**JavaScript.Info Notes** <https://javascript.info/object-methods>

**Object methods, “this”**

Objects represent real world entities. Things = Objects

Functions perform an actions. Actions = Functions

\* A function that is the property of an object is called its method.

**“this”**

In JavaScript, keyword this behaves unlike most other programming languages. It can be used in any function.

**“this” is not bound**

“this” is called at run time so it behaves differently than in other languages. It doesn’t depend on where the method was declared, but rather on what object is “before the dot”.

**Calling without an object: this == undefined**

**Arrow Functions have no “this”**

**Summary**

* Functions that are stored in object properties are called “methods”.
* Methods allow objects to “act” like object.doSomething().
* Methods can reference the object as this.

The value of this is defined at run-time.

* When a function is declared, it may use this, but that this has no value until the function is called.
* A function can be copied between objects.
* When a function is called in the “method” syntax: object.method(), the value of this during the call is object.

Please note that arrow functions are special: they have no this. When this is accessed inside an arrow function, it is taken from outside.

**JavaScript: Novice to Ninja**

**Chapter 5: Objects**

Objects can be used to encapsulate code that can be reused throughout the program

Objects can inherit properties form other objects

Objects are used to keep related information and functionality together in the same place

An object literal is an object that is created directly in the language by wrapping all its properties and methods in curly braces {}.

Object literals are a distinguishing feature of the JavaScript language, as they allow object ot be created quickly without the need for defining a class.

All objects are mutable at any time when the program is running. This means that its properties and methods can be changed or removed, and new properties and methods can be added to the object, even if it was declared using const. \*\*Why is this? Doesn’t this defeat the purpose of a const?

You can create an empty object like this: const spiderman = {};

\*not recommended though

\*object literal notation is the preferred way of creating objects

**Square brackets**

Dot notations is more common, but bracket notations can be used as well. Two advantages of bracket notation are, 1. It is the only way you can access properties with non-standard names (such as properties with spaces in the name), 2. You can evaluate an expression in the brackets and use that as the property.

JavaScript code can be placed inside brackets and the property key will be the return value of the code.

Can be used with the Symbol data type

const name = Symbol(‘name’);

const supergirl = {[name]: ‘Supergirl’}

The Symbol key [name] can be reused by other objects

\*Each symbol has a unique value, which means that using them as property keys avoids any naming clashes if you mistakenly use the same value for two different property keys.

**Calling Methods**

Can use dot or brackets notation to call a method

**Checking if properties exist**

You can use the in operator to see if a property exists within an object *or has been inherited*

Use the hasOwnProperty() method to see if the property is used from another object or belongs to the specified object

**Finding all the properties of an object**

Can use a for in loop. \*Really need to practice this to understand and remember it better

Object.keys() method returns an array of all the keys of an object

Object.values() method returns an array of all the values of an object

Object.values() method returns an array the objects key – value pairs

**Adding properties**

You can do this by assigning a property and a value like this:

superman.city = ‘Metropolis’;

\*this adds the property “city” and the value “Metropolis” to the “superman” object

**Removing properties**

delete superman.fly

Objects can contain other objects, aka nested

They are accessed the same as any other property, using dot or bracket notation (or a combination of the two)

Objects are copied **by Reference!!!**

Objects can be can be **passed as a parameter** to a function.

**Namespacing**

Naming collisions occur when the same variable or function name is used for different purposes by code sharing the same scope.

Solution to this: use the **object literal pattern** to create a namespace for groups of related functions.

This is done by placing all of our functions inside an object, thereby creating a namespace

And example would be to put the function square() inside and object myMath, then to call the function you would do this:

myMath.square()

**Built-in Objects**

**JSON**

parse()take a string of JSON and returns a JavaScript object

stringify()take a JavaScript object and returns a string of JSON data

**Math - objects**

**Math.PI** – The reatio of the circumference and diameter of a circle

**Math.SQRT2** – the square root of 2

**Math.SQRT1\_2** – the reciprocal of the square root of 2

**Math.E** – Euler’s constant

**Math.LN2** – The nature logarithm of 2

**Math.LN10** – The nature logarithm of 10

**Math.LOG2E** – Log base 2 of Euler’s constant

**Math.LOG10e** – Log base 10 of Euler’s constant

**Math – methods**

\*Look these up when you need to do math

**RegEx**

\*Look these up when you need them, too complicated to remember

Use these sites:

<https://www.regextester.com/>

<https://regex101.com/>

<https://www.regular-expressions.info/>

**Chapter 6: Document Object Model (a.k.a. DOM)**

Allows you to access elements of a web page and enable interaction with the page by adding and removing elements, changing the order, content and attributes of elements, including how they are styled.

DOM treats everything on a web page as a node**!**

The HTML tag is the root node.

JavaScript uses a special built-in object called document to modify parts of a web page

\*You did this in the last class…see those notes

Look into jQuery more…

**Creating An Element**

createElement() takes a tag name as a parameter and returns that element

\* See page 269 for the process

1. Create the element node

2. Create the text node

3. append the text node to the element node

You can create a function to make the process easier:

function createElement (tag, text) {

const el = document.createElement(tag);

return el;

}

To use the function to add Aquaman to a list:

const aquaman = createElement(‘li’, ‘Aquaman’);

**Remove Elements From a Page**

removeChild()

To use the function to add Aquaman to a list:

Heros.removeChild(aquaman);

**Live Collections** – Use is discouraged for performance reasons

**Chapter 7: Events**

Events are another part of DOM’s, they provide the link between user interaction and the web page

**Event Listener**

Document.body.addEventListener(“click”, doSomething);

When any part of the page is clicked on by the user, doSomething() is called

**Event Object**

**Coordinates of an Event**

You can find the position of the mouse when an event occurs in several ways

screenX & screen

clientX & clientY

pageX & pageY

**Types of Events**

click

mousedown and mouseup – These occur before click

dblclick

\* Be careful about using click and dblclick on the same element, a dblclick will always cause a click event also

mouseover

**Keyboard Events**

keydown – will continue to occur if the key is held down

keyup

keypress – only occurs for keys that produce character input & the ‘Delete’ key. It is the most reliable way to find out the character that was pressed on the keyboard

**Modifier Keys**

shiftKey

ctrlKey

altKey

metaKey

**Touch Events**

touchstart – Be careful using this one, it fires as soon as the user touches the screen, the click event is often safer

touchend – occurs when the user stops touching the screen

touchmove – occurs when user moves after touching the screen but hasn’t lifted from the screen

touchenter – occurs when user has touched screen and passes over the element with the listener attached

touchleave – occurs when user is still touching the surface but leaves the element to which event listener is attached

touchcancel – occurs when a touch event is interrupted such as finger leaving the screen or too many fingers being used at once

**Touch Events Properties**

events.touches.length

events.touches[0] – first touch object

events.touches[1] – second touch object

touch.identifier – is a unique ID to make sure you are dealing with the same touch