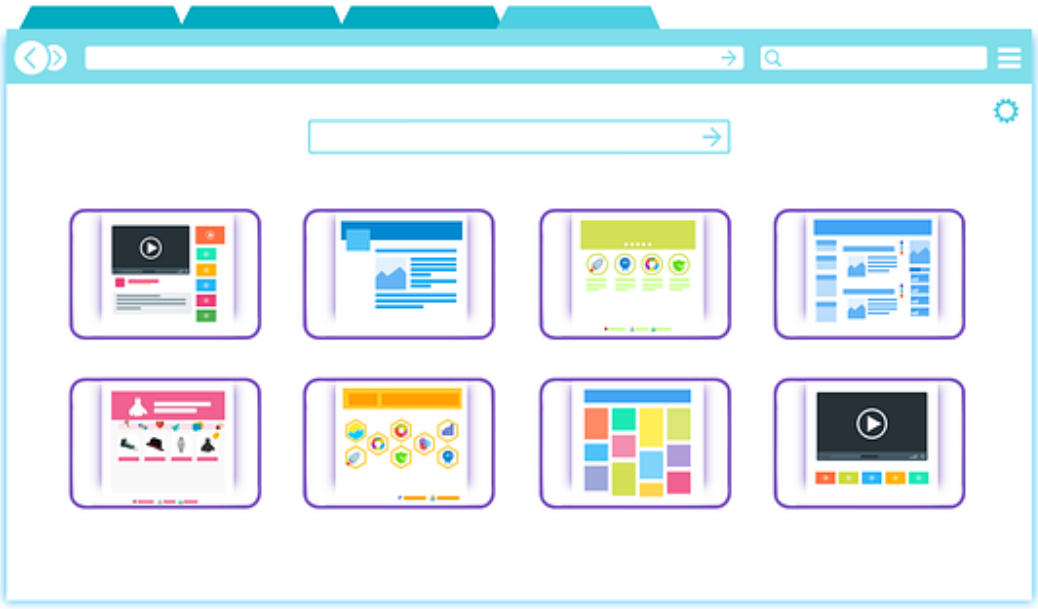
https://javascript.plainenglish.io/passing-arguments-to-event-listeners-in-javascript-1a81bc397ecb

Passing Arguments to Event Listeners in JavaScript

JavaScript essentials that you might not know about



Events are a fundamental part of JavaScript and the DOM interaction. Many times we only need access to the event object in the event handler. Since the event object gets passed to the event handler by default, we don´t need to worry about passing anything to the function. However, in some cases we do need to pass arguments to the function. There are different ways of doing it. Here is the way that I find the most intuitive.

Passing arguments to an event handler can be particularly useful when the handler is reused for different events. For example, we want to pass different content depending on which button is clicked, but the logic of the function is the same. In that case we should reuse the function and can just pass in varying content.

Example

We start off with the following code

```
function changeContent(content){
    targetNode.innerText = content;
}

let newContent = "Lorem ipsum";

//something like this but with an argument
btn.addEventListener("click", changeContent);
```

Let´s pretend `targetNode` as well as `btn` are variables that refer to elements in the DOM. `targetNode` is where we want to replace content in, and `btn` is the button that should trigger this. We need to add an event listener to `btn` and we also want to pass the new content to the event handler.

How do you pass `newContent` as the argument to the event handler?

Let us start with what is not possible. It is not possible to pass the argument to the function directly because this would be the same as calling the function.

```
// WRONG!

btn.addEventListener("click", changeContent(newContent));
```

The effect would be the same as

```
changeContent(newContent);
```

The function will be executed immediately when the line gets parsed. An event handler executing without the event having fired is of course not what we want. In addition, when the event fires, the event handler will not execute.

What works is using an anonymous function instead.

```
btn.addEventListener("click",
   () => { changeContent(newContent) });
```

This works, because we are not calling the function directly as we did above. We are passing a function definition to be used as a callback function that calls our actual function while passing in the argument. Since the outer (anonymous) function calls another function that does the actual job, I refer to this as the anonymous function / delegation approach.

We do not have to use an arrow function here, we can also just pass a regular anonymous function. There is a difference in the `this` keyword depending on your choice here, but this is the topic of another post.

```
btn.addEventListener("click", function(){
    changeContent(newContent)
});
```

We can also access the event object but we need to pass it to the function as well as adjust the function definition:

```
btn.addEventListener("click", (event) => {
    changeContent(event, newContent)
}); function changeContent(event, content){
    targetNode.innerText = content;
}
```

Sidenote

Since an event handler is nothing other than a callback function, you can use the same approach to pass arguments to any other callback function.

```
setTimeout( function() {
    sayHi("@kathimalati") }, 1000);

function sayHi(name) {
    alert(`Hi ${name}`);
}
```

Other approaches

As always, there are different approached to accomplishing what we want. There is another option of using the `bindfunction` to pass in values to your function, but this way you don´t pass it in as an argument. Rather you make it available via the `this` keyword. You can read more about that here. I chose the anonymous function / delegation approach, because I find it the most intuitive.