
- To modify an HTML element we first have to select it
- JavaScript provides many ways to do this. Some of the common methods for selecting an HTML element include these methods of the "document" object:
  - document.getElementById();
  - document.querySelector();
  - document.querySelectorAll();

Selecting an HTML Element By ID

- With JavaScript you can easily select an HTML element by it's ID attribute
- This is the fastest method to select an element

```
const heading = document.getElementById('main_heading');
```

The above JavaScript code will select the below HTML element

```
<h1 id="main_heading">Main Heading</h1>
```

Selecting an HTML Element Using querySelector

- With JavaScript you can select an HTML using a query selector similar to CSS
- IMPORTANT: If multiple elements match a query only the first instance on the page will be selected

```
const el = document.querySelector('.box p');
```

The above JavaScript code will select the below paragraph element

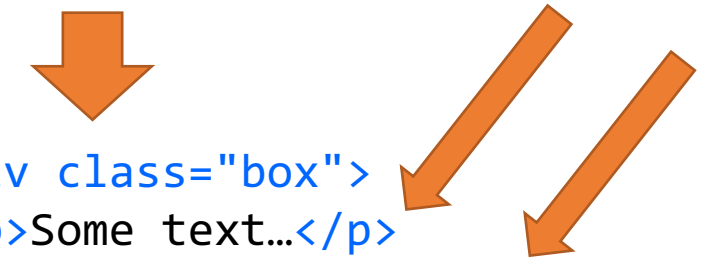
```
<div class="box">
 <p>Some text…</p>
</div>
```

Selecting an HTML Element Using querySelectorAll

- Similar to querySelector, querySelectorAll will select all the elements on a page that match a CSS selector
- IMPORTANT: Unlike getElementById and querySelector, querySelectorAll will return a node list instead of an element

```
const el = document.querySelectorAll('.box p');
```

The above JavaScript code will select all the below paragraph elements



```
<div class="box">
 <p>Some text…</p>
 <p>Some other text…</p>
</div>
```

Modifying An HTML Elements Properties

- Once you have selected an HTML element you can modify it's properties
- One of those properties is the "innerHTML" property
- The "innerHTML" property sets the text of the element or the HTML inside the selected element
- You can set it to a string or you can set it to a string with additional HTML tags inside
  - el.innerHTML = 'I was changed by JavaScript!';
  - el.innerHTML = ' I was changed by JavaScript ';

Setting An Elements text with Inner HTML

- You can set or replace an HTML element's content by setting the HTML element's innerHTML property value

```
const heading = document.getElementById('main-heading');
```

```
heading.innerHTML = 'Heading Text Changed');
```

The above JavaScript code will change the content of the <h1> element with an ID of "main-heading". See below for the output.

Heading Text Changed

Event Listeners

JavaScript Event Listeners

- JavaScript is an event driven language
- Functions can be triggered when a certain event happens in the browser
- You can set up "event listeners" on any HTML element. Event Listeners "listen" for an event to happen to on their attached HTML element. When that event happens, a function linked to that event listener is triggered, running that function's code
- There are many types of browser events that JavaScript can listen for

Common JavaScript Event Types

- Some common JavaScript events include:
  - "click"
    - Triggered when a user clicks or taps an element
  - "load"
    - Triggered when an element has completely loaded into the browser
  - "mouseover"
    - Triggered when a user mouse's over an element
  - "mouseenter"
    - Triggered when a user's mouse enters an element
  - "mouseout"
    - Triggered when a user's mouse leaves an element
  - "submit"
    - Triggered when a user submits a form

Attaching Event Listeners

- To attach an event listener to an HTML element use:
  - el.addEventListener()
- The "addEventListener()" takes three arguments, but only two are required and only these two required arguments are used for most scenarios

```
const el = document.querySelector('button');
el.addEventListener('click', someFunctionToRunWhenButtonIsClicked);
```

The event to listen for

The function to run when the listeners event is triggered

Click Event Listener using addEventListener

- You can "listen" for clicks on an HTML element and when a click happens you can run a set of instructions called a function
- You attach an event listener to an element by using the addEventListener() method

```
const btn = document.getElementById('btn-01');
```

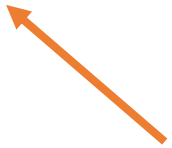
```
btn.addEventListener('click', someFunction);
```

The "btn" element now has a click event listener attached to it and JavaScript is now listening for clicks that appear on the button. When a click does occur the function "someFunction" is run

Getting Form Field Values

- You can get the value entered into a form field using the “value” property of a selected form field element
- You usually want to get the values of the form after the user has submitted the form
  - Normally you would retrieve them in the "submit" event function that would be fired when the user submits a form

```
const formElementValue = formElement.value
```



The value property retrieves the value from a form

```
const formElement = document.getElementById('input_name');
```

Attach a "submit" event listener to the "form" element and listen for the "submit" event to happen. When the "submit" event occurs do the next step inside the "submit" event function

```
const formElementValue = formElement.value;
```

The above JavaScript code will get the value from a form input field and store it in a variable. To output the value simple call the variable. See below

```
alert(formElementValue);
```