Chapter 3 – Getting Started with JavaScript

Contents

[JavaScript!!!!!! 2](#_Toc455513748)

[JavaScript Values 2](#_Toc455513749)

[JavaScript Number Type! 2](#_Toc455513750)

[JavaScript Strings!!! 2](#_Toc455513751)

[JavaScript Conditions!!! 3](#_Toc455513752)

[Boolean expressions 3](#_Toc455513753)

[Short-circuiting 3](#_Toc455513754)

[isNan 3](#_Toc455513755)

[If Statement 3](#_Toc455513756)

[Switch statement 4](#_Toc455513757)

[Test if a variable has a value 4](#_Toc455513758)

[Evaluating equality 5](#_Toc455513759)

[Rules for naming variables 5](#_Toc455513760)

[JavaScript functions!!!!! 5](#_Toc455513761)

[Declaration of a Function 5](#_Toc455513762)

[Storing a function is a variable 5](#_Toc455513763)

[Special notes about calling JavaScript functions 6](#_Toc455513764)

[JavaScript Pop-ups! 6](#_Toc455513765)

[JavaScript variable scope 7](#_Toc455513766)

[Converting between types 8](#_Toc455513767)

[JavaScript loooooooooops!! 8](#_Toc455513768)

[While loop 8](#_Toc455513769)

[Do While loop 9](#_Toc455513770)

[For loop 9](#_Toc455513771)

[Handling Errors: Try Catch Finally 9](#_Toc455513772)

# JavaScript!!!!!!

So I hear you want to learn about JavaScript. Well your in luck because I am going to describe everything you need to know about this amazing language. Your welcome! Enjoy the contents below!!!!!!!!!!!!

## JavaScript Values

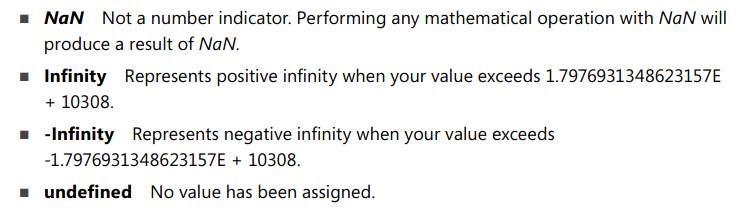
Yeah so JavaScript thinks its super cool because it has these things called values. They aren’t so special! Well actually they are quite special. Basically every variable in JavaScript can equal one of the following values:

* **Object:** an object is an object is an object.. it’s a just a normal object..
* **Primitives:** int, short, long, Boolean, string, char, etc.
* **Function:** you know, like those things you call in your program to do things.

**You can determine the value of a variable by using the typeof(variableName) function**

## JavaScript Number Type!

The number type is a number, obviously. You can represent a number in various ways. You can have a plain number like 1 or 2. But that’s boring. Here are some special ways to represent a number.



## JavaScript Strings!!!

Strings, strings, the magical yarn, the more you create, the larger sweater you will have when your grandma knits it!! Actually these aren’t strings that your grandma can knit. They are actually computer bits that you string together to make sentences!!! See what I did there? Here are some fun things you can do with strings!

* **Declaring a string can be done with single quotes or double quotes**
  + ‘this is a valid string’
  + “this is a valid string”
* **You can place single quotes inside of a double quotes string and vice versa**
  + ‘this “is” a valid string’
  + “this ‘is’ a valid string”
* **Escaping quotes!**
  + “this \”is\” a valid string”
  + ‘this \’ is \’ a valid string’
* **Concatenating strings!**
  + “this” + ‘is’ + ‘a’ + “valid” + ‘string’

## JavaScript Conditions!!!

That person lied?? Oh they were actually telling the truth?? Well make up your mind already!! Just use Booleans to represent whether what’s being said is true or false. It’s easier than this back and forth drama.

### Boolean expressions

* **And** is “&&”
* **Or** is “||”
* **Not** is “!”
* **Math expressions:** <=, >=, >, <

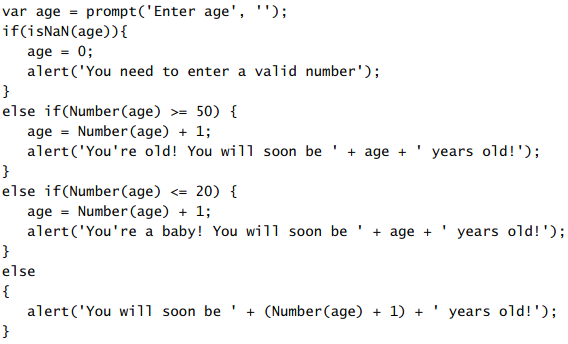
### Short-circuiting

JavaScript evaluates the left side of an expression before the right side. In cases where you have something like “false and true”, the left side evaluates to false, so the right side won’t even get read by the compiler.

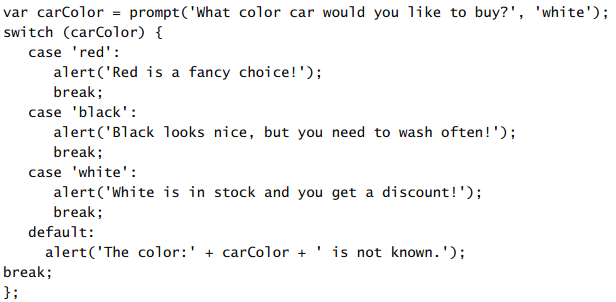
### isNan

**isNan(object)** function can be used to see if an object is a number

### If Statement

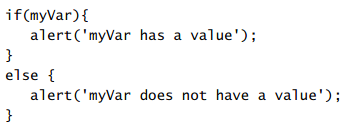


### Switch statement

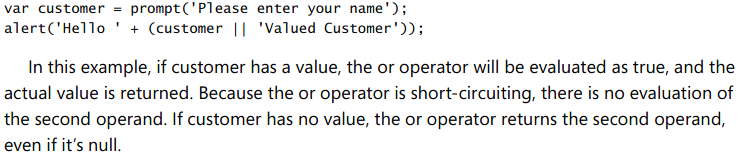


### Test if a variable has a value

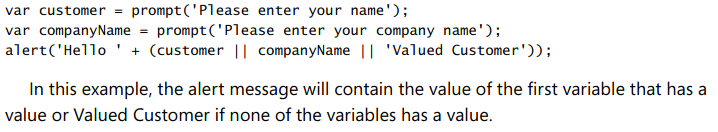
* **Using an if statement**



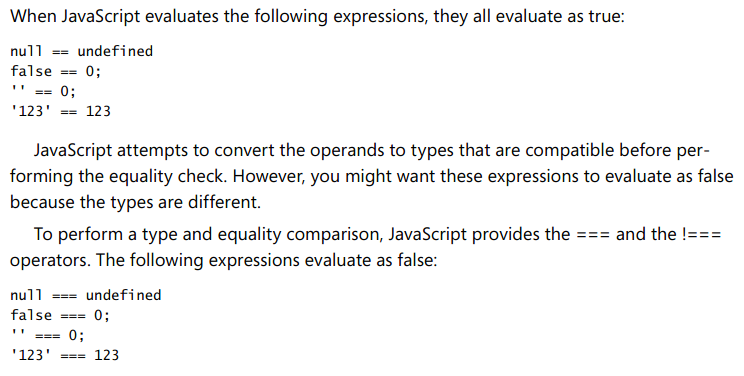
* **Coalesce operator**



* **Chaining coalesce operators**



### Evaluating equality



## Rules for naming variables

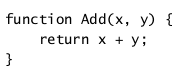
Yeah JavaScript complains if you name your variables badly. So you need to make sure you conform to the following naming convention:

* Cannot begin with numbers
* Cannot contain math or logical operators (-, +, \*, etc)
* No punctuation marks (!, #, @, &, etc.)
* \_ and $ are allowed
* No spaces
* Cannot be a JavaScript keyword
* Camel case!

## JavaScript functions!!!!!

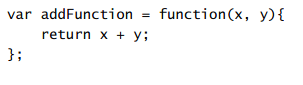
JavaScript functions are pretty great! And there’s some things that are worth knowing about them! Thankfully for you I will tell you everything you might want to know about them!!!!

### Declaration of a Function



### Storing a function is a variable

The cool thing about this is that the variable is now of type function! So if you said typeof(addFunction), then it would return Function.



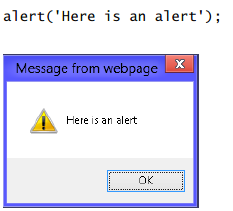
### Special notes about calling JavaScript functions

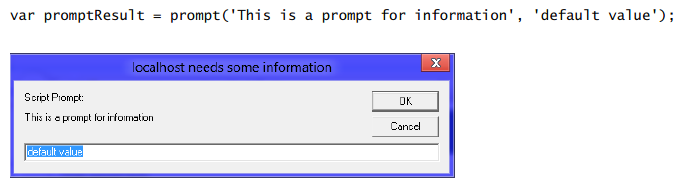
* If you call a JavaScript function with too many arguments, then JavaScript will discard the extras.
* If you call a JavaScript function with too few arguments, then the missing arguments will be passed as undefined.

## JavaScript Pop-ups!

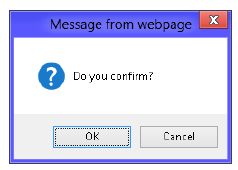
So JavaScript has these things built into it that everyone despises. You know what I’m talking about! POPUPS!!!!! The bane of most people’s existence and the reason that pop-up blockers are built into every modern browser! If you feel sadistic and want to try to code one of these then keep reading:

1. **ALERT:** returns nothing



1. **PROMPT** : returns a string
2. **CONFIRM:** Returns a boolean

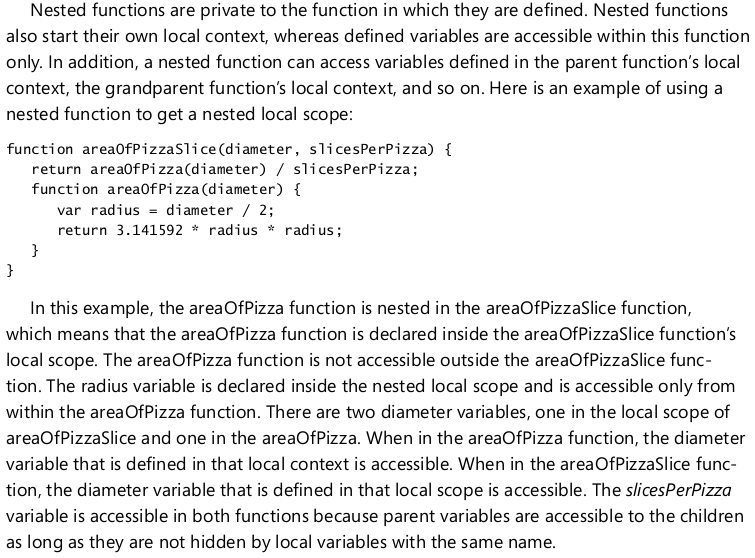




## JavaScript variable scope

I recommend that brushing your teeth that you use scope mouthwash. It’s a good call! It’ll leave your breath smelling minty clean! Oh, you didn’t want to hear about scope? What then? Oh variables. Well Okay. Here is what you need to know:

* **There are two scopes: global and local.**
* **Local scope** means that variables declare in a function are only accessible within that function.
  + Declaring variables in local scope requires using the var keyword.
    - E.g. var variable = 5;
* **Global scope** means that variables declared are accessible from anywhere.
  + Declaring variables in global scope requires not using any keyword.
    - E.g. variable = 5;
* **Scope of nested functions:**



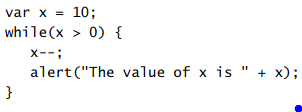
## Converting between types

You can convert between different types! Yeah nothing special here. We all know that’s possible in every other language. So why not this one too?

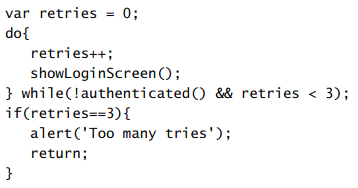
* Convert to a number using the function Number(string)
* Convert to a string using the function String(object)

## JavaScript loooooooooops!!

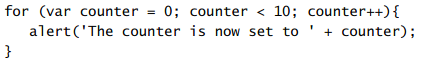
### While loop



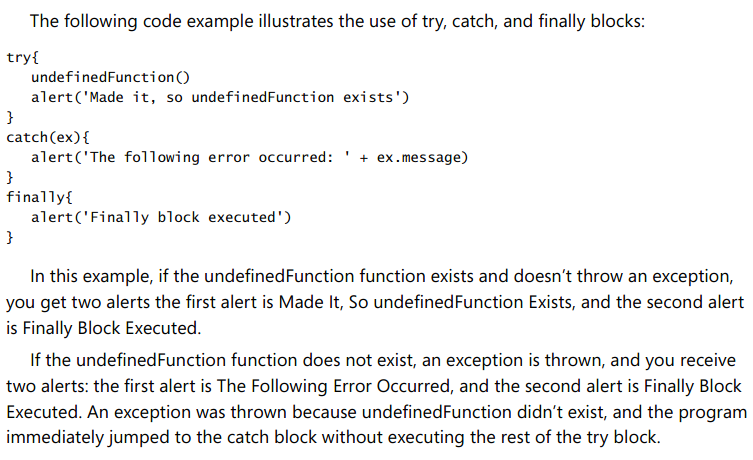
### Do While loop



### For loop



## Handling Errors: Try Catch Finally

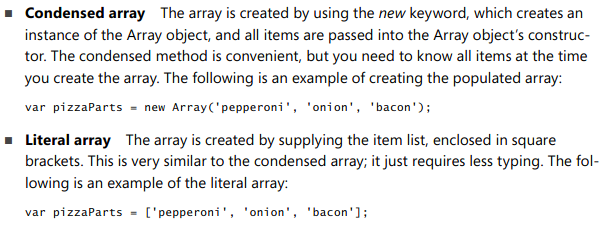


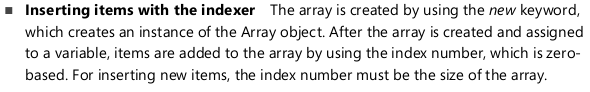
## JavaScript Arrays!!!!

So JavaScript and every other language on earth has these things called arrays. Why do we need them? I don’t know. You act like I’m the expert on this topic or something. I’m supposed to be? Well, okay. I’ll feign experience for your sake then! We need them because they store data in a unique way!!! You can learn about them in the sections below.

### Create an Array

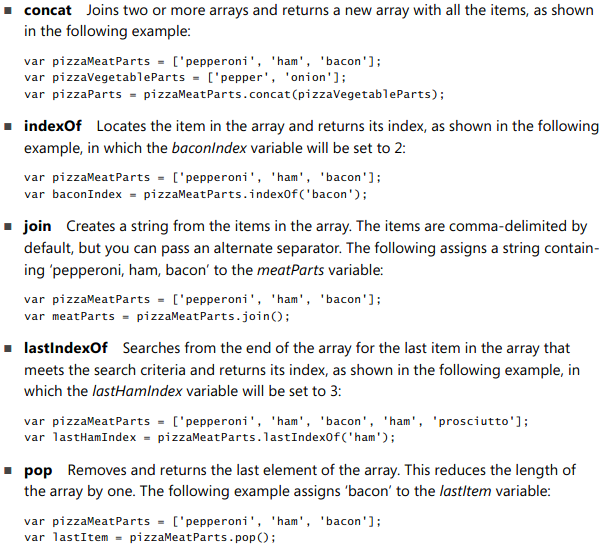
There are three ways to create an array. They are listed below:

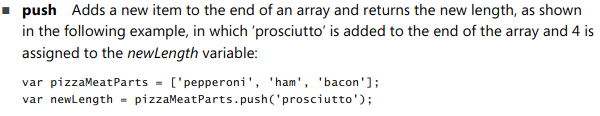


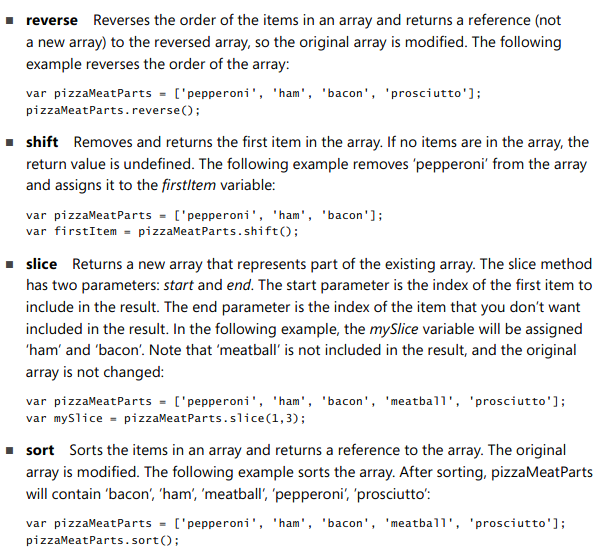


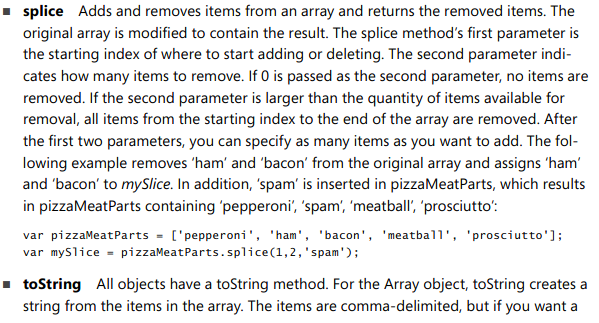


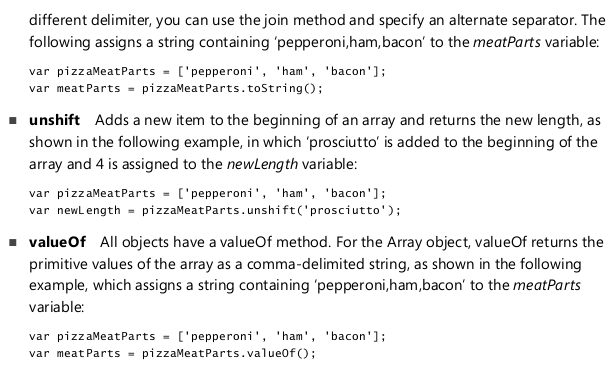
### Array Methods



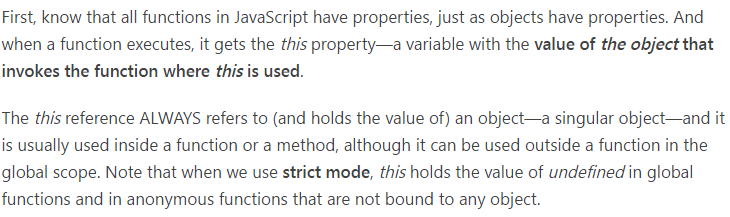






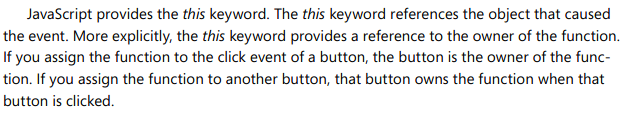


## Using “this” keyword



A good blog about how to “this” works: <http://javascriptissexy.com/understand-javascripts-this-with-clarity-and-master-it/>

The example below refers to how the “this” keyword works in an event:

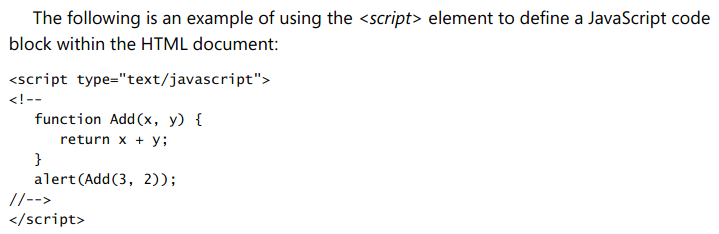


# The script tag!!!

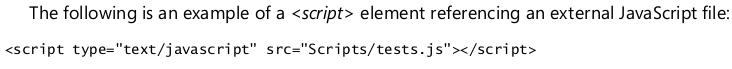
The script tag is awesome and I’ll show you why!

## Using the script tag to define inline javascript

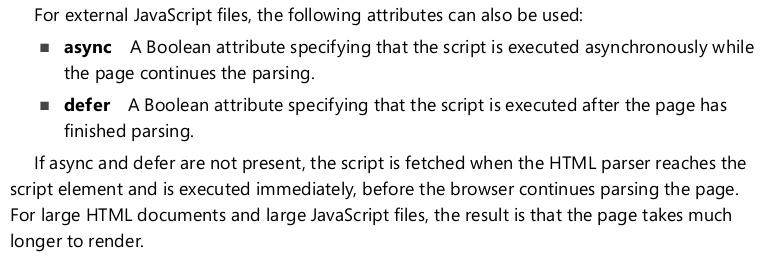
You can use the script tag to define inline javascript.



## Using the script tag to reference an external javascript file



* The script tag must be empty



## Handling browsers that don’t support JavaScript



## Where in HTML to place your Script tags

You can maximize performance by placing your script tags in the proper place.

* Place your script tag at the end of your HTML document, so that the browser doesn’t stop parsing HTML to import your JavaScript.
* Place your script tag after your CSS reference, so that the browser can render your JavaScript and CSS at the same time.

# Accessing DOM Objects

The DOM is a hierarchy of elements on an HTML page. You already learned about HTML right? Well basically the DOM is all of your HTML elements rendered on a page in the hierarchy that you defined in your HTML file.

## Static vs Live Node Lists

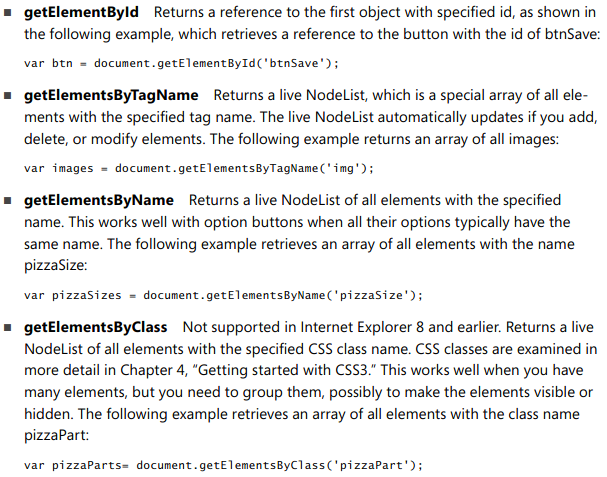
When accessing DOM objects, you can have a list of nodes returned from the DOM. Your search method indicates whether a static of live node list is returned. A description of static and live node lists is as follows:

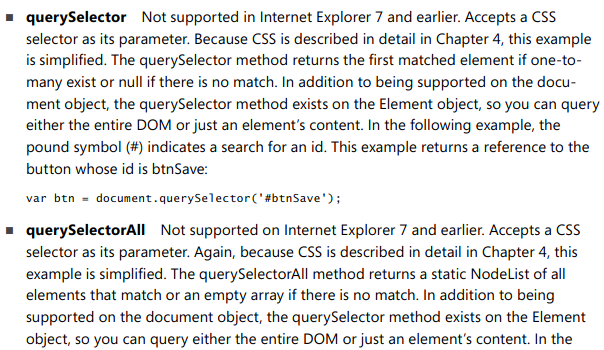
**Static:** A list of nodes that doesn’t change.

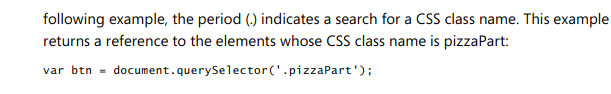
**Live:** A list of nodes that does change.

There is a performance hit for using a search method that returns a static node list because it takes longer for the DOM to return a list of nodes that will never change. Therefore, best practice is to use search methods that return a live node list.

## Methods to Access the DOM



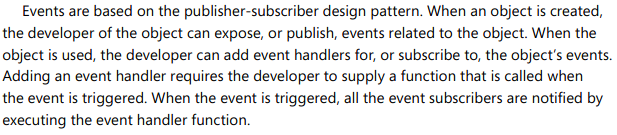




# Events

We all love events, right? They are a ton of fun, you can drink and meet new people! It’s a suuuper exciting thing to happen! Well, DOM events are the same! They are fun too! Events let you make your web pages interactive!!!! I’ll show you why they are such great things in the sections below.

## Publish- Subscribe Pattern



## Event Propagation

### Event Capturing and Event Bubbling

**The situation**

You have a button nested in a hyperlink tag. The button and hyperlink subscribe to the onclick event.

<html>

<body>

<a>

<input type=button onclick=”DoSomething”>Save</input>

</a>

</body>

</html>

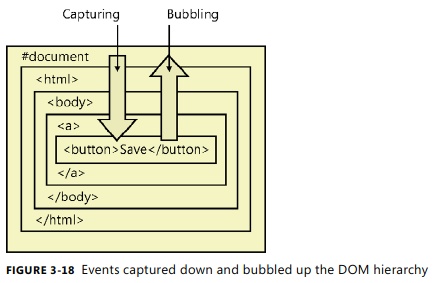
**The problem**

Someone clicks the button and events are triggered. What order are the events triggered in?

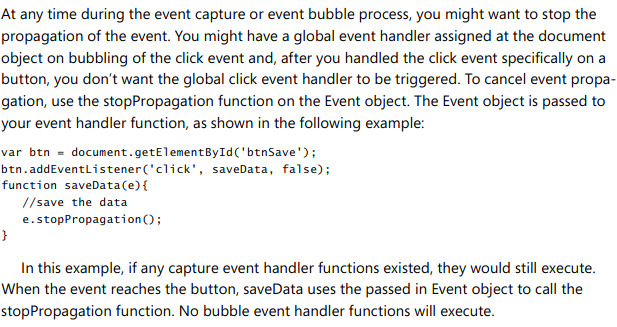
**The solution**

The DOM handles this situation by using event capturing and event bubbling. The order of events is as follows:

1. The button element triggers the on click event.
2. Event capturing starts
3. The DOM captures the onclick event and passes it to the HTML element
4. The HTML element captures the on click event and passes it to the Body element
5. The Body element captures the on click event and passes it to the Hyperlink element
6. The Hyperlink element captures the on click event and passes it to the Button element.
7. The Button element captures the on click event. The event is not passed anymore because this is the leaf node of the document.
8. Event capturing ends.
9. Event bubbling starts.
10. The button element executes it’s on click event if it exists.
11. The Hyperlink element executes it’s on click event if it exists.
12. The body element executes it’s on click event if it exists.
13. The HTML element executes it’s on click event if it exists.
14. Event bubbling ends.



### Cancel Event Propagation

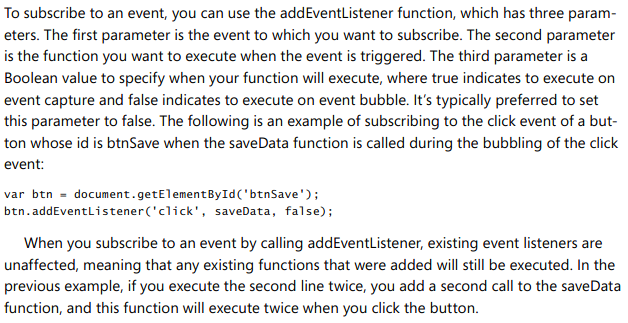


## Subscribe to an event

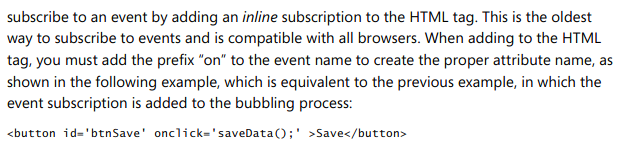
There are multiple ways ot subscribe to an event. They are as follows:

### AddEventListener

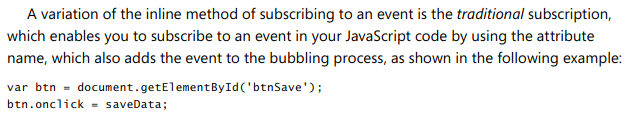
This is best practice because it is the W3C specified way to subscribe to events.



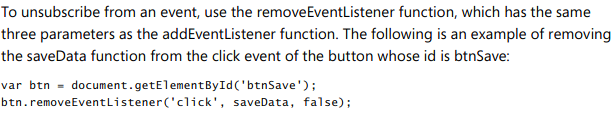
### In-line using HTML



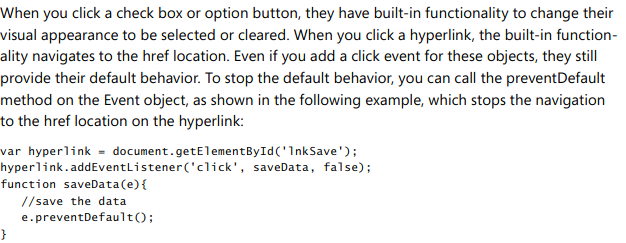
### In-line using JavasScript



## Unsubscribe from an Event

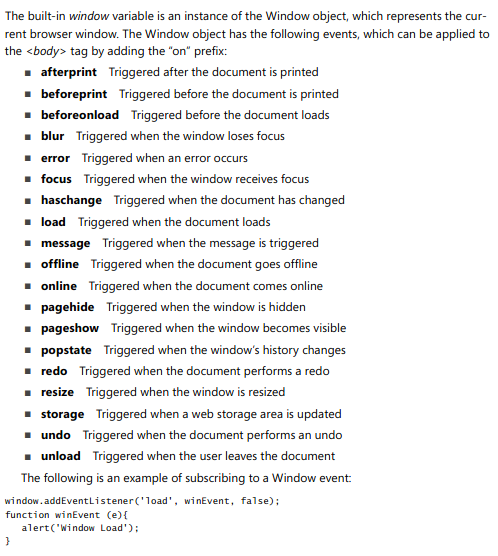


## Prevent Default Event

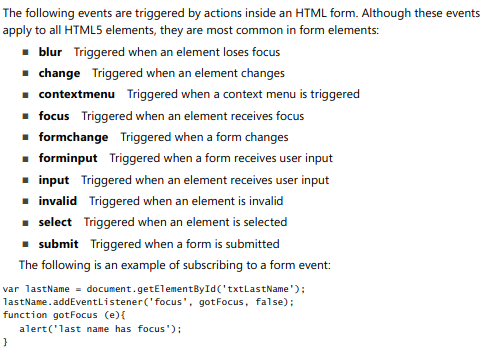


## Event Reference

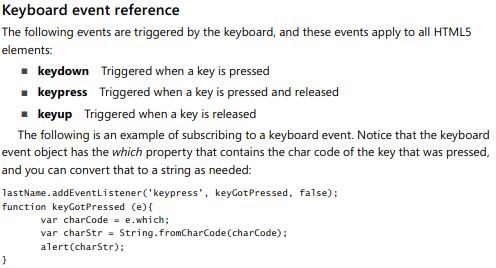
### Window Events



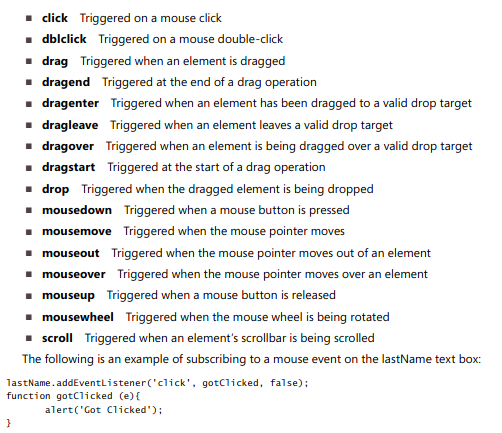
### Form Events



### Keyboard Events



### Mouse Events



### Media Events

