Event Listener

We can use event Listener when the DOM elements on the page emits events for things like click, drag etc., they will fire of events when they are interacted, we can use event Listeners to listen the events and react accordingly. We can attach event Listener to al the elements as well as document and windows. Lets see an example.

<button class='btn'>Click Me! </button>

In order to add event listener we first need to select an element we need to attach.

const btn = document.querySelector('.btn');

Next we will addEventListener() on the button element. EventListener will take two types of arguments.

1. What type of event that you want to listen?
2. A call-back function.

A call-back function is a regular function its just a word we use describe when we pass a function to a method that would be called at a later point of time. We provide name and reference to a function to addEventListener(). One of the more common ways to pass the callback function is as an anonymous function as shown below.

btn.addEventListener('click', function(){

console.log('Its Clicked');

});

Each time you click the button you will see Its Clicked in the console.

There are three steps with event listeners:

1. Go get something
2. Listen for something (such as a click)
3. Do something

In our example above, we passed in an anonymous function (it is an anonymous function because there is no name to the function, no way for us to reference that function outside of the listener). However, we can create a named function as shown below.

```
Function handleClick(){
console.log('Its Clicked');
}
btn.addEventListener('click', handleClick);
```

In the above code if you notice we are passing the reference of the function. Instead of passing it handleClick() like this, This allows us to still call handleClick from elsewhere in our code, as well as from within the console. If you try to load handleClick() instead of handleClick and refresh the page you will see Its Clicked message before clicking button this is because it is running on the page load when we are passing the function handleClick() instead of handleClick. If we pass handleClick the browser will run for us, We simply pass it a reference so it knows what to run when the time comes.

Removing an Event Listener

To remove an EvenListener from an element you must have a function reference. removeEventListener() takes 2 arguments.

1. type of event
2. function.

It is not possible for the event Listener to remove the element from all the click evens. You need specify the function reference that need to be removed as shown below.

```
btn.removeEventListener('click', handleClick);
```

If you refresh the page, you will see that the even listener no loner works. This is called unbinding. Binding means taking a function and listening for a specific click event of an element. If we had done an anonymous function, we couldn't have removed the click handler. If you ever in the future need to remove an event listener, remember not to use an anonymous function.

Listening to Events on Multiple elements

Wanting to listen on multiple elements is a very common thing. Let's say you have 40 buttons on the page and anytime you come across a specific type of button, or a specific type of image, or anything like that, you want

to listen for the event for all of the things that are on that page. Lets see an example as below.

```
<button class="buy">Buy Item 1</button>
<button class="buy">Buy Item 2</button>
<button class="buy">Buy Item 3</button>
<button class="buy">Buy Item 4</button>
<button class="buy">Buy Item 5</button>
<button class="buy">Buy Item 6</button>
<button class="buy">Buy Item 7</button>
<button class="buy">Buy Item 8</button>
<button class="buy">Buy Item 9</button>
<button class="buy">Buy Item 10</button>
```

Now, how can you listen for a click on all of them?

It doesn't make sense to have to select all 10 of them and then have to attach an event listener 10 times. That is actually what we are doing, but there is a much more efficient way. First we need to select all the buttons.

```
const buyButtons = document.querySelectorAll("button.buy");
```

If you want to add the event listener to all the buy buttons, you have to loop over and for each element attach it individually.

ForEach

You may have noticed that in the buyButtons prototype, there was a method called forEach. That is going to allow us to loop over each of the items. We will take the buyButtons and call the forEach() method on them. ForEach is a method the runs a function for each time in our nodeList. Anything you put in for each will happen 10 times. Lets see an example.

```
buyButtons.forEach(function(buyButton){
console.log(buyButton);
}
```

If run the above code you will see that all the buyButtons logged in the console. Lets create a function with reference.

```
Function handleBuyButtonClick(buyButton){
Console.log(‘Buying the Item’);
buyButton.addEventListener (‘click’, buyItem);
}
buyButtons.forEach(handleBuyButtonClick);
```

You can even use an arrow function for the handlers as shown below.

```
buyButtons.forEach((button) => {
 button.addEventListener('click', () => {
 console.log("You clicked it!")
})
})
```

The only downside for using the arrow function for your event listener like we did in the example above is that you cannot unbind it because it's an anonymous function in this case.

Targets, Bubbling, Propagation and Capture

We are going to loop every single button and attach a handler to it as shown below.

```
buyButtons.forEach(function(buyButton){
burButton.addEventListener(‘click’, handleBuyButtonClick);
});
```

If I have one function which is called by 10 different button event, How to know which button as trigged the function. In order to solve this problem we can modify our function by passing a parameter or a placeholder which we learned in the previous topics.

```
function handleBuyButtonClick(event){

console.log('Buying the Item');

console.log(event);

}
```

When the browser runs the function handleBuyButtonClick and when someone clicks the button it will run the function an pass to us number of arguments the first one is the event object. If you click the button you will see few things in the console will the first one is the pointer event. The button clicks, touches and mouse movements are consolidated into one event called as pointer events. If you expand the pointer event you will see all sort of things they are as explained below.

isTrusted: It is a Boolean value which tells us if the click was coming from someones mouse because there is a chance to fake the mouse clicks.

Pressure: on new ipad or other devices, there is pressure sensitivity. It says the screen resolution screenX, screen Y, client X, client Y , page X and page Y etc.,

The one we are interested is target and currentTarget.
Within handleBuyBackButtonClick, log event.target as shown below.

```
function handleBuyButtonClick(event){

console.log('Buying the Item');

console.log(event.target);

}
```

If you refresh the page and click the button it will show you which button was clicked. That is very useful because we could do something like add a data attribute, such as data-price="", like so

```
<button data-price="100" class="buy">Buy Item 1</button>

<button data-price="200" class="buy">Buy Item 2</button>

<button data-price="300" class="buy">Buy Item 3</button>

<button data-price="400" class="buy">Buy Item 4</button>

<button data-price="500" class="buy">Buy Item 5</button>

<button data-price="600" class="buy">Buy Item 6</button>
```

<button data-price="700" class="buy">Buy Item 7</button>

<button data-price="800" class="buy">Buy Item 8</button>

<button data-price="900" class="buy">Buy Item 9</button>

<button data-price="1000" class="buy">Buy Item 10</button>

That allows you to go into the handleBuyButtonClick function and add the following code

function handleBuyButtonClick(event){

console.log('Buying the Item');

console.log(event.target.dataset);

}

Now when you click on a specific button, it should show you the dataset in the console. If you use dataset.price in the function log it will show you the price in the console. If you do log typeof event.target.dataset.price, you will see that the price is a string, so you need to convert it.

function handleBuyButtonClick(event){

console.log('Buying the Item');

console.log(parseFloat(event.target.dataset.price));

}

Now you will get a true number in the console.

What is the diferent between **event.target** and **event.currentTarget**?

function handleBuyButtonClick(event) { const button = event.target; // console.log(event.target);

console.log(event.currentTarget);

console.log(event.target === event.currentTarget);

}

Both them are exactly same. The difference come in when there are elements nested inside of the element that you are listening to. Take all the numbers in our buttons and wrap them in a strong tag like so

<button data-price="100" class="buy">Buy Item
1</button>

When you clicked the button event.targed is the thing that is actually got clicked and event.currentTarget is the thing that fired the event Listener. In most case you probably need to use event.currentTarget than even.target.

Propagation

It is possible to click on multiple things at a time and this is called propagation. When we clicked the strong tag, what happens is the event bubbles up it means you clicked strong tag meaning you are clicking on the button you are actually clicking on the body, the HTML tag, window and so on. The way we can prevent this is by a method called stopPropigation(). With in handleBuyButtonClick function add event.stopPropigation(). Now when you refresh the HTML page, if you click anywhere on the window, the window click event will fire, but if you click on the button, it will not. The windows listener does not fire because we stopped it.

This KeyWord

Lets see an example to describe this keyword.

```
photoEl.addEventListener("mousemove", function() {
console.log(e.currentTarget);

console.log(this);

});
```

The keyword ‘this’ is a special keyword in JavaScript it means it is a reserved word. The output of both the consoles listed above are same. If we called a method called addEventListener, look to the left of it and that is what this will be equal to. If you change the anonymous function that we used in photoEl as an arrow function this keyword is no longer useful as shown below.

```
photoEl.addEventListener("mousemove", (e) => {
console.log(e.currentTarget);

console.log(this);

});
```

Prevent Default and form Elements

We are going to discuss about preventDefault() that exists in the events when working with forms.

PreventDefault Method

There are few HTML elements that have a default functionality when they are used. Let's say you have an link to an html page, like so

```
<a class="web" href="https://webhub.com">Web Bos</a>
```

When you click the link, it takes us to the href, like we would expect.

Now you we can do is stop the default action of the browser and instead handle the event with JavaScript. In the JS file, select the link and then listen for a click on it, like so

```
const web = document.querySelector(".web");

web.addEventListener("click", function(event) {

 console.log(event);

});
```

When you click the link, you will see the event in the console for a split second before you are redirected to the next page. The default link is to change the page howevey, if you call event.preventDefault() method with in the eventListener it will stop default things to happen. Lets see an example

```
web.addEventListener("click", function(event) {

event.preventDefault();

const shouldChangePage = confirm( "This website might be malicious! Do you wish to proceed?" );

if (shouldChangePage) {

window.location = event.currentTarget.href;

 }

console.log(shouldChangePage);

});
```

event.currentTarget.href will give you the value of the href attribute which is the url you need to redirect the user to manually. Now when you

click the link, and hit okay in the alert dialogue that comes up in the browser, you will be redirected. If you were to hit "Cancel" rather than OK, you would not be redirected. The other way of doing it is

web.addEventListener('click', function(event){

const shouldChangePage = confirm("This website might be malicious! Do you wish to proceed?");

}

If(!shouldChangePage){

Event.preventDefault();

}

console.log(shouldChangePage);

});

Now preventDefault will only be called if the user does not click okay, in which case you do not want to redirect the user which preventDefault will prevent from happening. This gives us the exact same functionality, but this time you don't need to touch window.location. Another common one is submitting form elements, the best way to select a form element is to give a name rather than a class. If you want to select an element by it's name in JavaScript, you use an attribute selector. This is standard in CSS as well.

const signupForm = document.querySelectorAll('[name="signup"]);

Take the sign-up form and add an event listener. However, instead of listening for a click, listen for a submit event.When someone submits a form, the event will fire and give you access to the form, like so

signupForm.addEventListener("submit", function(event){

console.log(event);

event.preventDefault();

});

You could just put a required attribute on the input boxes like so <input required type="checkbox" id="agree" name="agree">. To grab the name and the email id of the form we can do the following.

```
signupForm.addEventListener("submit", function(event) {
event.preventDefault();

console.log(event.currentTarget.name);
console.log(event.currentTarget.email);

});
```

That will give you the actual values that are in the input boxes in the console.

Accessibility Gotchas and Keyboard codes

Making a web site accessible means that you are making it useful to everyone regardless of what defect it have or what input device they are having or what situation they are in, If HTML is markup correctly they are accessible. However what developers do is they often develop code that are used only with mouse clicks, and which may not be accessible to all the users. This topic will cover some pitfalls that happen in JavaScript.

Difference between Buttons and Links

The difference between buttons and links are big accessibility issues. Buttons are meant to be use actions inside of the application while Links are used to change the page. Links are not meant to be used where buttons are. We often see codes like Save.

Then the developer will add a click event via JavaScript to a tag and preventDefault, and then perform some sort of action. However, that is not a use-case for a link because that "Save" click doesn't redirect you to anywhere, and thus is not a link. For example, in twitter with out getting logged, if you try to click tweets and replies you will see a pop message saying you are not logged in. That is an example of a valid use case for using prevent default with a link. When you need to do something, don't mix up buttons and links.

Keyboard Accessibility

Elements that are not accessible to keyboard should not have click event, unless they need to. For example, let us say we have an image like

```
<img class="photo" src="https://picsum.photos/200" alt="Nice">
```

Via JavaScript, select the photo and add a click event listener.

```
const photo = document.querySelector(".photo");

photo.addEventListener("click", function() {

console.log("you clicked the photo");

});
```

here are plenty of valid use cases for clicking on a photo. For example, maybe it will open a larger version of that photo, or you want to draw on the photo or zoom it in. If you refresh the HTML page and click on the photo, you will see "You clicked the photo" logged every time you click it. If you through your mouse away, using the tab button on your keyboard, you should be able to tab through everything on the browser. However, you will not be able to click on the photo with only the keyboard. That is a problem because if the user doesn't have a mouse or they are unable to use a mouse, then they can't use that part of your website. An easy solution is giving the element the following attribute: role="button". If you want something to be button when it is not, you should use role to the div as shown below.

```
<div role="button" tabindex="0">click me</div>
```

You need to add tabindex and set it to ‘0’ when you add role=’button’, so that the user can tab with keyboard. For the image, you could give it a role of button, but it's easier to put the image actually inside a button, like so

```
<button> <img class="photo" src="https://picsum.photos/200" alt="Nice" /></button>
```

If you modified the code like so instead,

```
<div role="button" tabindex="0">Click me</div>

<img class="photo" tabindex="0" src="https://picsum.photos/200" alt="Nice" />
```

Now if you tab using your keyboard, you can go back and forth between the image and the div with button role. If you focus on the image like shown above and hit the enter key, unlike some other buttons and elements, it will not trigger a click. This is because if you see the code with evenListener above we did not name they function, if you change the anonymous function to function name this will work as shown below.

```
function handlePhotoClick() {

console.log("you clicked the photo");

}

photo.addEventListener("click", handlePhotoClick);

photo.addEventListener("keyup", handlePhotoClick);
```

Now, when you go to the image, you can click on over to it. You might notice that when upi tab to the image, the console is logging "You clicked the photo", even though you haven't actually clicked it yet (we only care about the enter key).

How can you only listen for the enter key?

The event has a bunch of information about this. The event object should have event.key and event.keycode. event.key is more modern and supported in more browsers. Every single key on an event will give you a bunch of information.

There is event.key, event. which, event.code and event.key. event.key is the best way to go about it. Now what we can say is if the event is a click or if the event.key equals enter like so

```
function handlePhotoClick(event) {

 if (event.type === "clicked" || event.key === "Enter") {

console.log("you clicked the photo");

}}
```

Now if you try tabbing and then pressing enter when on the image, you will see "You clicked the photo" logged, meaning it is accessible.