**Javascript (thenewboston tutorial)**

**Introduction:**

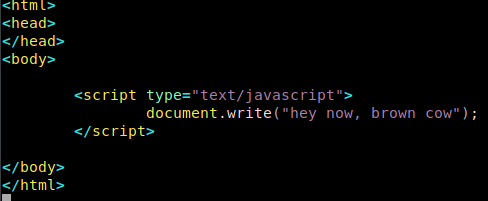
Javascript is a scripting langauge to make webpages INTERACTIVE.

Before we begin you need a few things first. You need a text editor. Any will work fine, but notepad++ or something similar is recommended, but vi is fine.

You’ll need to know some basic webpage stuff, know your html ahead of time.

Just like everything else in xhtml you need to add it through the use of tags.

We need to tell our webpage that we’ll be using javascript before we actually use it:

 This is very basic javascript. Use this as a test. Document.write is indeed javascript and this will type to your html. It returns something when your html gets rendered in your browser.

Older browsers dont’ use javascript but you can try and make sure that odler browsers never attempt to use your javascript, but we won’t need that for now. This lesson is from 2011 after all.

**Comments and Statements:**

Sometimes you can leave comments to yourself in html! It’s a line of text/code that your browser ignores completely.

It’s basically just like comments in regular code. A sticky-note! A line that your browser ignores just so the programmer can follow along easier. You do comments with “//”

*//nothing happens here*

What about a nice longgg comment? Use /\*

*/\**

*this is still a comment*

*also a comment*

*still nothign*

*\*/*

Now a computer program, at its core, will just follow instructions. A script. It just goes one by one in order and does what you tell it EXACTLY. Each instruction is called a STATEMENT. Every statement in javascript ends in a semicolon.

*document.write(“i love ham”);*

The above just prints out “i love ham”

*document.write(“i love ham”);*

*document.write(“i love bacon”);*

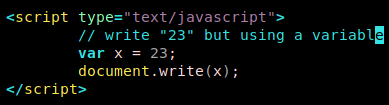
Will go one by one. These are statements. Each statement goes on until it finds a semicolon in javascript.

Simple, but keep the concept in mind alright? Next subject is a little longer.

**Variables in Javascript:**

Remember variables in algebra? A variable is something (like the letter X set to “ham” and set it equal to something else. Now “I love X” means “I love ham”.

That’s what we’re doing here:



That is what a variable is, how to make it, and how it’s useful.

You can name variables ALMOST anything you want, but they are case-sensitive. In this example, capital-X won’t work.

You also should not be overwriting built-in javascript things! do not make a variable named “var” that’s bad practice.

You also need to start your variable name with a letter or underscore. You cannot start a variable with a symbol. \_bucky is fine, bucky is fine, &bucky is NOT fine.

Lastly, the hardest thing to remember is that your variables can’t start with numbers:

*var 6bucky = 32;*

*// won’t work out*

**Different Types of Variables:**

Javascript has many types of variables like most programming languages.

The first type deals with numbers:

//variable for number

*var tuna = 20;*

*document.write(tuna);*

And we can see this variable on our webpage. But you can also set numbers with decimal points equal to variables!

*//floating numbers*

*var tuna = 20.243;*

*document.write(tuna);*

Floats and ints are similar to python where it’s just ASSUMED. Now let’s move onto the next thing which is TEXT.

TEXT is called strings much like most programming languages. You also use ‘var’ for this!

*//strings*

*var tuna = “strings are in double quotes”;*

*document.write(tuna);*

Viola! Strings are incredibly easy and similar to python in this sense.

How would we put double quotes in a string though..? It doesn’t work at all because we can’t put quotes in something defined with quotes! We need to use an ESCAPE CHARACTER.

*//strings with double quotes!*

*var tuna = “john said, \”boy do I love dogs\” “*

*document.write(tuna);*

“\” is the escape character. If you literally mean double quotes, use an escape character. You’ll see many cases where we’ll need these.

So we learned numbers and text (strings). There are two other types of variables we need to know now.

Booleans are true and false – lowercased in javascript.

*//boolean*

*var tuna = false;*

*//other boolean*

*var salmon = true;*

The fourth type of variable is called null. Null has no value at all.

*//null*

*var tuna = null;*

This isn’t ZERO it’s just an empty variable. You’ll see it later on just know that it exists at this time. Just remember you can use numbers. Let’s move onto the next tutorial.

**Using Variables with Strings of Text:**

So suppose we have a variable and we want to throw it IN a string? Let’s make two variables:

*var name = “mike”;*

*var age = 25;*

Now how would we write “mike is 25” with variables..? With the addition sign!!

*document.write(name + “ is “ + age “ years old!”);*

This outputs: mike is 25 years old

Only use the quotes for strings you’re declaring right there. Variables should never be within quotes themselves. Here we only use it for “ is “ and “ years old”

You can also stick them alongside each other like:

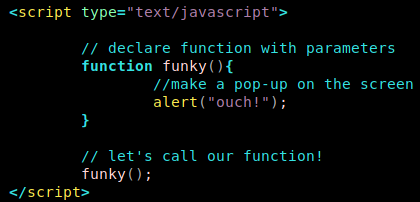
*document.write(name+age);*

Which outputs “mike25” - got it? Super simple. Javascript is not difficult if you already understand basic programming, but don’t get comfy. We’re going to dive a little bit deeper into things. Just hang in there.

**Functions:**

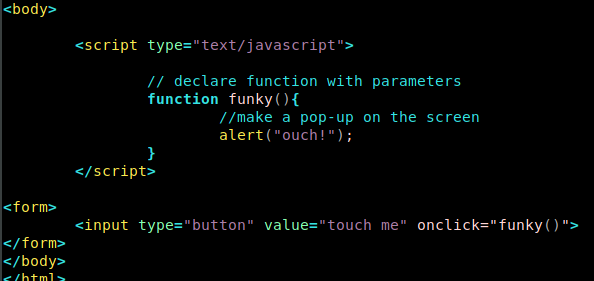
A function is like a mini-program that you can re-use whenever you want in your javascript. Never do the same thing twice, remember.

EVERYTHING can be a function and you should break it down as much as possible. Let’s make one:



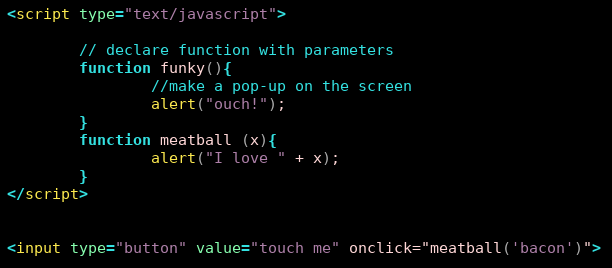
Check it out! Pretty cool huh? We also learn **alert** here which makes **a popup message –** we can use funky as many times as we want and wherever we want in our javascript code while trying to make an interactive experience for the user!

Let’s use a form in html and make a button that will make this popup message!!! We can use our javascript function in the ‘onclick’ section of things like buttons!

 Pretty amazing right? The HTML at the bottom allows you to create a button (remember this?) and we can use javascript declared functions in the ‘onclick’ section even outside of script tags! That’s pretty amazing. It makes a button and when we click it it will run out “funky” alert function!

**Parameters and Functions:**

What are these parenthesis and why do we need them?

 Sometimes you need additional information for your function to work! Maybe you just want to display “ALERT” or maybe you need some info about WHAT TO ALERT ABOUT? Your function should be able to accept these PARAMETERS.

This works a lot like python:

See? It ends up looking and working just like python. No need to declare what ‘x’ is in the parameters field.

Tested and working! Now re-use it with a bunch of different parameters and it’s working.

You can have it generate multiple dialogs for a single button click over and over again! Isn’t that awesome?

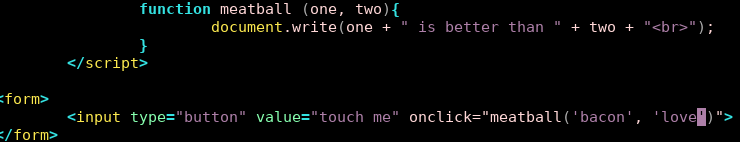
This is a really simple example using an alter box. Parameters are very easy. Let’s see what happens if our function needs many pieces of information of different types!

**Passing Multiple Parameters into a function:**

In this tutorial we’re going to learn how to use many parameters for a single function.

Just give it TWO parameters inside of your parenthesis! Separate them by commas.

FUN FACT IN THE MIDDLE. HTML tags are treated like html! If you write out <br> it’ll treat it as a line-break! Check it out:

 See that? Very simple. This will print out onto the page:

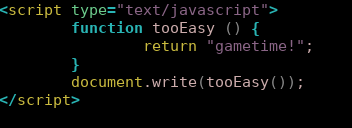
“bacon is better than love”

-...a very true statement, but also an example of how easy it is to pass in multiple variables into a function and use them in a string like we learned earlier (using +) !

**The return statement:**

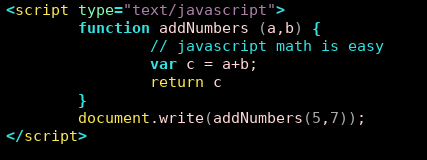
Sometimes we’ll make a function and it’ll just be a DOER function. It prints, it makes a popup, it DOES something. But sometimes we want the function to return a value to us. This is very useful when we want an answer returned to our main program/webpage!

This works exactly like it does in python.



This function will display “gametime!” in text on our webpage! “Now let’s do a realworld example….

Here’s an example where we have a fucntion that will add whatever numbers you give it! You can use it as many times as you’d like in your function!



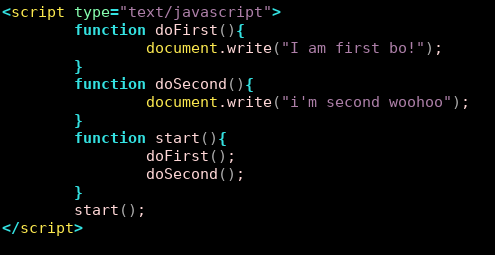
Pretty awesome huh? This renders “12” to the screen in plain text.

That is the basics of return value, and hey, there’s javascript math right there for you! Nice and easy, huh?

We’re spending a lot of time on functions because that’s where the majority of the power of javascript lies where plain html tends to fail.

**Functions from within functions:**

Say we have two functions that print out “do first” and “do second”. Now we can make a “start” function that’ll call our other functions!



Notice what’s happening here. This will print the functions in order BUT we’re only ever calling the start() function from our main! This gives you an idea of how scope works in javascript. There IS a global scope even if you don’t think that there is. If you have a lot of code you can organize some in the doFirst and some in the doSecond. This makes it much easier to debug and is highly recommended.

Make your own “main” in javascript WHENEVER possible.

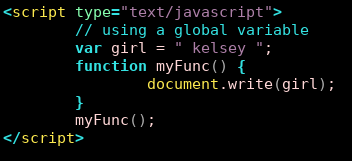
Be careful, you can do endless loops of programming by nesting functions within each other. Be smart about it, and USE THE ‘MAIN’ METHOD as javascript doesn’t reallyyy give you a ‘main’ necessarily.

**Global and Local Variables:**

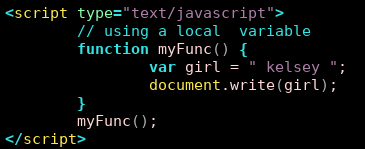
We’re finally going over scope! Yay!

A global variable is OUTSIDE Of any function. A local function only exists inside the body of a function! Simple enough, but follow the rules carefully.

HERE IS A FUNCTION USING A GLOBAL VARIABLE:



HERE IS THE SAME FUNCTION USING LOCAL VARIABLES:



If you don’t NEED the ‘girl’ variable outside of these functions then it should definitely only exist inside the function. Both of these functions WORK, but one is less of a waste of memory. No need to have ‘kelsey’ floating around in a variable while you don’t care about it and will not need it later.

**Math Operators:**

Math operators are easy in javascript. You know you can use ‘var’ to set variables for numbers and it will figure out whether or not you want an int or a float. Let’s learn some extra fun stuff with it.

v*ar apples = 4 + 76;*

*document.write(apples);*

The above will print out “80”.

*// 6*

*var apples = 10 – 4;*

*// 40*

*var apples = 10 \* 4;*

*// 2.5*

*var apples = 10 / 4.0;*

*// ALSO 2.5*

*var apples = 10 / 4;*

*// want the remainder? Modulus works here!*

*Var apples = 10 % 4; // returns “2”*

Not that bad right? It works almost exactly like python does!

Increment and decrement operators also work in javascript.

*// increment example*

*chops = 7;*

*chops = chops + 1; // adds one*

*// the same as…..*

*chops++; // increments by one!*

*// chops is now increased by 1!*

*// decrementing:*

*chops = 2;*

*chops--; // chops is now ‘1’*

Incrementing is a nice shortcut that will be useful in for-loops later-on.

**Assignment Operators:**

This is going to be a really quick and easy one. It’s all about “=” which we should understand very well by now.

*// assign 24 to bucky*

v*ar bucky = 24;*

There’s shortcuts too though! There’s add-and-assign!

*// add and assign 50 to the current value ‘24’ of bucky*

*bucky += 54; // now, 78*

*// and others*

*bucky -= 54;*

*bucky \*= 54;*

*bucky /= 54;*

*bucky %= 54;*

You could have functioned without these butttt these shortcuts are very useful and will make your code more-readable to someone going through as well as use less text+lines for you.

**If Statements:**

In this tutorial we’re going to learn how to do conditionals in javascript and comparison operators.

It’s a simple way to have your computer make a decision. If the scenario is true, it’ll run a bit of code. Concept is similar to other languages.

*var apples = 34;*

*var hotdogs = 53;*

*if (apples==hotdogs){*

*document.write(“yay it worked”);*

*}*

*...*and that’s it! It’s the same as C++ and C# really. Similar to most languages, you need to use “==” as “=” is an asigment operator.

What the opposite without using an else? Do:

*if(apples!=hotdogs){ }*

Math operators work fine here as well.

*if(53>99){ }*

Verify that it works in our sample html file...

***If/Else Statements:***

This one is easy.

If the conditional is false, instead of doing nothing we’re just giving it ANOTHER option! This is done through ‘else’.

*if(condition){*

*// stuff to run if test is true ;*

*}else {*

*// stuff to run if test is true;*

*}*

That’s it! After your if immediately include an else with its own brackets for its own block of code! That’s this entire section. If you understand it in other languages you understand it here. Now you’re ready for some javascript nesting.

**Nesting and Fridays:**

Now we can have an if-statement INSIDE of an if statement! We can stack conditions and get MORE SPECIFIC in determining the block of code we’re about to run.

*var firstname = “mike”;*

*var lastname = “paps”;*

*if(firstname==”mike”){*

*if(lastname==”paps”){*

*document.write(“hello master paps!”);*

*}*

*else{*

*document.write(“you’re a mike, but not MY mike...”);*

*}*

*} else {*

*document.write(“who on earth are you?!”);*

*}*

So now our code has to pass TWO tests instead of just one test before it reaches our final bit of code. Get it? You can put those else’s ANYWHERE along the line of nesting these. Nests can get very deep, so pay attention while you’re doing it!

**Complex Conditions:**

In the last tutorial we embedded if’s in other if’s and through else’s in there. Functionally it went great, but it really doesn’t FEEL like good programming does it? There’s an easier way to test more than one condition!

PAY ATTENTION this is a little different from python:

Complex conditions can do multiple conditions in ONE LINE:

*var firstname=”mike”;*

*var lastname=”paps”;*

*if( (firstname==”mike”) && (lastname==”roberts”) ) {*

*document.write(“hello master paps!”);*

*}*

Did you see that? The AND is && and you connect conditions that can be true/false inside paraenthesis!

Basically you say if( (true/false) && (true/false) ) {

document.write(“both are true”);

}

Got it? You can tag on AS MANY AS YOU WANT.

What if you want to only have ONE condition to be true to execute code? (“OR” in python think).

*if( (firstname==”mike”) || (lastname==”paps”) ) {*

*document.write(“hello mike and/or master paps!”);*

*}*

Only one of those conditions need to be true! You can combine AND and OR to make whichever complex conditionals you want! ( && vs || ).

**Javascript Switches:**

Finally getting into some things that are unique about javascript-ish.

This is kind of making custom-responses.

*var girl = ‘natalie’;*

*if(girl==’natalie’){*

*// do stuff*

*}*

*if(girl ==’emily’){*

*//do different stuff*

*}*

There’s a much easier way to do this. A switch statement takes one variable and test them for all of the possible scenarios. It’ll function the same but it’s more-organized and a better way to accomplish this.

*switch(girl){*

*case “natalie”: // yes that’s a colon*

*document.write(“natalieeee”);*

*break;*

*case “emily”:*

*document.write(“emily!”);*

*break;*

*case “ashley”:*

*document.write(“no one has ever met a nice ashley”);*

*break;*

*}*

Look how easy that was! You can do this forever and it’s so much more-organized than a series of if’s and else’s!

Want to put a catch-all in your switch? You’ll use DEFAULT as a regular case block!

*switch(girl){*

*case ‘natalie’:*

*document.write(“natalie”);*

*break;*

*default:*

*document.write(“no clue who “ + girl + “ is.”);*

*}*

Switches are incredibly easy and you should use them anytime that you’d string a bunch of if’s and else if’s together trying to find various options and treat them differently.

**Javascript for-loops:**

Loops are extremely useful in every programming language because it’s purpose is to save time and eliminate repitition.

Say we wanted to do:

*document.write(“i want to write this ten times”);*

-...ten times. Wouldn’t it be cool if we could repeat it over and over again (you know where this is going….). It’s set up in a VERY specific way. It takes three statements to work.

It’s for(make a number variable; end if this is false; what to do to that number variable each time) { }

*for(x=0; x<10; x++ ){*

*document.write(“we’re looping this over and over again!”);*

*}*

The above for-loop will start at zero and go all the way until X isn’t less than ten anymore (so it runs ten times!

Tested in-browser and it’s working.

**While-loops in javascript:**

So that was for-loops, but what about while-loops?

A while loop is similar to a for-loop. It’ll do a bit of code over and over until a condition happens and it finally stops.

Let’s go ahead and make a counter-variable and do a while-loop over it similar to our for-loop.

*while(x<10) {*

*document.write(“x is still less than 10”);*

*x++;*

*}*

That’s very similar to the for-loop we made above. While-loops are fun and simple because it’s easier to do things that aren’t necessarily integer-counting.

Be very careful to avoid infinite loops!

**Do-while loops:**

The do-loop. Another syntax of a while-loop EXCEPT it will run the first time no matter what.

*var x=0;*

*do{*

*document.write(“hellooo”);*

*x++;*

*} while(x<10);*

This is exactly the same as the while-loop above. Not really much of a functionality change, just a syntax thing EXCEPT it will run the code the first time no matter what.

**Event-Handlers:**

Now we’re really getting into it!

Event-handlers look like regular html from a distance. Let’s show one in a demo.

One place that you can insert them are in form-elements, specifically form-buttons!

*<form>*

*<input type=”button” value=”touch me!” />*

*</form>*

This, in html, puts a button on the screen. Let’s insert event-handlers to spirce it up a bit.

These can exist OUTSIDE of <script> tags because the browser will automatically know thanks to the keywords.

The most commonly-used event-handlers is onClick (we’ve seen this in android). Onclick in a button definition will execute javascript code!

*<form>*

*<input type=”button” value=”touch me!” onClick=”alert(‘caterpillar!’)” />*

*</form>*

Tested and working. You can also define full functions in <script> tags and have them execute in the onClick section!

You can keep going too:

*<form>*

*<input type=”button” value=”touch me!” onClick=”alert(‘caterpillar!’);*

*alert(‘ahhh’);” />*

*</form>*

**onMouseOver and onLoad:**

Some more amazing event-handlers to enhance your html page!

There are tons of these and you can place them in different locations (links, places you’d never even think of).

onMouseOver gets called whenever you mouse-over (hover over) the object!

Let’s try this with a link in html:

*<a href=”*[*http://thenewboston.com*](http://thenewboston.com/)*” onMouseOver=”alert(‘its workingg’);”>*Hover over me!</a>

It works!

onMouseOut is the same only it works when you take the mouse OFF of it!

*<a href=”*[*http://thenewboston.com*](http://thenewboston.com/)*” onMouseOut=”alert(‘its backkk’);”>*Hover over me!</a>

You can add event-handlers into the body-tag itself believe it or not! Add some random text in there first before we start.

*<body onLoad=”alert(‘it loaded the webpage’);”>*

*this is some random text*

*</body>*

It worked perfectly in-browser. Alert-boxes aren’t useful examples but you’re getting the idea!

What about on the way out? Like when the user exits your webpage? that’s onUnload:

*<body onUnload=”alert(‘goodbye user!’);”>*

*this is some random text*

*</body>*

You can stick these event-handlers just about anywhere! In an upcoming video we’ll see some of the html places that you can put these event-handlers to make more-powerful webpages.

**Objects:**

Now is the time to learn about object oriented programming in javascript!

In this tutorial we’re going to go over OOP, what they are, and some built-in objects in javascript before we get to making our own.

It’s easily misunderstood – so focus very carefully here. It kind of throws a twist in here. Everything was set up out there before but now we’re venturing off into making our own data types etc. Just stick with the videos and focus.

An object is a piece of data that has its own properties and methods.

Properties are things that are like atttributes. Height, weight, name, age, etc would all be properties.

Methods are like functions that it can do. Jump, drive, speak, would all be methods.

*var tuna = “hey I am a tuna fish”; // string object with its own properties*

To access the properties and methods we’d need to add a DOT-SEPERATOR “.”

*document.write(tuna.length); // prints the length of the string “tuna”*

See? Accessing properties is as easy as that. Oh hey look at that! “document” is an object and “write” is a good example of a method belonging to it! *document.write()* is calling the write method for the document object!

So now you’re saying “okay that’s interesting but why do I care?” - well in the next tutorial we’ll create objects of our own. These objects of our own will take our javascript to the next level.

**Creating our own JS objects:**

Remember an object is a type of data that has its own objects and methods. There’s a few ways to do this. You need to use something called a CONSTRUCTOR function.

Normally when objects are being defined we’ll put the javascript tags in the <head> tags of the html script. Just a normal convention. It almost looks like making a function!

“This” is a keyword similar to “self” in python:

// here is our constructor function

*<head>*

*<script type=”text/javascript”>*

*// person object has ‘name’ and ‘age’ properties*

*function person (name, age) {*

*this.name = name;*

*this.age = age;*

*}*

*</script>*

*</head>*

So there’s our blueprints! Now let’s make an actual object based off of those blueprints or an “instance” of the object. This is done in javascript like a normal variable with the “new” keyword like this:

*var mike = new person(“mike”, 25);*

*var taylor = new person(“taylor swift”, 26);*

And there! We’ve created a person object for mike and a separate one for Taylor swift using blueprints for a “person” that we’ve defined by ourselves up in <head> !!

So why is this useful? Can we print it out? Of course we can! Let’s list out some things about our objects.

*document.write(mike.name); // prints mike*

*document.write(taylor.name); // prints taylor*

Objects are kind of like functions whose scope keeps existing and that don’t necessarily exit right away. You’re carrying the SCOPE with you. They’re a lot less magical than python objects.

**Objects Initializers:**

Now there’s a few different ways to make objects. The way above was using a constructor function. There’s another fancier/faster way. It squishes everything together into one line.

You’d use this if you’re creating one or two objects of the same type.

Use the constructors (above) if you plan on making many many objects.

So let’s go ahead and think about our object. You use “{“ brackets and set it up almost like a dictionary.

*mike = {name:”mike”, age:25};*

And boom. We have the exact same “mike” object as we did above! This is useful only if you’re making a few objects. Use contstructors if there might be many, but know what this is if you see it in existing code.

*document.write(mike.name + “ is “ + mike.age + “ years old.”);*

**Adding methods to our objects:**

So let’s start off with our constructor function for people and create an instance ‘mike’ only tell it about another function we’ll define on the outside:

*function people(name, age) {*

*this.name = name;*

*this.age = age;*

*this.yearsUntilRetire = yearsLeft;*

*}*

*var mike = new people(‘mike’, 25);*

So we can access the properties easy, but how can we add methods to this object (basically a function that uses properties of the object)? We make a totally separate outside function and use the ‘this’ keyword there!!!

*function yearsLeft() {*

*var numyears = 65 – this.age;*

*return numyears;*

*}*

Now we can do something like:

*document.write(mike.name + “ will retire after “ + mike.yearsUntilRetire() + “ years left till retirement”);*

And there! You’re using properties and methods with your object!!

**Arrays in Javascript:**

Arrays are a way to store many values in a single variable. They’re a lot like lists in python. It’s a lot easier and faster to create an array and it’s easier to manage.

Let’s see an example.

Suppose you had to list a bunch of people’s names? Instead of making variables for person1 person2 person3 that’ll take a ton of lines of code! It gets all jumbled.

An easier way to do this is to create an array. This might only take one or two lines of code!

Creation is a little different however. ARRAYS ARE OBJECTS so like your objects above you’ll need to make an INSTANCE of the array object with the ‘new’ keyword and the Array() object.

*var myArray = new Array(‘mike’, ‘sarah’, ‘tommy’, ‘marta’, ‘sal’);*

Alright that makes sense, so we have five different values in one variable. So how would we try and use this later on? You access them using an ARRAY INDEX. Each value in there is numbered (Starting at zero) so for the above example if we wanted to access ‘mike’ we’d use:

*document.write(myArray[0]);*

And 1 for ‘sarah’, 2 for ‘tommy’ etc etc. Each value in an array has an index! You use square brackets to tell it what you’re trying to get.

**Other ways to create arrays:**

How would you create an array without any items in the first place? You can tell an array how many items you’ll be storing in it! Just pass a number when you create your instance instead of the values.

*var things = new Array(3);*

*things[0] = ‘mike’;*

*things[1] = ‘mike2’;*

*things[2] = ‘mike3’; // anything beyond this would error-out!*

This is useful if you want to make sure your array only accepts X items! And then you access them just like normal.

*document.write(things[2]);*

Very simple. What if you want to create your array but simply don’t have items yet AND you don’t know the size of your array right now? That’s easy and doable. Just do the following to create the shell of an array which you can populate later:

*var stuff = new Array(); // empty array can be populated later*

And we’re done! Everything else from here on out is the same. It’s available just like the above arrays, just without anything in it right now.

**Array properties and methods:**

Arrays are objects so they have properties and methods! Let’s check out some of the more-useful ones.

*var dudes = new Array(“mike”, “randy”);*

Let’s take a look at some of the properties now. “Length” is a good one!

*// print length of array ‘dudes’*

*document.write(dudes.length);*

Nice! It prints out the number of values in our array. What other methods could there be? (A ton actually). In order to use some of these you typically need another array because many of them deal with sorting or combining etc. So let’s make another one!

*// bucky’s example not mine*

*var chicks = new Array(“lisa”, “erin”, “taylor”);*

Now let’s use an array method called concat to combine these two arrays!

*var people = dudes.concat(chicks);*

And our new array has six people and is the combined array of dudes and chicks! Pretty cool. In the next tutorial we’ll be learning about the more-useful and powerful array methods.

**Join and Pop:**

In this tutorial we’ll be learning about a few new methods.

“Join” takes an array and converts it to a string (similar to python). Let’s go right to an example.

*var movies = new Array(“avatar”, “godfather”, “goodfellas”);*

So now let’s see. We have three movies (elements in the array). What’s an easy way to convert this to a string?

*var string1 = movies.join(); // takes no parameters – separates with commas*

*document.write(string1); // writes the list separated by commas!*

When join doesn’t take parameters it separates by commas but we can tell it what to separate them by! That’s the parameter!

*var string2 = movies.join(‘-’);*

*document.write(string2);*

Prints out the list separated-by-hyphens-like-this.

Pop is very powerful. It returns and removes the last element of an array.

*document.write(movies[2]); // returns ‘goodfellas*

Okay that works fine. What if we want to return it then remove it..?

*document.write(movies.pop()); // removes movies[2] (goodfellas)*

Tested and working. Pop() returns then removes, bucky fails to demonstrate the ‘return’ portion but it’s there (similar to how you would in python).

**Array Reverse, Push, and Sort:**

These are three very commonly-used methods to manipulate the array.

*var bodyparts = new Array(“head”, “shoulders”, “knees”, “toes”);*

To reverse it simply call the reverse(); method for array objects.

*bodyparts.reverse();*

Now it’s the same array but the indices have been reversed in order!

The PUSH method is kind of the opposite of pop. Pop removed things from the end of the array and PUSH will ADD things to the end of the array.

*bodyparts.push(‘tongue’, ‘fingers’); // adds these two to end of array*

Finally there’s the *‘sort’* method. Sort just sorts things in alphanumeric order.

*bodyparts.sort(); // sorts alphabetically*

*var string1 = bodyparts.join();*

*document.write(string1); // print out our sorted bodyparts array as string*

Couldn’t be easier!

**Add Array Elements using a loop (Prompt):**

As the title suggests we’re going to add array elements in a loop. This is a cool little tecnique that we’ll likely be using in the real world.

We’ll also be learning about “prompt”!

*# prompt a textbox dialog that will ask them for info*

*var pie = prompt(“enter your name:”, “default text”);*

*document.write(pie);*

That’s pretty cool. This is a way to get user-inputs on-the-fly from a webpage. Tested and working! That’s pretty awesome. Wonder if this will work with rapid?

But it’s not relevant to our arrays yet so let’s marry the two.

*# use a loop and prompt boxes to populate an array*

*var crap = new Array(3);*

*for (var i=0; i<3; i++){*

*var temp = prompt(“enter your variable!”, “default text”);*

*crap[i] = temp;*

*}*

*document.write(crap[0] + crap[1] + crap[2]);*

So this will loop through and populate our array (well, we’re doing the work!).

Tested and working! This is pretty cool and follows similar methods that python and c++ would use to combine loops and arrays.

**Techniques to print/use arrays:**

Let’s first create an array:

*var stuff = new Array(‘apples’, ‘pears’, ‘bacon’, ‘tuna’, ‘ham’);*

First let’s use the sort method we learned earlier:

*stuff.sort();*

Now let’s make a for-loop to print them out:

*# use length method for arrays*

*for(var i=0; i<stuff.length(); i++){*

*document.write(stuff[i]);*

*}*

**Associative Arrays:**

These are different from the arrays that we used before. Instead of using array index numbers to access these, we can use strings of characters (keys) to access them instead. This sounds a lot like python dictionaries! Watch:

*var mike = new Array();*

*mike[‘caterpillar’] = ‘green’;*

Now the array ‘mike’ has a key-value pair of ‘caterpillar’ being the index for ‘green’.

*# prints out ‘green’*

*document.write(“mike’s fave color caterpillar is “ + mike[‘caterpillar’]);*

*Voila.* Tested and working.

You just make a normal array and instead of assigning things to indicies you assign them to strings. You can make this as large as you want and use variables for the string names.

**Javascript Math Objects:**

Just like the array object this is a special object which will have special built-in methods. This is a built-in javascript object. It’s very useful.

Properties are most-likely mathematical values.

*document.write(Math.PI); # print out pi*

*document.write(Math.E); # euler’s constant*

Let’s write a math program:

*var n = prompt (“enter a number: “, “”);*

*var answer = Math.sqrt(n); # square root of ‘n’*

*alert(“square root is “ + n);*

Tested and working! Very easy. If you’re ever doing common math operations that are more complicated than adding and multiplaying, realize that they’re likely in the Math object that’s built into your javascript already.

**Date Objects:**

Another very common object that you’ll be using all the time.

This object grabs the current date and the curren time. First we’re going to learn about the Set Interval object.

The inverval function can do X every X seconds ! Very useful and powerful for interactive pages!

*function doSomething() {*

*document.write(“weee “);*

*}*

*setInterval(“doSomething()”, 1000); # string of script to execute and miliseconds for interval*

Holy cow this is awesome! Let’s make a clock that, each second, it will return what time it is using the Date Objects.

*function printTime() {*

*var now = new Date(); # create a date object*

*var hours = now.getHours(); # get hours*

*var minutes = now.getMinutes(); # get minutes*

*var seconds = now.getSeconds(); # get seconds*

*}*

So now the variable called “now” has info on the date (tons of info) but we just want hours, minutes, and seconds! That’s why we use the built-in date object methods getHours, getMinutes, and getSeconds.

An issue:

<https://stackoverflow.com/questions/26521998/settimeout-and-setinterval-not-working-in-firefox-os>

It seems that modern browsers ban passing strings into setInterval and setTimeout, but you can still define a whole function within them (nope that doesn’t work either)

We tried using setTimeout which only works ONE time then the browser stops it. This is disappointing but useful for javascript applications outside of browsers still and at least we get the date built-in object and some methods out of it. This was less-prevelant when bucky was making the video in 2011, but whatever. The show must go on.

**Accessing Forms:**

We’re going to talk about javascript and forms.

One of the reason it’s so popular is that it works perfectly with forms and is great for verifying data (suppose a user had to enter an email – you’d want to be able to ensure that you can validate that it ends in @gmail.com or something right?)

Let’s see an example. Whenever you add a form to your page javascript gets a form object AUTOMATICALLY to work with. All we need to do is learn how to access it. So let’s make that xhtml form. Outside of javascript:

*<form>*

*Username: <input type=”text” />*

*<input type=”submit” value=”Submit!!!”/>*

*</form>*

So javascript uses an array and indexes all of your forms. In javascript you can access your forms by-number. The first form would be document.forms[0]. The more forms you have on your webpage the more elements will be on your page etc etc..

One of these properties is length so let’s print out the length of the object in that form. We’ll need to use the document object to access the forms array.

*var x = document.forms[0].length;*

*document.write(“there are “ + x + “ fields in forms 0”);*

This works.

**Accessing the ELEMENTS in the forms:**

So the length of a form is nice but pretty useless without being able to access those elements! Each form has a property called “name”

<form name=”mikesForm”>

Username: <input type=”text” name=”username” />

Password: <input type=”password” name=”password” />

<input type=”submit” value=”submit!!! />

</form>

To access it you’d so something like:

*var x = document.forms[0].elements[0].name; # access to name of first element from first form*

*document.write(“first element’s name is “ + x);*

Or you can access them with names instead of indexes+arrays!

*var x = document.mikesForm.username.name;*

This is the more-common approach.

**Simple Form Validation:**

This is the final tutorial of this intro series and we’ll be learning about how to access and verify/check the information given by our html forms.

We’re going to see some basic form validation. We’ll have a simple form and whenever we click “submit” it’ll check whether or not our checkbox is checked (so a little bit of html here).

Once you understand the basics of form-validation you’ll probably be able to use it for lots of different examples.

*<head>*

*<script type="text/javascript">*

*function validator() {*

*if(document.mikesform.thebox.checked) {*

*document.write("box is checked!");*

*} else {*

*document.write("box is not checked :(");*

*}*

*}*

*</script>*

*</head>*

*<body>*

*<form name='mikesform'>*

*<input type="checkbox" name="thebox"/>*

*<input type="button" value="press me!" onClick = "validator();" />*

*</form>*

*<script type="text/javascript">*

*var x = document.forms[0].length;*

*document.write("this form has " + x + " fields");*

*var y = document.forms[0].elements[0].name;*

*document.write("first one is " + y);*

*</script>*

*</body>*

See that’s not so bad? A full program/site right there basically. Those are the basics of form-validation.

Want to get a string from a text form field?

*document.forms[0].elements[0].value;*

And that’s it. You’re not exactly a javascript expert at this point but you can play/manipulate the basics enough to get stuff done with a little duckduckgo’ing.