Back in the introduction to JavaScript, we mentioned that one of the important reasons of learning JavaScript is that we can use it to make our pages interactive. Its how we run some code when a user clicks on a button or type something in or hit the arrow key or whatever they do we can have some corresponding code that is executed at a given point in time, not just executed right at the beginning of the page load.

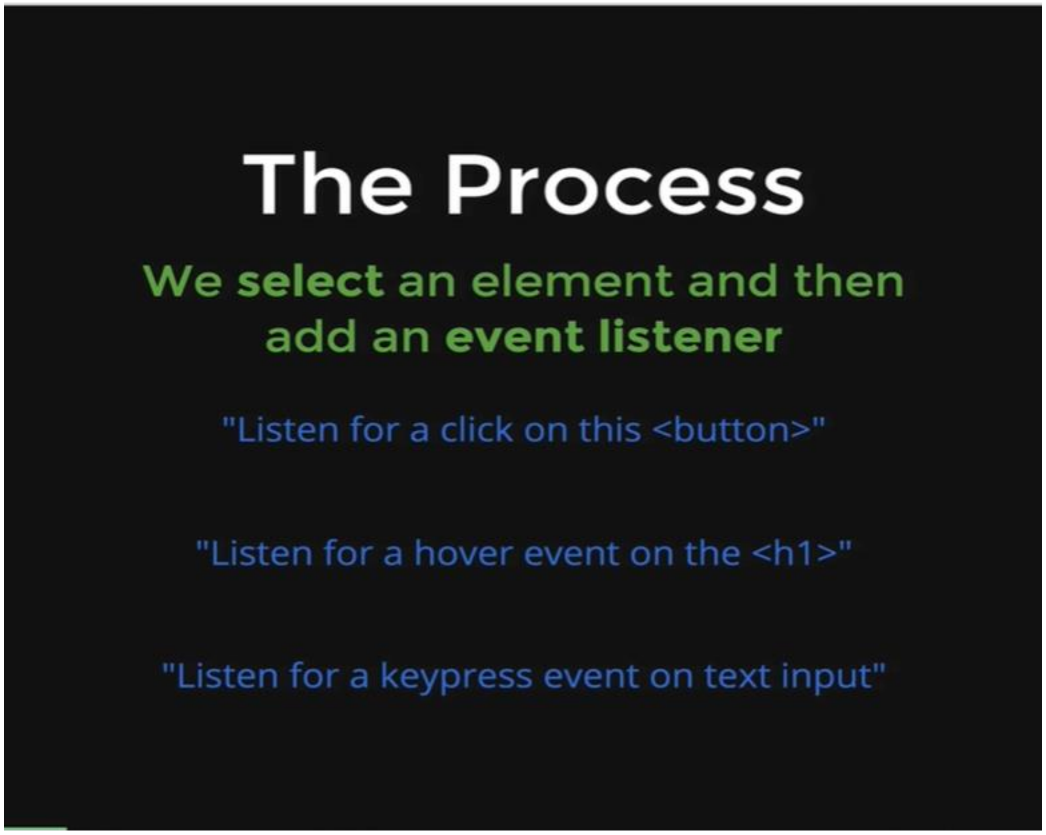
To do that we need to talk about *DOM Events*. This lesson will be an introduction to DOM Events, conceptually the process for setting them up, then also the syntax.

Events are everywhere

- *Clicking on a button*
- *Hovering over a link*
- *Dragging and Dropping*
- *Pressing the Enter key*

Events are everywhere. Think about when we interact with a webpage, the different things that happen. We can click on things like a button, we can hover over a link and something might pop up, we can drag and drop elements on certain pages, we might want to have a code that runs when a user double clicks. The point is events are all over the place and they are exciting. It is one of our favorite topics, because now we can make our webpages interactive. We can take the manipulation stuff that we have been learning about, changing colors, changing texts and then we can do that when something specific happens.

This unlocks the potential of manipulating the DOM. Its how we are going to develop games, its how we are going make form validators. These all stem from these events.



The Process

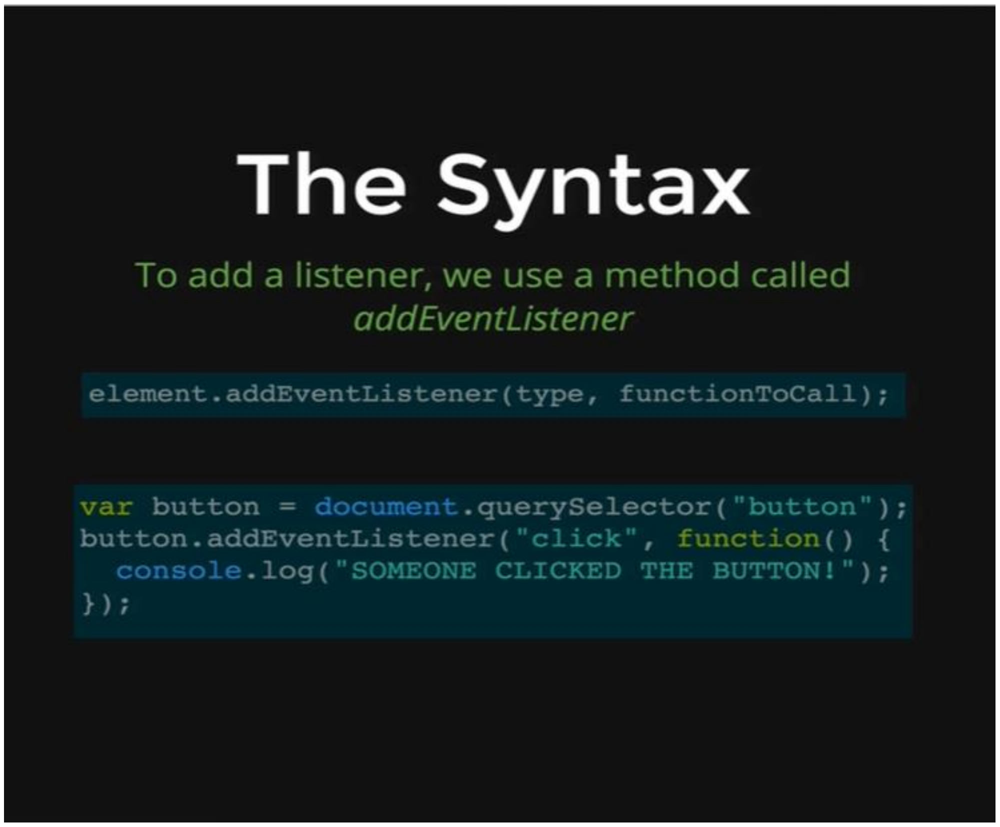
We select an element and then add an event listener

"Listen for a click on this <button>"

"Listen for a hover event on the <h1>"

"Listen for a keypress event on text input"

The way that these events work is that we attach them to specific elements. We select an element, like the first *button* or the second *h1* or the input where the type is equal to text and then we attach an *event-listener* to that selected element. Like on the picture, we might be listening to a click on a button, we might listen for a hover event on an *h1*, we might listen for a keypress event in a text input. We will talk more about the differences between those events, there is so many different types of events in JavaScript; but what is important is that we need to have an event that is being listened for on a given element. Just like before when we were talking about *select* and then *manipulate.* That still applies, we need to select something and then the manipulation that happens is that us adding an event-listener to that thing that we selected.



The Syntax

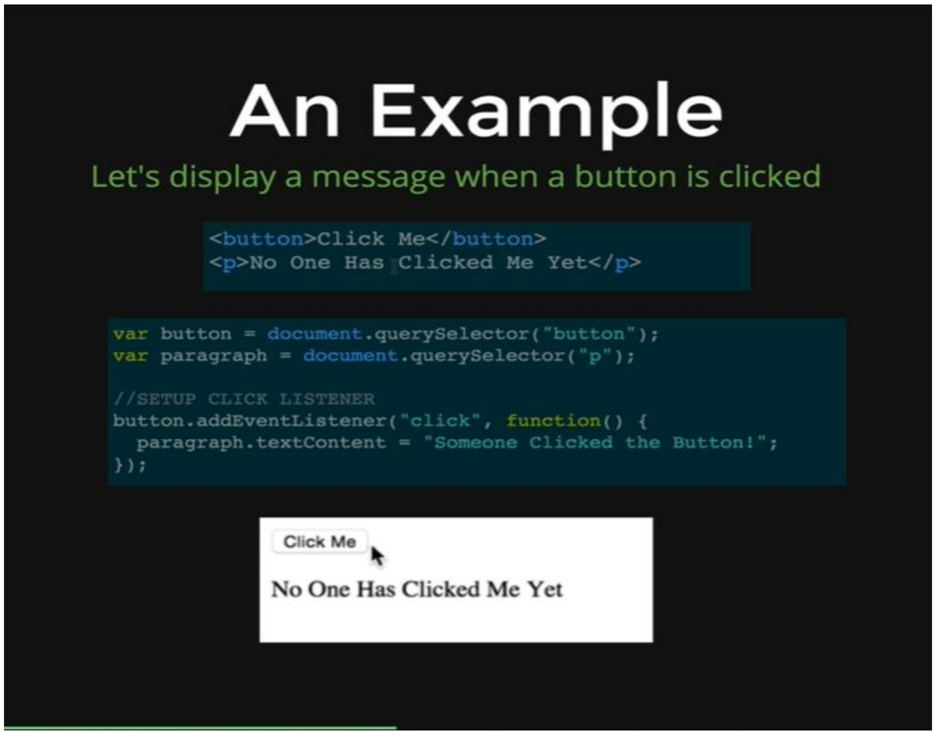
To add a listener, we use a method called *addEventListener*

```
element.addEventListener(type, functionToCall);
```

```
var button = document.querySelector("button");
button.addEventListener("click", function() {
  console.log("SOMEONE CLICKED THE BUTTON!");
});
```

There is one method in particular that we are going to use all the time which is called *addEventListener*; *addEventListener* is what we use when we select an element. Once we select an element, the first button or input type equal to password and we have something selected, then we connect *addEventListener* on it, then we pass two arguments, the first one is the *type* of event that we want to listen for and the second argument is the code that we want to run when that event happens. We have an example on our picture. We are selecting the first button on our page, using the *document.querySelector*, does not matter where the button is or how it looks or anything, just any button and then we are calling *button.addEventListener*, and we are passing the *click* event and the second argument is a function and this is not run right away. This function is called a *callback.* This function is not run immediately, it only runs once the *click* event fires, so once the user clicks on the button, JavaScript calls the function and then the function runs, thus using the console.log it shows the user **"SOMEONE CLICKED THE BUTTON!"**.

Thus, the function used in the argument will run, once the button connected with the *addEventListener* is clicked.

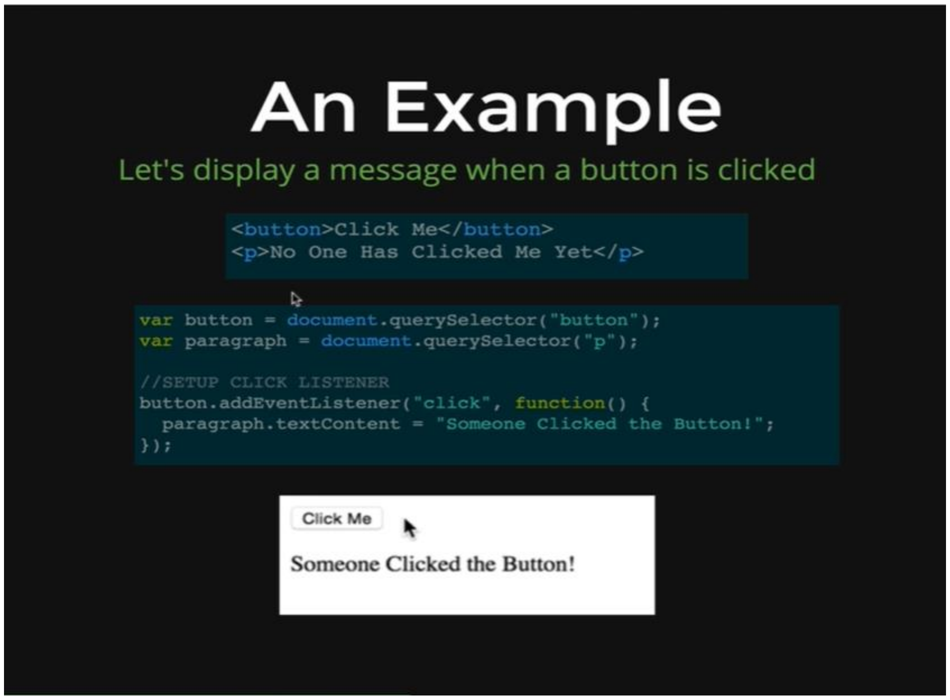


An Example

Let's display a message when a button is clicked

```
<button>Click Me</button>
<p>No One Has Clicked Me Yet</p>
```

```
var button = document.querySelector("button");
var paragraph = document.querySelector("p");

//SETUP CLICK LISTENER
button.addEventListener("click", function() {
  paragraph.textContent = "Someone Clicked the Button!";
});
```



An Example

Let's display a message when a button is clicked

```
<button>Click Me</button>
<p>No One Has Clicked Me Yet</p>
```

```
var button = document.querySelector("button");
var paragraph = document.querySelector("p");

//SETUP CLICK LISTENER
button.addEventListener("click", function() {
  paragraph.textContent = "Someone Clicked the Button!";
});
```

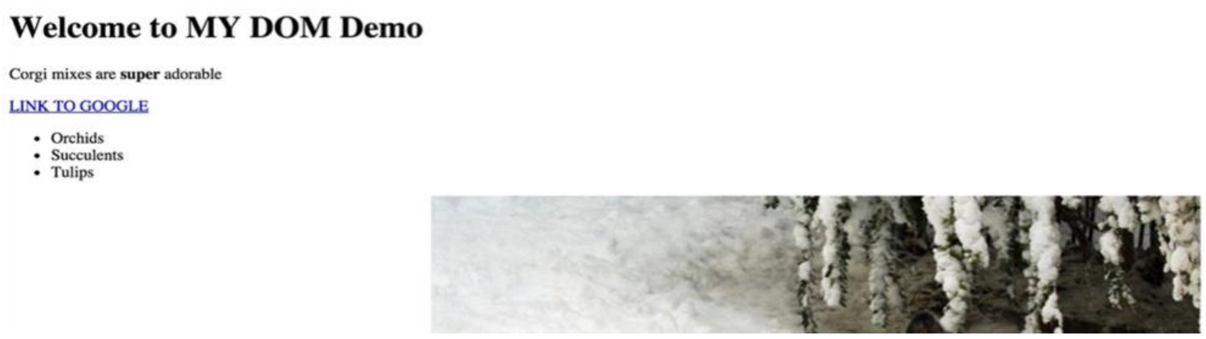
Here we have another example. In this case, we have a markup button and a paragraph, and we selected both with a *querySelector* and we saved them both to two different variables and then all that we do is connect an *addEventListener* to our button. We are passing the *click* as an event and as the first argument, and then the function as the second argument that will run once the user clicks that button. The paragraph (*p*) initially contains the text content “No One Has Clicked Me Yet” and on clicking the button the paragraph’s text content changes to “Someone Clicked the Button!”. This occurs as we have connected the *paragraph* variable with the *textContent,* thus our text content gets reset to the new content.

Now we will do a quick demonstration using our HTML page.

```
<!DOCTYPE html>

<html>

<head>

        <title>DOM Demo</title>

        <style type = “text/css”>

                .big {

                        font-size: 100px;

                        color: orange;

                        border: 5px solid red

                }

        </style>

</head>

<body>


<h1>Welcome to My DOM</h1>


<p>Corgi mixes are <strong>super</strong> adorable</p>


<a href= “www.google.com”>LINK TO GOOGLE</a>

<ul>

        <li>Orchids</li>

        <li>Succulents</li>

        <li>Tulips</li>
```

```
</ul>

<img src= http://barrelhorseworld.com/dogs/images/1145556d.jpg>
<img src=
http://api.ning.com/files/8Ni7RKwUY0n21uCgBw2JAxhPqmlBSklPmAiXbElaMFfa30mZFQKiE5gcWZP2iZJlSLIrtvQ3j4UH3Ezo5ZIc84ovRRC2J2ZT/TobySnow20.jpg>

</body>
</html>
```

Output:



Welcome to MY DOM Demo

Corgi mixes are **super** adorable

LINK TO GOOGLE

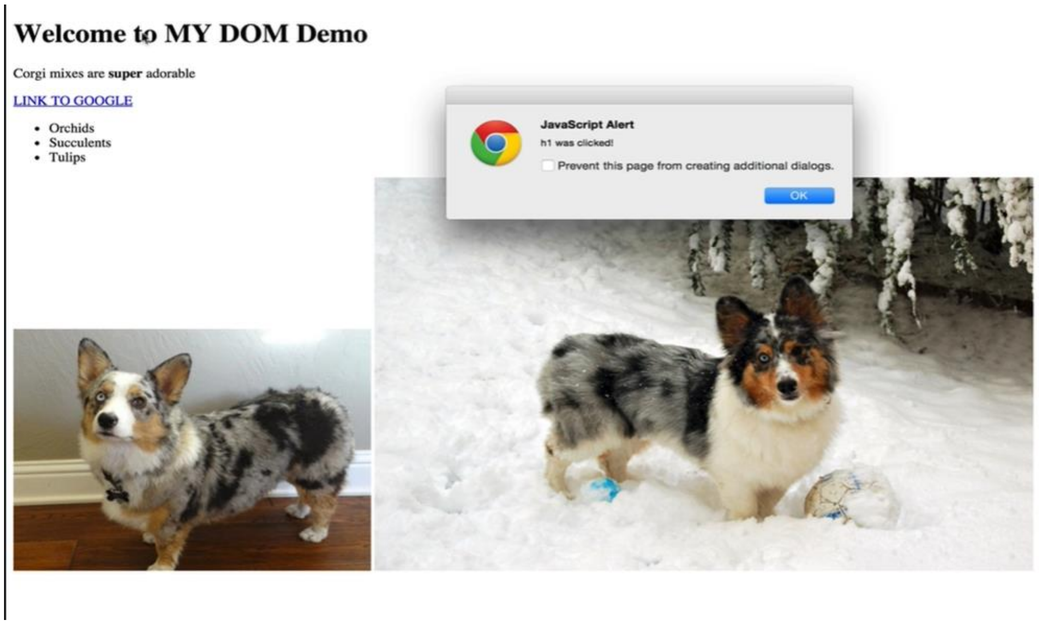
- Orchids
- Succulents
- Tulips

Let’s start by making the *h1* change color when a user clicks on it.

Console coding:

```
var h1 = document.querySelector(“h1”);


h1
    ➔ <h1>Welcome to MY DOM</h1>


h1.addEventListener(“click”, function() {
        alert(“h1 was clicked!”);
        })
```

After writing this code, we press *Enter*, then when we click the *h1* text-content, we get the pop-up.



Welcome to MY DOM Demo

Corgi mixes are **super** adorable

LINK TO GOOGLE

- Orchids
- Succulents
- Tulips

JavaScript Alert

h1 was clicked!

Prevent this page from creating additional dialogs.

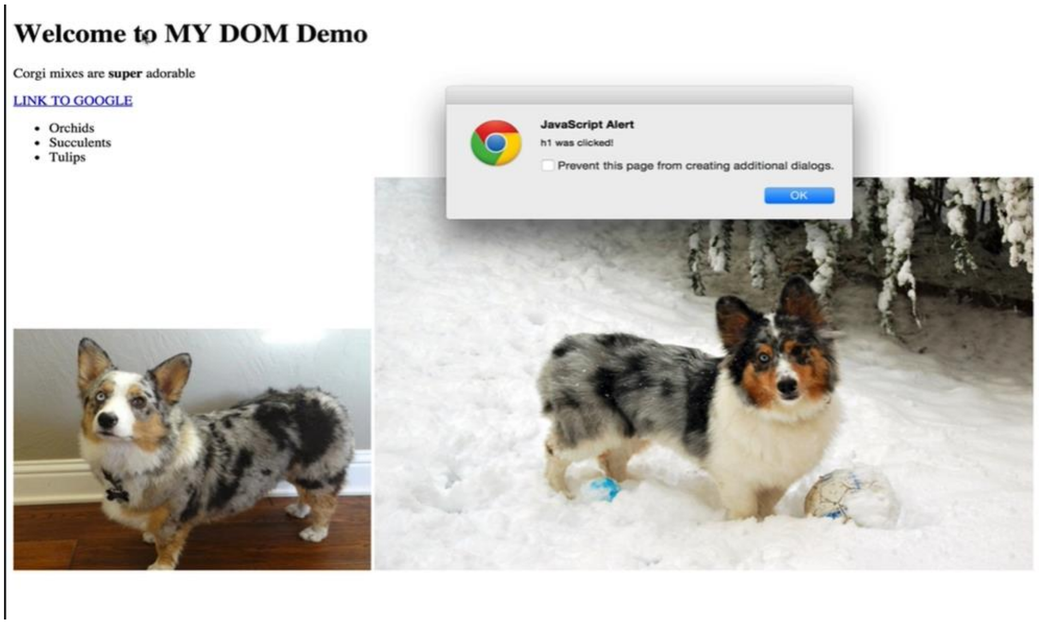
This is important, it only applies to the *h1*. Rather than just alerting, let’s try doing something else.

Console coding:

```
h1.addEventListener(“click”,  function() {
        h1.style.background = “orange”;
        })
```

Again, we do not see anything happening after clicking *Enter* but we know in the background the *addEventListener* code got registered in the console.

Then when we click the *h1* again, we get the alert popping up as below,



Welcome to MY DOM Demo

Corgi mixes are **super** adorable

LINK TO GOOGLE

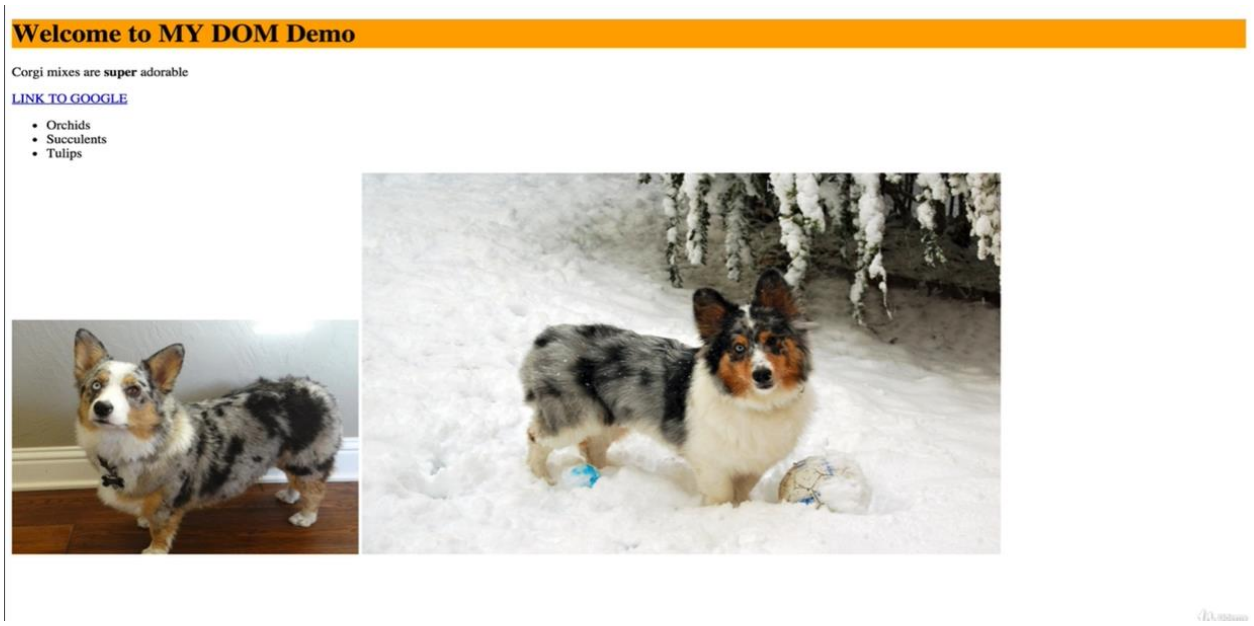
- Orchids
- Succulents
- Tulips

JavaScript Alert

h1 was clicked!

Prevent this page from creating additional dialogs.

But then we click the *OK* button and the *h1* color changes to orange as below,


Welcome to MY DOM Demo

Corgi mixes are **super** adorable

LINK TO GOOGLE

- Orchids
- Succulents
- Tulips

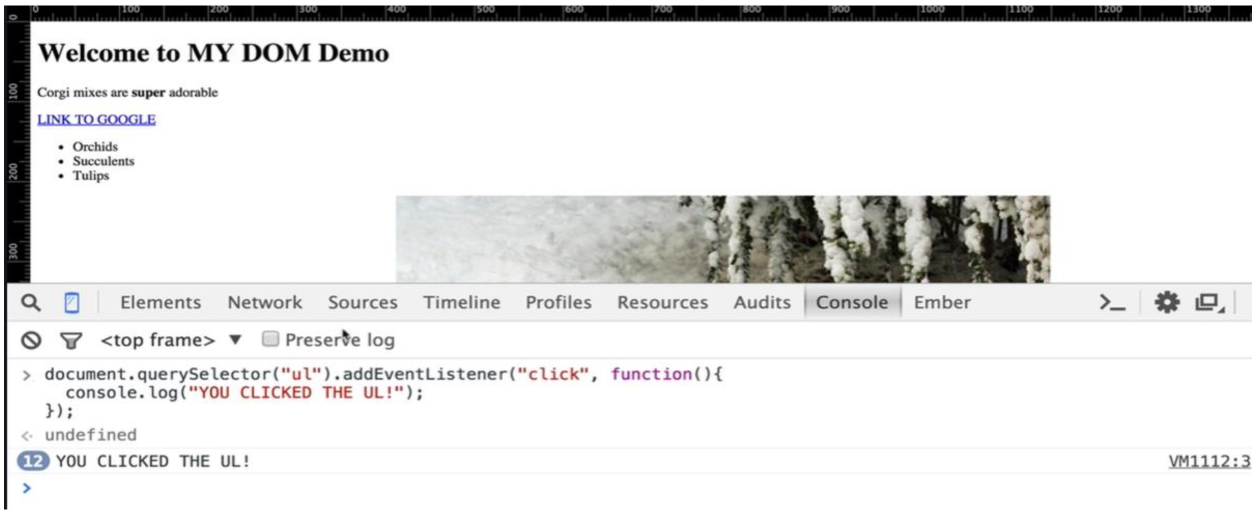
The moral here is that we can have more than one event-listener on a given element. When we click on that *h1*, the first listener that we added was listening for a click and it ran an alert. Then we added another listener afterwards that also listened for a click, but it changed the background color. So, when we click, both runs, as the new event-listener does not replace the old event-listener instead the new event-listener gets registered to the console while the old one also stays in the background with the new one. As a result both of them runs after clicking the *h1*.

We will also do a quick demonstration by adding an event-listener to the *ul*. We will not set it to a variable this time.

Console coding:

```
document.querySelector("ul").addEventListener("click", function() {
        console.log("YOU CLICKED THE UL!");
});
```

After pressing enter, the method gets registered in the console. Now let's click the *ul*. But how would we click the *ul*. The ul has three *li*s inside them. It's a big chunk on our page. So, we can click anywhere in the *ul*. That includes the *li*s or any other element in the *ul*. Any click in the *ul* will display the content in the console as below,



```
> document.querySelector("ul").addEventListener("click", function(){
    console.log("YOU CLICKED THE UL!");
  });
< undefined
12 YOU CLICKED THE UL!                                    VM1112:3
>
```

Now we will change the *li* itself, when we click on an individual *li* we want something to happen. There are few ways of doing that and we are going to start with the simplest one which is attaching one event-listener to each *li*.

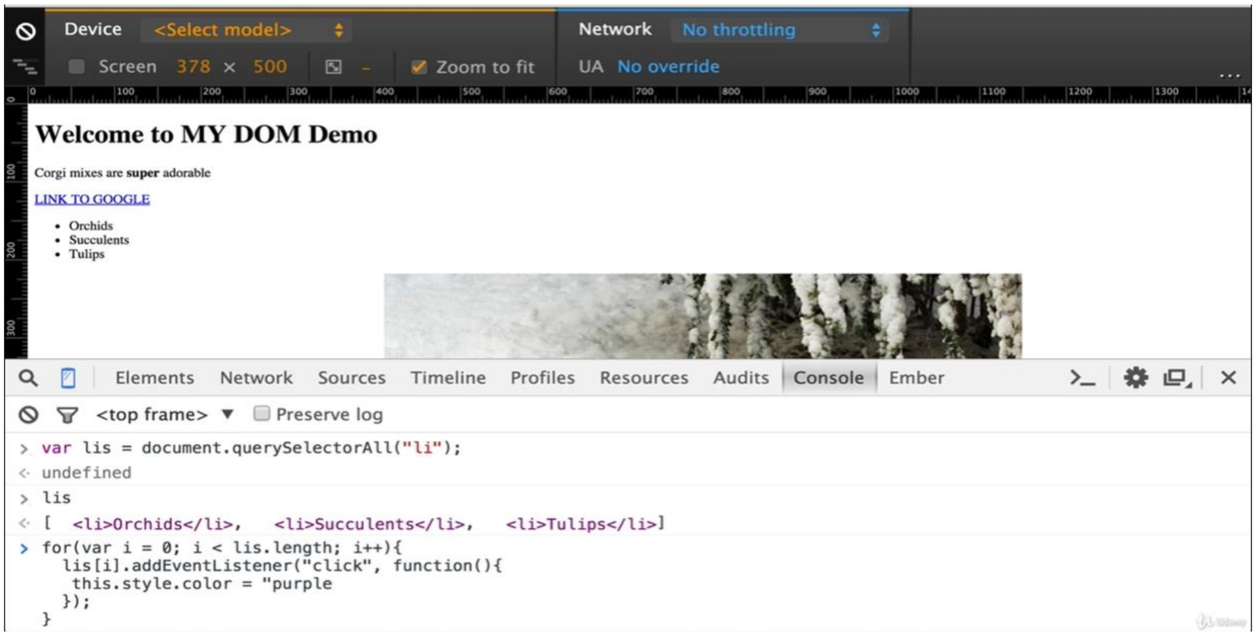
As a heads up what we will be doing eventually is attaching one event-listener to the *ul* and then inside of that event-listener we are going to detect which *li* specifically inside the *ul* was clicked on. We will do it all with one event-listener. For now, we will be adding a separate event-listener to each of the *li*.

Console coding:

```
var lis = document.querySelectorAll("li");

li

    ➔ [ <li>Orchids</li>, <li>Succulents</li>, <li>Tulips</li> ]

for(var i = 0; i < lis.length; i++) {
        lis[i].addEventListener("click", function() {
         this.style.color = "pink";
        });
}
```

After pressing *Enter*, the *for loop* gets registered in our console. Inside of an event-listener the *this* keyword refers to the item that was clicked on, or the item that was hovered on, or the item where the keypress occurred on; whatever the element that goes before the *.addEventListener*, whatever is selected there is what the *this* keyword refers to.

Now let's click on the *li* and see how it changes.



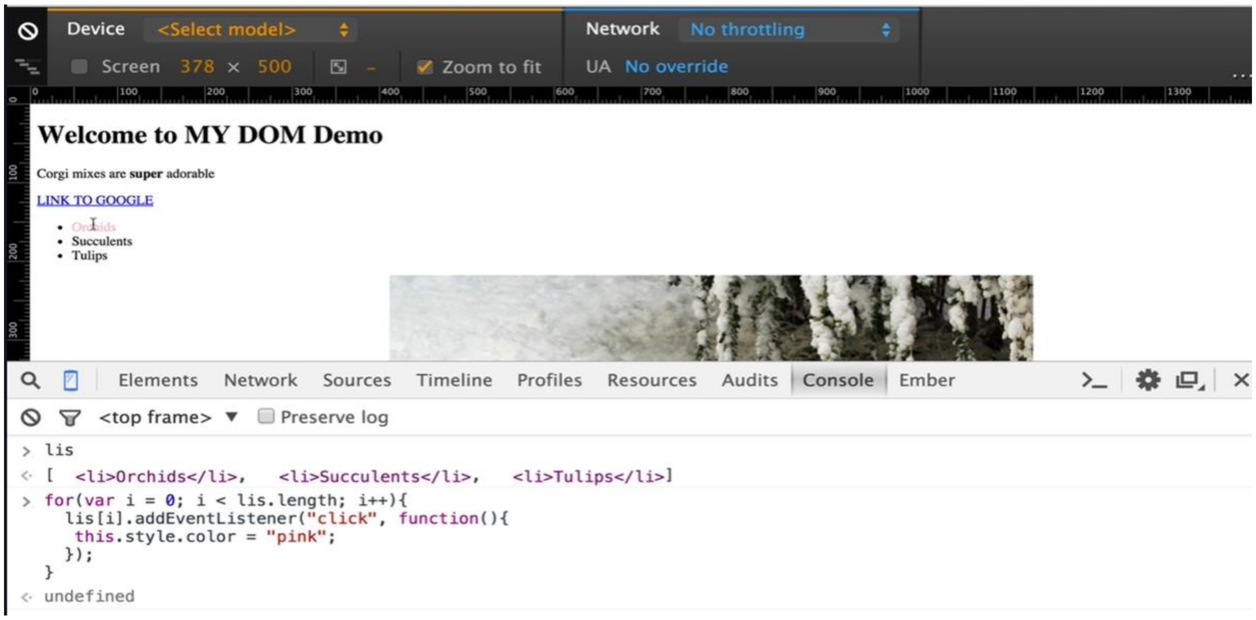
Welcome to MY DOM Demo

Corgi mixes are **super** adorable

LINK TO GOOGLE

- Orchids
- Succulents
- Tulips

```
> var lis = document.querySelectorAll("li");
<· undefined
> lis
<· [  <li>Orchids</li>,   <li>Succulents</li>,   <li>Tulips</li>]
> for(var i = 0; i < lis.length; i++){
    lis[i].addEventListener("click", function(){
     this.style.color = "purple
    });
  }
```



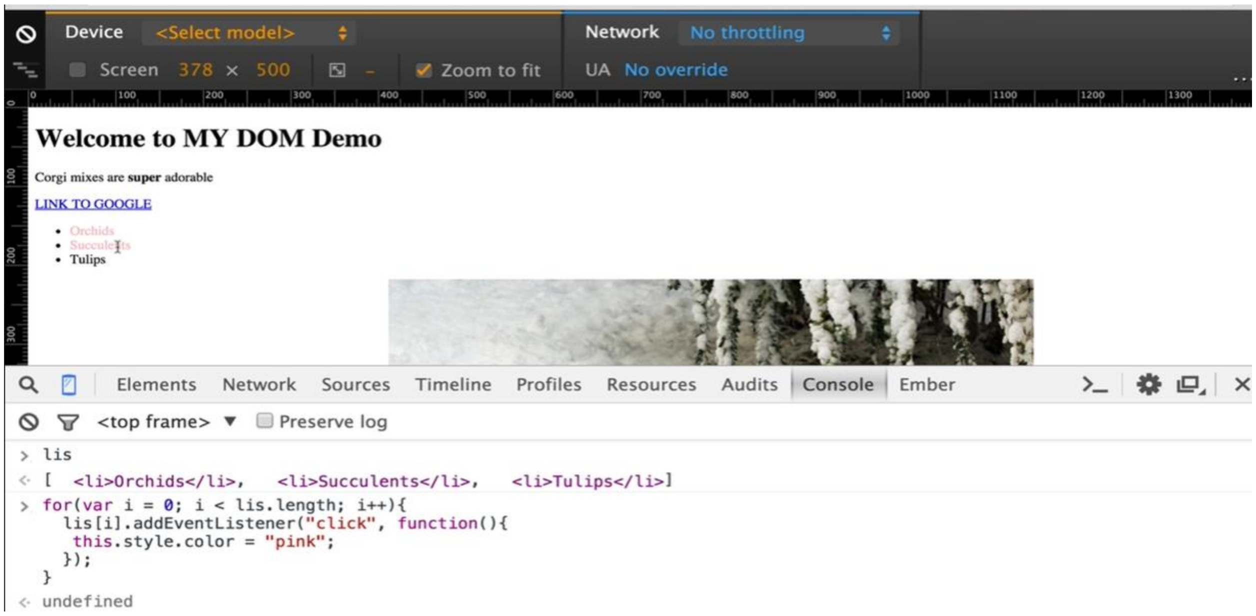
Welcome to MY DOM Demo

Corgi mixes are **super** adorable

LINK TO GOOGLE

- Orchids
- Succulents
- Tulips

```
> lis
<· [  <li>Orchids</li>,   <li>Succulents</li>,   <li>Tulips</li>]
> for(var i = 0; i < lis.length; i++){
    lis[i].addEventListener("click", function(){
     this.style.color = "pink";
    });
  }
<· undefined
```



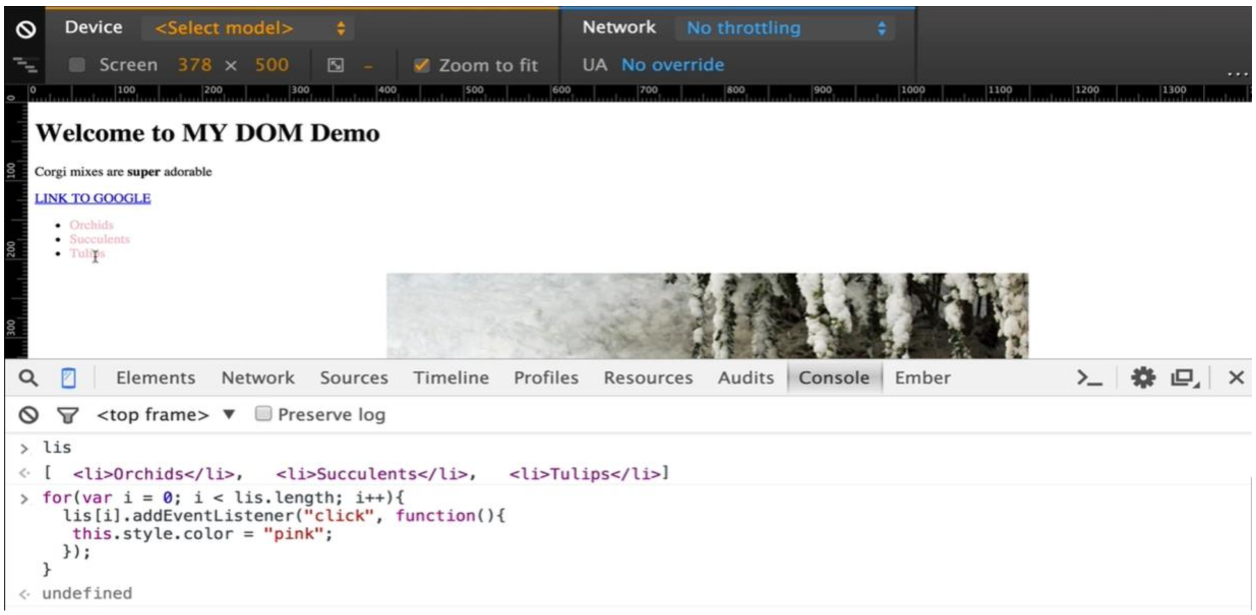
Welcome to MY DOM Demo

Corgi mixes are **super** adorable

LINK TO GOOGLE

- Orchids
- Succulents
- Tulips

```
> lis
<· [  <li>Orchids</li>,   <li>Succulents</li>,   <li>Tulips</li>]
> for(var i = 0; i < lis.length; i++){
     lis[i].addEventListener("click", function(){
      this.style.color = "pink";
     });
  }
<· undefined
```



Welcome to MY DOM Demo

Corgi mixes are **super** adorable

LINK TO GOOGLE

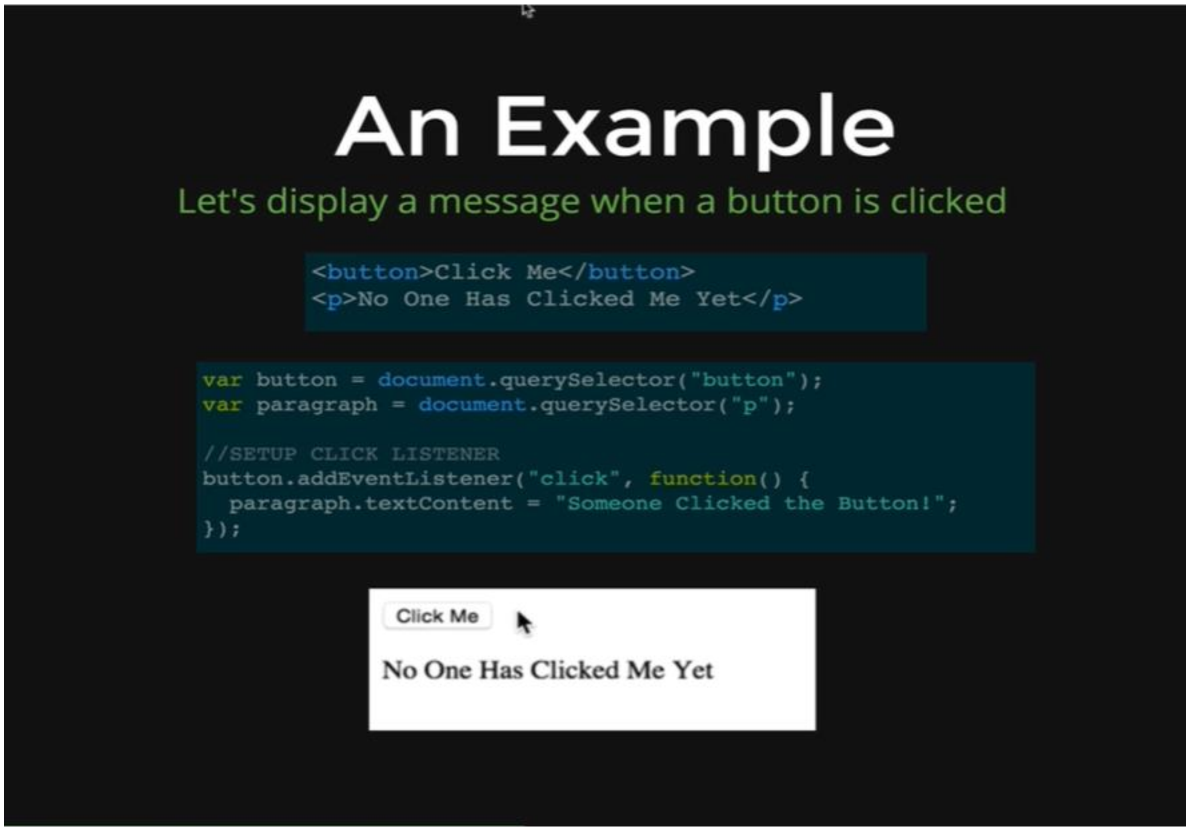
- Orchids
- Succulents
- Tulips

```
> lis
<· [  <li>Orchids</li>,   <li>Succulents</li>,   <li>Tulips</li>]
> for(var i = 0; i < lis.length; i++){
     lis[i].addEventListener("click", function(){
      this.style.color = "pink";
     });
  }
<· undefined
```

As you can see on the pictures that as we click on the first *li Orchids*, it turns pink, then we click on the second *li Succulents*, it turns pink and then we click on the third *li Tulips*, and it also turns pink.

```
for(var i = 0; i < lis.length; i++) {
        lis[i].addEventListener("click", function() {
         this.style.color = "pink";
        });
```

This pattern here of selecting a bunch of things, looping through each *li* individually and adding an event-listener to each of them is very common. We will see it a lot in next few lessons that we go through.

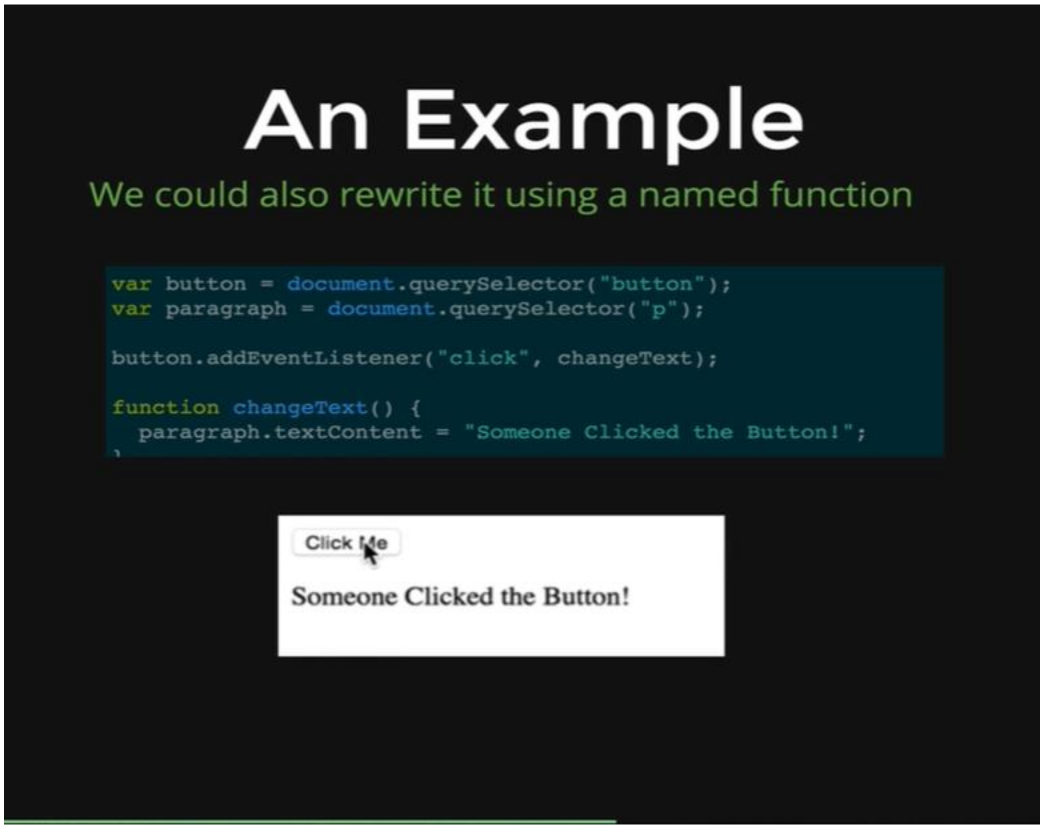


An Example

Let's display a message when a button is clicked

```
<button>Click Me</button>
<p>No One Has Clicked Me Yet</p>
```

```
var button = document.querySelector("button");
var paragraph = document.querySelector("p");

//SETUP CLICK LISTENER
button.addEventListener("click", function() {
  paragraph.textContent = "Someone Clicked the Button!";
});
```



An Example

We could also rewrite it using a named function

```
var button = document.querySelector("button");
var paragraph = document.querySelector("p");

button.addEventListener("click", changeText);

function changeText() {
  paragraph.textContent = "Someone Clicked the Button!";
}
```

The last thing we will mention here is that we do not always use an anonymous function, like we did with the function where we were changing the text content in a paragraph by selecting the paragraph tag. There we were defining a function with parentheses after it and that function had no name after it. We always do not need to do that, we can declare named functions separately like the one we did on this picture, where we defined a function by the name *changeText.* Then all we need to do is pass the function name (*changeText*) with the event name (*click*) and that calls the function on occurring the event.

Notice that we do not have parentheses when we are passing the function name to the *addEventListener* method, because that would execute the function immediately. So, we are just passing the value of the function, we are passing the function content basically to the event-listener and saying that here is the function, we want you to run it when the user clicks on the button but don’t run it just yet.