Table of Contents

1 In these Sushi Cards you'll be learning JavaScript—one of the world's most popular programing languages. You probably use things built with it every day. It's in every major website, including YouTube, Facebook, Instagram and Google. It's also in lots of mobile apps and browser games.

2 Go to dojo.soy/trinket and click "Sign Up For Your Free Account" if you do not already have an account. You will need an email address to sign up.

3 Enter your email address and choose a password, or ask somebody to do this for you.

4 Creating an account allows you to save your work and access it from any computer. It also allows you to make a copy of a project somebody else has shared with you so you can make your own changes to it!

5 Go to dojo.soy/js-b-start. You will see a box containing an example JavaScript website project. On the right hand side is the website, and on the left hand side is the code that makes the website. If you are not signed in, you will need to enter your email address and password to be able to **Remix** the project.

6 Click the "Remix" button at the top right of the project (if it is not green, you have to sign in and then click it again). This creates a copy of the project for you to work with. It should say "remixed" after you click it

7 Next to the "Sign Out" button at the very top right corner of the page you should see your username and a drop-down menu (the tiny triangle tells you there is a drop-down). Click on it to show the menu and then select "My Trinkets".

In Trinket (this website), projects are called "Trinkets"

8 The project you just remixed will be shown together with some example projects for other programing languages. It will be called "Beginner JavaScript Remix". Click on it to begin editing!

9 These cards are designed to follow on from the beginner HTML Sushi Cards, but don't worry if you haven't done those. There's not much HTML in these cards and it'll be explained when you see it. Either way, you'll be getting some music from the internet and setting up a player on the page so the user can pick which track they want to listen to. The first thing you'll need to do is look at your “music.html” file and see that it's pretty basic:

```
<html>
    <body>
        <div id="jsSpace"></div>
    </body>
</html>
```

All you have there is the basic HTML code for a page with a `div` (division) element with an `id` attribute where the `div` is called "jsSpace". What you need to do now is include a few JavaScript files on the page. You do that using the `script` tag with the `src` attribute set to the name of your file. You've got three JavaScript files:

- **techie-functions**—Some of my code that you don't need to change, but feel free to take a look to understand how it's working
- **functions.js**—Some of my code that you'll need to make some changes to over the course of these Sushi Cards
- **my-script.js**—Where you'll be writing most of your code

The order you add the `script` tags in is important because you need to load a piece of code before you can use it. You'll need to load them in the order listed above, like so:

```
<html>
    <body>
        <div id="jsSpace"></div>
        <script src="techie-functions.js"></script>
        <script src="functions.js"></script>
        <script src="my-script.js"></script>
    </body>
</html>
```

10 If you look in the preview of the page, you'll see that there's actually nothing visible on it right now! You're going to use JavaScript to insert HTML into the page. Specifically, you'll need to:

- Get some song information from the internet
- Display the names of the songs
- Create a music player (it'll be invisible, but you'll hear it just fine!)
- Let the user choose which song to play on that player

You'll start to do all of this on the next card.

1 JavaScript code is normally made up of **functions**: sets of instructions joined together with a name. Think of something like making a cup of tea as a **function**:

- Get kettle
- Put water in kettle
- Boil water
- Get teapot
- Put teabag in teapot
- Put water in teapot
- Wait...
- Get cup
- Pour tea into cup

That's a lot of steps! It's much easier to teach someone how to make a cup of tea once and then just ask them to do that in future! It's the same with JavaScript. We use functions to do sometimes pretty complex things with simple commands. The great thing is that *you* don't have to write the function to use it. So to start with, use a few of mine by updating your **my-script.js** file to look like this:

```
getSongs()


whenSongsReadyDo(
  function(){
    var mySongs = getSongTitlesAndArtists()
    displaySongsList(mySongs)
    setupPlayer()
  }
)
```

What's happening here is your code asks the internet for some songs with `getSongs()`. Then it checks if the songs have arrived yet with `whenSongsReadyDo()` and, once they've arrived, runs the **function** inside it, which is the rest of the code for your program. That code does three things:

- It gets a list of songs and stores them in a container, which we call a **variable** called “mySongs”.
- It gives the list in “mySongs” to a function that displays them by creating some HTML
- It creates a music player and sets it to react to clicks on the songs

2 Now switch to the **index.html** file and look at the preview window. You'll see that there's a song title there—“Yesterday”—and a play button. If you click the button, the song will play. This is great, but it's not a lot of use for a few reasons. You're going to fix them on the next few cards:

- First, “Yesterday” is one of the most recorded songs in history and there's no way of telling whose version this is!
- Second, it only shows one song, even though there are lots of versions of “Yesterday”
- Third, you probably want to have cooler music on your webpage, so you'll need to search for something else!

Time for you to improve on my **function**!

1 Anyone checking out your music page will get quite confused if you're listing songs with the same title, or with cover versions recorded by different artists. So, you need to add more information to the song listing. Specifically, the name of the artist. When you're creating the song listings, you're using the `displaySongsList()` **function**. I wrote that function and you can find it in the "functions.js" file. It looks like this:

```
function displaySongsList (songsList) {

  var wrapper = document.getElementById ('jsSpace')

  var songsListDisplay = document.createElement ('ul')

  var song = songsList[0]

  songsListDisplay.appendChild(buildSongDisplay(song))

  wrapper.appendChild(songsListDisplay)
}
```

The version in the file has some notes on what each line is doing, but the one to care about, right now, is this one:

```
songsListDisplay.appendChild(buildSongDisplay(song))
```

2 There's a lot happening here, with **functions** inside **functions**. The **comments** in the file are worth reading to understand it.I could just append a piece of text, but I wanted to use some fancier HTML and add some values from the song, so I wrote another **function** called `buildSongDisplay()`. It's at the very top of the file and right now just looks like this:

```
function buildSongDisplay (song) {
  var songDisplay = document.createElement ('li')
  var songInfo = '<strong>' + song.title + ' </strong> &#9658;'
  songDisplay.innerHTML = songInfo
  songDisplay.classList.add ('songListItem')
  songDisplay.dataset.songId = song.id
  return songDisplay
}
```

The important lines, for you, are the ones that tell the page what goes inside that `li`.

```
var songInfo = '<strong>' + song.title + ' </strong> &#9658;'
songDisplay.innerHTML = songInfo
```

The first line creates a variable and stores text in it, including `song.title`, which is a **property** of the `song` **variable** that gets **passed** into the **function**. The second line puts it inside the `li`. You want to change the second line to add the artist. First, to add the word “by”, you'll need to change that first line like this:

```
var songInfo = '<strong>' + song.title + ' </strong> by &#9658;'
```

Now save the change and go back to the “music.html” page. Let it reload and see the difference. You can add any text you like in there. The weird set of symbols and numbers at the end (`&#9658;`) gives you the play arrow. Try another four numbers and see what you get. Change it back afterwards though!

3 Go back to “functions.js” and change the line a little more, to look up the artist **property** of the song, like this:

```
var songInfo = '<strong>' + song.title + ' </strong> by '+ song.artist +'&#9658;'
```

Notice that you need to leave the single quotes (`'`) and use a plus either side of the **variable** (**properties** are a particular kind of **variable**) name. Re-load the “music.html” page again and check it out.

4 There are a few more **properties** on the song that you can use. Update the `buildSongDisplay()` **function** to display one or two more of them:

- *genre*—the kind of music e.g. pop, rock, dance, etc.
- *releaseDate*—when the song was released
- *length*—how long the full song is

If you've done the Beginner HTML Sushi Cards, or know HTML some other way, try some other tags in here too! Maybe add a class to the `<strong>` and play with the CSS! Now check out your changes in “music.html” again!

1 Now you have the song listing displaying the way you want it to, it's time to make it display some music you want! The very first line in “my-script.js” is `getSongs()`. This **function** calls a **webservice**—a computer on the internet that will respond to a **query** that's correctly formatted with some information in a consistent format. In this case, what comes back is in **json** (JavaScript Object Notation) format, but my code handles most of that so you don't have to worry about it! You'll see it again in the Intermediate and Advanced Sushi though!

2 On the previous card you saw **functions** being **passed variables**. These **variables**, called the **parameters** of the **function** are used by the **function** to decide what to do and how. You don't need to change my `getSongs()` to get different songs, you just need to change the **parameters** you **pass** to it. First, go ahead and ask it for some music you'd like better than this song. You can do that by **passing** the name of a song, or an artist, like this:

```
getSongs('U2')
```

or like this:

```
getSongs('Let it go')
```

Plug in a song or band you love, save “my-script.js” and then reload “music.html” and check it out!

3 The other **parameter** that `getSongs()` will take is the number of results you want. Right now, you're only seeing one song. So how about adding a few more? When you're **passing** multiple parameters to a **function** you need to separate them with a comma (`,`) like this:

```
getSongs('U2', 3)
```

Notice that the number doesn't get surrounded by quotes (`''`)—you only need to do that for text. In fact, it will usually confuse JavaScript if you wrap a number in quotes! Make the change to fetch 3 songs and save “my-script.js” again.

4 Now reload “music.html” and see what happens.

Nothing changed, right? That's a bit odd. Do you know why?

It's because while you're now getting more than one song in, your code in the `displaySongList ()` **function** from the last card is still only displaying the first song. You'll need to go back and update it to display as many songs as it gets. You'll see how to do that on the next card!

1 Now you need to make the page display more than one song. You can do that by opening up “functions.js” and changing `displaySongList ()` to include a **loop** that repeats the same code until you have no more songs. The kind of **loop** you'll be using is called a **while loop**. It does the same thing over and over while a condition is true. By being clever with **variables**, you can make it display every song on the list. So, to begin with, make a couple of variables. Add them in just below the line that creates the `ul`, like this:

```
var songsListDisplay = document.createElement ('ul')


var songsDisplayed = 0
var songsToDisplay = songsList.length
```

The variables are created using the `var` **keyword**. Every **variable** needs that before its name when it's created. The first **variable**, `songsListDisplay` is just holding a number. It's going to be used as a counter, and it's starting at 0. JavaScript counts from 0 instead of 1.

2 The second **variable** is more interesting. It's storing the length of the `songsList`, which is a **property** of the `songsList` you can ask it for. The way this **loop** is going to work, to display every song, is by keeping a count of how many songs it has displayed in `songsDisplayed` and comparing that to `songsToDisplay`. As long as the count is less than (`<`) the target number, it will keep going. Add the **loop** in below the variables like this

```
var songsToDisplay = songsList.length


while(songsDisplayed < songsToDisplay){
  var song = songsList[songsDisplayed]
  songsDisplayed = songsDisplayed + 1
}
```

3 Look at how the `song` **variable** is created every time the **loop** runs. Because it is created in the **loop**, it is destroyed every time the loop ends. There's a new bit of code there, populating the `song` **variable**: `songsList[songsDisplayed]`. The list of all the songs stored in the `songsList` **variable** is numbered from 0 to (number of songs in list - 1) and you can access the information about any song by using the square brackets `[ ]` with the right number in them. By using the value of `songsDisplayed` and changing that value by 1 every time the loop runs, you move one step down the list every pass (called an **iteration**) through the loop. Take a look, too, at how the value of `songsDisplayed` is changed. Notice that we can add one to its current value and then store it back into itself. Also, notice how important the right order of the lines is here. If they were reversed, the first song would never be displayed!

4 There's one thing still missing here: The line that actually displays the song! Find these lines after the end of the **loop**:

```
var song = songsList[0]

songsListDisplay.appendChild(buildSongDisplay(song))
```

Delete the first one. You don't need it anymore as you are creating the `song` **variable** inside the **loop**. Cut (ctrl+x on Windows, cmd+x on a Mac) the second line and paste (ctrl+v on Windows, cmd+v on a Mac) it as the last line inside your **while loop**.

5 Now reload “music.html” and watch all the songs appear!

1 You're almost done, but some of that code you started out with is a little messy. Now that you understand more of how **functions** work, you can clean it all up. You're going to create a pair of **functions** to replace all the code in the “my-script.js” file. First, make a function called `loadSongsToPage ()` and cut and paste the three lines inside the nameless (anonymous) **function** that's in `whenSongsReadyDo ()` so that your file looks like this:

```
getSongs('U2', 3)


function loadSongsToPage (){
  var mySongs = getSongTitlesAndArtists()
  displaySongsList(mySongs)
  setupPlayer()
}

whenSongsReadyDo(
  function(){

  }
)
```

2 Next, replace the anonymous **function** with your new one, like so:

```
whenSongsReadyDo(loadSongsToPage)
```

That looks a little neater and is a bit clearer about what's going on. Because the **functions** are well named, you can read what's happening almost like a sentence.

3 Now, wrap everything up in a function called `putMusicOnThePage ()` like this:

```
function loadSongsToPage (){
 var mySongs = getSongTitlesAndArtists()
 displaySongsList(mySongs)
 setupPlayer()
 }


function putMusicOnThePage (){
  getSongs('U2', 3)
  whenSongsReadyDo(loadSongsToPage())
}


putMusicOnThePage()
```

4 Load the HTML page and check it all still works. It's nice, but you can make it even better. Just add a couple of parameters to your `putMusicOnThePage ()` function for the search query and the number of songs, like so:

```
function putMusicOnThePage (query, songCount){
  getSongs(query, songCount)
  whenSongsReadyDo(loadSongsToPage)
 }


putMusicOnThePage('U2', 3)
```

Your code still does the same thing, but now any programmer can come along and use it to put music on the page without needing to worry about exactly how that works. They just write one line and it appears! You've been doing this all along, with your **functions** calling the **functions** we wrote together (like `displaySongsList ()`) and calling my **functions**. You didn't need to know how mine worked, you just need to know what to give them and what to expect them to **return**.

That's the real power of **functions** in programing—you break up the pieces so things can happen or change inside one piece (or one programer's work) without the other pieces (or programers) needing to know or care as long as the same information going into a **function** still leads to the same effects coming out.

This is amazingly powerful and you'll see it in the rest of the Sushi Cards. You'll also be able to do it yourself if you try working with other Ninjas in your Dojo on a project together: You can write your own **functions**, in your own `.js` files and, as long as you've all planned in advance what input should get what output, you should be able to use each other's code by combining them in one HTML file like you did back on the first card in this series!