**JavaScript**

Along with HTML and CSS, JavaScript is one of the 3 core technologies of the World Wide Web. It is weakly-typed, ***client side***, interpreted programming language. It is used by developers to add more functionality and interactivity to websites.

**Client-side Scripting:** is code that is executed by a browser. When a webpage is requested from a server and is received by the computer- the code will then run. This allows JavaScript to be executed based on user interaction (pressing a button) without needing the page to load.

**Server-side Scripting:** is executed by a web server. When a webpage is requested the script is first executed and then the page is sent to the computer.

While JavaScript is and was always intended as a client-side scripting, in recent times there have been frameworks set up to use it for server-side requests. These include AJAX and Node.js to name but a few.

JavaScript is also used in non-web-based environments such as desktop widgets, mobile applications, PDF documents and video game development.

Despite some naming, syntactic, and standard library similarities, ***JavaScript and Java are otherwise unrelated*** and have very different semantics.

**Where to Use JavaScript**

Because JavaScript is supported by all modern Web browsers, there is no need for plug-ins to get it to work online. All that is required, is for the following tags to be added into the HTML document and all corresponding JavaScript code to be added between them.

**<script> </script>**

These script tags can be placed in the **<head>** or **<body>** section of the page however it is generally recommended to place all JavaScript code at the very bottom of the page just before the closing **</body>**. The reason for this is as follows:

When a web page is loading in the browser the HTML begins to be parsed. When it encounters a **<script>** tag the HTML parsers stops and waits for the JavaScript to be run and executed. It therefore interrupts the loading of HTML - the visual components of the page, which may ruin a user’s experience.

Placing the script at the end at least allows the HTML and CSS to load up first before any interruptions.

Despite these facts however developers do debate this topic.

**Debate about where to place JavaScript Code**

<http://stackoverflow.com/questions/436411/where-is-the-best-place-to-put-script-tags-in-html-markup>

<https://developers.google.com/speed/docs/insights/BlockingJS#InlineJS>

It is more common to place JavaScript code in a separate **.js** file and link that file to a HTML page as shown here:

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

Doing it this way allows you to separate all JavaScript code from markup HTML which is considered neater. In the above example myscript.js contains js code which is interpreted by the browser as soon as that line of code is read. Nothing should go between the script tags here. For the purpose of our in class tasks, we will be using the simpler inline <script> tags as it is quicker.

**Variables in JavaScript**

Because JavaScript is weakly-typed, it means you don’t have to specify the data type when you declare and assign a variable.

To declare a variable in JavaScript you must use the one of these keywords (**var**, **const**, **let**)keyword and follow it with the name (also known as the identifier) of the variable. To assign a value to a variable you must use the **=** sign (the assignment operator). You can declare a variable and assign it a value in the same line.

**<script>**

**var myVariable;  *//*** *Declaring a variable*

**myVariable = "This is an String";  *//*** *Assigning a value to variable*

**var anotherVariable = "Another String"; *//*** *Declaring and assigning a variable*

**alert( myVariable ); *//*** *Creates pop up window with “This is a String”*

**anotherVariable = "Assigning a new value"; *//*** *Reassigning a new value to an existing variable*

**</script>**

**\*NOTE** that each line of JavaScript should end with a semicolon however this is not mandatory.

To use the variable later on in code you just need to reference the name of the variable. In the last line of code, an alert box will be displayed showing “This is a String”. Notice how when you refer to the variable you don’t use the inverted commas (**""**).

**Rules to Naming Variables**

1. Variable names/identifiers must be unique. A common mistake is creating variables with the same name which can happen in large scale projects. This could have the effect of reassigning a value to an existing variable as opposed to creating a completely new one.
2. Names can contain letters, numbers, underscores \_ and dollars signs
3. Names should not begin with a number
4. Names are case sensitive. **myName != myname**. These are different variables because N is capitalized in the first name.
5. Names should not contain reserved keywords (words that JavaScript already use in their language) e.g. **function**, **var**, **const**, **let**, **for**

**Four Ways to Create Variables in JavaScript**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Can change/reassign value** | **Scoping** | **Example** | **Notes** |
| **let** | Yes | Block scoped | **let score = 0;** | Use this unless variable must be a fixed value. |
| **const** | No | Block scoped | **const gravity = 9.8;** | Use this unless variable must change or update. |
| **var** | Yes | Function scoped | **var fName = "James";** | Used to be the main way of declaring variables but has gone out of favour with recent updates to js. It is used today however when declaring functions as variables. |
| *No keyword* | Yes | Global variable | **celebrity = "Britney";** | Should be avoided because the variable becomes global and can be accessed anywhere in the code. |

**Useful JavaScript Code**

**alert( "This is an Alert Box" );**

**console.log( "This message will appear in the console window" );**

**const myPrompt = prompt( "This is a Prompt Box" );**

**const myConfirm = confirm( "This is a Confirm Box" );**

**//** *This is a single line comment*

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

*which must be closed off* **\*/**

The alert, prompt and confirm boxes produce windows on a web page that can be used to display messages or get data.

For the **prompt** box, whatever the user types into the text field will become the value of the variable it is set to. In this case if the user types “James” into it, the variable **myPrompt** will be assigned the string “James”.

**myPrompt = "James";**

The **confirm** box is slightly different. The user can only press OK or Cancel and this will thus produce a Boolean value (true or false). If the user presses OK then the variable **myConfirm** will equal true.

**myConfirm = true;**

**Useful Coding Terminology**

**Expression**: An expression is any unit of code that can be evaluated to a value. One example would be what you would find inside the brackets of an **if ()** statement. The code in here should evaluate to either true or false.

**Statement**: A statement is an instruction to perform a specific action. Such actions include creating a variable or a function, looping through an array of elements, evaluating code based on a specific condition etc. JavaScript programs are actually a sequence of statements. Statements do not necessarily return a value.

**BEDMAS**

When carrying out somewhat complex arithmetic (math) in code it becomes useful to remember the mnemonic BEDMAS sometimes referred to as BODMAS or BOMDAS. This explains the order in which math equations should be carried out. For instance:

**3 + 6 x 2 = 15** because you must multiply **6** and **2** first before adding **3**

since multiplication **M** precedes addition **A** in BEDMAS

The following link explains all: <https://www.mathsisfun.com/operation-order-bodmas.html>

**Conditional Statements: if else**

Conditional Statements are used extensively in JavaScript and all other programming languages to affect the flow of code. The below case is an example of an **if**, **else if**, **else** statement whereby if the first 2 conditions aren’t satisfied then the code block in the final **else** statement is executed.

**var myAge = 18;**

**if ( myAge <= 12 ) {**

**alert( "You are a child" );**

**}**

**else if ( myAge > 12 && myAge < 19 ) {**

**alert( "You are a teenager" ); //** *This alert will occur because myAge = 18*

**}**

**else {**

**alert( "You are an adult" );**

**}**

In the example above you can see the use of...

* **logical operators**
  + **&&** and
  + **||** or *(this was not shown above)*
* **comparison operators**
  + **<=** less than or equal to
  + **>**  greater than
  + **<** less than

...which extend the conditions of the conditional statement.

**Other examples of Conditional Statements**

**Conditional Ternary Operator** acts as a shorthand way of writing a simple if else. These can be extended or chained together like **if**, **else** **if**, **else** **if**, **else** however they become a lot harder to distinguish this way so are best kept simple. They are primarily used to assign values to a variable based on a condition.

**(** a condition expression **) ?** execute this code if condition is true **:** execute this code if condition is false **;**

**const drink = ( age >= 19 ) ? "Beer" : "Milk";**

In the above example the variable drink is assigned the value of “Beer” if age is greater than or equal to 19 or “Milk” if it is not.

**Switch Statement** takes in an expression and based on what it becomes performs different actions. The Switch expression is compared strictly (**===**) to the values placed beside each **case** in the code block. If it matches it begins to execute that case’s code until it hits the **break** keyword and then exits the switch block. Don’t forget the **break** keyword because if omitted the code will continue executing. The **default** at the end, executes if the expression fails to match any previous **case**. There is no need for a **break** at the end of a **default**. Switch statements look very straight forward and are suited to having many different paths or outcomes (imagine having lots of else ifs). They are limited in that they cannot use comparison operators like **>** or **<** appropriately.

**const colour = "green";**

**switch( colour ) {** // *The expression is evaluated inside the brackets ()*

**case "red":**

// *Ignores code because "red" !== "green"*

**break;**

**case "orange":**

// *Ignores code because "orange" !== "red"*

**break;**

**case "green":**

// *Executes this code*

**break;**

**default:**

// *If switch expression doesn’t match any*

// *of the cases, this code will execute*

**}**

**Common Methods and Properties of Strings and Numbers in JS**

Programming languages come with their own built-in functions that allow you to deal, interact or change certain data types. When you’re writing code and have to make changes to a variable it’s a good idea to look online and see if there is something available to you that can assist you in this regard. For instance if you have a number and want to format it to look like a price (giving it two decimal places) you could use **toFixed()**. If you have a string and want to capitalize all the letters you could use **toUpperCase()**. A **method** is another word for a function and deals with performing an action like the two examples mentioned above. A **property** refers to a description such as the **length** of a string which tells you how many characters it has.

In the next section we’re going to be using the variable **let num = 10.5** and **let str = "James"** in demonstrating some of these useful JavaScript number and string methods.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **String Method** | **Outcome** |  | **Number Method** | **Outcome** |
| **str.length** | 5 |  | **isNaN( num )** | false |
| **str.toUpperCase()** | JAMES |  | **num.toFixed( 4 )** | 10.5000 |
| **str.charAt(0)** | J |  | **parseInt( num )** | 10 |
| **str.slice(0, 2)** | Ja |  | **parseFloat( num )** | 10.5 |
| **str.indexOf(“m”)** | 2 |  |  |  |

For more examples of these methods look up: <https://www.w3schools.com/js/js_string_methods.asp>

<https://www.w3schools.com/js/js_number_methods.asp>