**The Ultimate Handbook of Useful JavaScript Syntax**

*A work inspired by Marijn Haverbeke’s “Eloquent JavaScript”*

**Siddarth Narasimhan**

**User Notes**

These notes are based on the “*Eloquent* *JavaScript*” book that can be found online here:

<https://eloquentjavascript.net/index.html>. The definitions in this document can be better described by reading the book above, which also contain examples and practice questions.

All JavaScript in the book are not shown in this document. Only the useful, uncommon or hard to remember syntax are provided and explained. This document is best used upon having some prior coding knowledge and experience.

The column on the left shows you the syntax for the function whereas the column on the right explains how that function is used and what it is used for.

In the following sections, a word in italics indicate that it is a parameter to be inputted by the user. A word written normally is the function or syntax that is to be used to achieve the desired action. An optional parameter is represented by using *Optional:* followed by the optional parameter name .

Good luck!

**1 Data Structures: Objects and Arrays**

|  |  |
| --- | --- |
| **Syntax** | **Use** |
| *Variable*.toUpperCase  *Variable*.toLowerCase | Convert a string *Variable* into all uppercase/lowercase characters. |
| *Array.*push(*Element*)  *Array.*pop() | 1. Add an *Element* to the end of an *Array*.  2. Remove the *Element* from the end of the *Array.* |
| delete *Object*.*Property* | Deletes the *Property* from the specified *Object.* |
| console.log(“*ObjectProperty*” in *Object*) | Outputs whether the *Object* has a property called *ObjectProperty,* with a Boolean return type. |
| Object.keys(*Object)* | Returns an array containing all of the properties in *Object*. |
| Object.assign(*Object, Property)* | Copies each *Property* into the *Object,* and assigns each property in the *Object* with the specified values of each *Property.* |
| *Array.*includes(*Value*) | Outputs whether the *Array* contains *Value*, with a Boolean return type. |
| for (let entry of *Array*) | Loops through the values in the *Array*. |
| *Array.*shift()  *Array.*unshift(*Element*) | 1. Removes the first item of the *Array.*  2. Adds an *Element* to the beginning of the *Array.* |
| *Array.*indexOf (*Element*, *Optional: Start*)  *Array.*lastIndexOf (*Element, Optional: Start*) | 1. Searches the *Array* left to right for an *Element* and returns the index at which the *Element* was found. Optional argument for what index to start searching at.  2. Searches the *Array* right to left for an *Element* and returns the index at which the *Element* was found. Optional argument for what index to start searching at. |
| *Array.*slice(*start, end*) | Returns a segment of an array, sliced between the *start* and the *end* indices. If no *end* index is provided, it sets *end* to the end index of the array. |
| *Array.*concat(*ConcatArray)* | Concatenates *ConcatArray* to the end of *Array*. |
| *Variable.*trim() | Removes any spaces or newlines in *Variable.* |
| *Variable.*padStart(*newLength, padSymbol*) | Adds *padSymbol* to the beginning of *Variable* until the specified new length of *Variable* (*newLength)* is achieved. (eg. 3 can be represented as 003). |
| *Variable.*repeat(*NumberOfTimes)* | A new variable is created where the contents of *Variable* is repeated *NumberOfTimes* times. |

**2 Higher-Order Functions**

|  |  |
| --- | --- |
| **Syntax** | **Use** |
| *Array.*forEach(*Parameter* => *FunctionAction*) | Loops through each value in the *Array* and outputs the *FunctionAction* that you specify. |
| *Object.*filter(*Parameter* => *Parameter.Property* == *DesiredValue*) | Looks through an object and returns all objects in the main object where the property equals the desired value. Works in similar to the WHERE clause in SQL. |
| *Array.*map(*FunctionUsedToMap*) | Outputs a new array with the *FunctionUsedToMap* function being applied to all of the elements in the *Array*. Can be applied to objects as well. |
| *Array.*reduce(*FunctionUsedToReduce*) | Combines all the elements in *Array* into a single value, depeding on the *FunctionUsedToReduce* |
| *Array.*some(*Function*) | Returns whether the *Function* returns true for any of the elements in the *Array*. |
| *Array.*findIndex(*Function*) | Finds the index in the *Array* where the *Function* returns true. |

**3 The Secret Life of Objects**

|  |  |
| --- | --- |
| **Syntax** | **Use** |
| Object.getPrototypeOf(*Object*) | Outputs the prototype of the *Object.* |
| Object.create(*Object*) | Creates a new object with the same prototype as *Object*. |
| Symbol.iterator | Allows you to make classes iterable. This is a new function that needs to be made inside the class that you want to be iterable. |
| get()  set(*Value*) | These are general functions declared inside a class.  1. Used to retrive information.  2. Used to change the *Value* of a variable. |
| **Pillars of OOP:**  Encapsulation  Abstraction  Inheritance  Poly-Morphism | 1. Group related variables together (into an object) which reduces complexity.  2. Hide the details and complexity and only shows the essentials.  3. Eliminate redundant code.  4. Ability to process objects differently depending on their type. Can help refactor long switch/case or if/else statements. |

**4 Project: A Robot**

|  |  |
| --- | --- |
| **Syntax** | **Use** |
| Object.freeze(*Object*) | Ensures that the object’s value(s) cannot be changed. |

**5 Bugs and Errors**

|  |  |
| --- | --- |
| **Syntax** | **Use** |
| throw new Error(*Message*) | Stops the code and reports an error to the user with the specified *Message*. |
| try{}  catch{}  finally{} | Syntax for error handling. Runs the “try” code block. If any error occurs, it runs the “catch” code block. The “finally” block can be used to fix an error, which runs no matter what happens in the “try” block. |

**6 Regular Expressions**

|  |  |
| --- | --- |
| **Syntax** | **Use** |
| RegExp(“*Expression”*) | Creates a regular expression, called *Expression.* |
| *RegExp*.test(*TestString*) | Checks whether the *TestString* contains *RegExp.* |
| **Character Groups** | |
| [^*ExpressionToMatch*] | Matching any character except the ones in the *ExpressionToMatch* set. |
| *RegExp*.exec(*TestString*)  *RegExp*.index  *String*.match | 1. Returns null if *RegExp* is not found in *TestString*. Otherwise returns, the string in the array for which *RegExp* was found in the *TestString.*  2. Returns the position in *TestString* that *RegExp* is found.  3. Has similar properties as the others. |
| console.log(new Date())  console.log(new Date().getTime) | 1. Returns the current date.  2. Returns the number of milliseconds since Unix Time (1970) |
| let newRegExp = RegExp(\b hello \b/) | Add this to the beginning or end of the regular expression. If it is added at the beginning, it returns true if the word in question is at the start of the word. If it is added on both sides, it returns true if it is a separate word (true) or if it is contained within the word (false). |
| let newRegExp = RegExp(pig|horse) | The pipe denotes a choice between the pattern to its left and the pattern to its right. |
| *RegExp.*replace(*Find,* *Replace*) | Finds the first instance of *Find* in the *RegExp* and replaces it with *Replace*. If you want to replace all instances, you add ‘g’ at the end of the regular expression. |
| *RegExp.*search(*SearchStr*) | Returns the index where *SearchStr* was found in *RegExp.* |

**7 Modules**

|  |  |
| --- | --- |
| **Syntax** | **Use** |
| NPM | Repository containing several JavaScript packages. |

**8 Asynchronous Programming**

|  |  |
| --- | --- |
| **Syntax** | **Use** |
| setTimeout(*Function, Timeout*) | Waits a *Timeout* specified in milliseconds before running the *Function.* |

**9 Document Object Model**

|  |  |
| --- | --- |
| **Syntax** | **Use** |
| document.getElementById(*ID Name*) | Bridge to HTML. Allows one to return an element written in an HTML file with an ID Name of *ID Name*. |
| document.createElement(*Element*) | Creates an HTML *Element*. The *Element* is the tag name written as a string. |
| *AppendTo*.appendChild(*Child*) | Appends *Child* to *AppendTo*. |
| document.querySelector(*TagName*)  document.querySelectorAll(*TagName*) | 1. Returns the method for which the *TagName* was first matched in the document.  2. Returns all methods in a list for which the *TagName* was first matched in the document. |

**10 Handling Events**

|  |  |
| --- | --- |
| **Syntax** | **Use** |
| window.addEventListener(*Action, Function*) | The *Function* is triggered when the *Action* is performed by the user.  Possible Values for *Action*: “keydown”, “keyup”, “click” |
| *button.*removeEventListener | Removes the event listener from a *button*. |
| event.button  event.key(*Key*)  event.buttons | 1. Monitors which part of the user’s mouse was used to click the button.  2. Event is triggered when *Key* on the keyboard is hit.  3. Tells us whether a button is held down. |
| **Events**  keydown  keyup  mousedown  mouseup  click  dblclick  mousemove  touchstart  touchmove  touchend  scroll  focus  blur  load | All these events are used as the first parameter in window.addEventListener() |

**11 Drawing on Canvas**

|  |  |
| --- | --- |
| **Syntax** | **Use** |
| let context = document.querySelector (“canvas”).getContext(“2d”) | Creates a 2D canvas. To create a 3D canvas, use WebGL. |
| context.fillRect(*p1,p2,p3,p4*)  .fillStyle  context.strokeRect(*p1,p2,p3,p4*)  .StrokeStyle  .lineWidth | 1. Creates a rectangle in the canvas. *p1* is the rightward positioning of the rectangle from the top left corner of the canvas and *p2* is the downard positioning. *p3* and *p4* correspond to the x and y size of the rectangle.  2. Creates a rectangle frame, with the same parameters as fillRect. |
| context.beginPath()  context.moveTo(*startX,startY*)  context.lineTo(*endX,endY*) | 1. Allows you to build paths (lines)  2. Begins the line at the *startX* and *startY* coordinate.  3. Ends the line at the *endX* and *endY* coordinate. |
| context.quadraticCurveTo(*controlX, controlY, endX, endY*) | Creates a curve. The *controlX* and *controlY* are the coordinates to which the curve is attracted to. Use the moveTo method to begin the curve. Specify the end of the curve using *endX* and *endY.* |
| context.arc(*centreX, centreY, radius, arcStartAngle, arcEndAngle*) | Creates a circle with its centre at coordinates (*centreX, centreY*) and radius equal to *radius.* The *arcStartAngle* is the angle from the first quadrant for which to begin drawing the circle and the *arcEndAngle* is the end angle to stop the circle, both specified in radians. |