TASK

Programming in JavaScript II: Event Handling

Visit our website

Introduction

WELCOME TO THE EVENT HANDLING TASK!

Now that you know some HTML and JavaScript we can use it to make our web pages more interesting and useful! In this task, you will learn how to apply JavaScript to your HTML. To be able to do this you will need to understand two *very* important concepts including:

1. What *functions* are and how to create your own JavaScript functions.
2. How to write JavaScript functions that respond to DOM *events*.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact an expert code reviewer to get support on any aspect of your course.

The best way to get help is to login to Discord at **https://discord.com/invite/hyperdev** where our specialist team is ready to support you.

Our expert code reviewers are happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

EVENTS

JavaScript is often used to handle *events* that occur on a website. An event is basically an action that occurs that your program responds to. You have encountered DOM (Document Object Model) events. DOM events are either things that a user does, such as clicking a button or entering text into a textbox or actions caused by the browser, such as when a web page has completely loaded.

The most common events you'll deal with when getting started are:

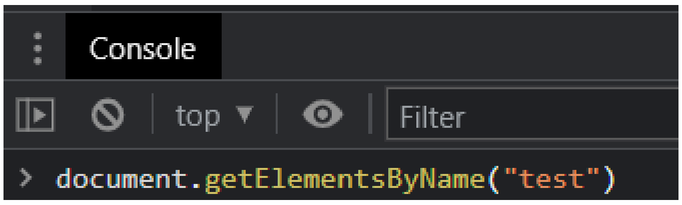
Event	Description
`onchange`	Some HTML element has been modified
`onclick`	An HTML element has been clicked on
`onmouseover`	An HTML element was hovered over
`onmouseout`	Mouse cursor moves off HTML element
`onkeydown`	A keyboard button is pressed
`onload`	The HTML page has finished loading

Other events are listed and explained **here**.

Every DOM element object in Javascript has the DOM event methods available to them. We can see this by doing the following. Copy the following code into an HTML file and open it in the browser:

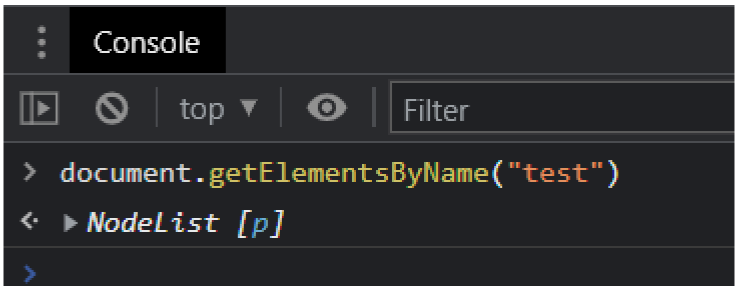
```
<!DOCTYPE html>

<html>

<body>
    <p name="test">This is a test</p>
</body>


</html>
```

Open the developer tools in your browser and enter the following command in the console:



```
> document.getElementsByName("test")
```

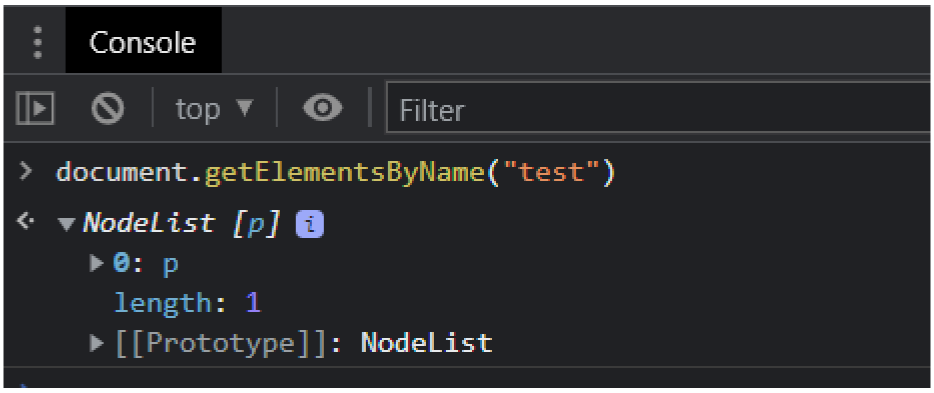
The above DOM method will retrieve all element objects that have the name "test" and put them in an array. You will get the following output after running this DOM method:



```
> document.getElementsByName("test")
<· ▶ NodeList [p]
>
```

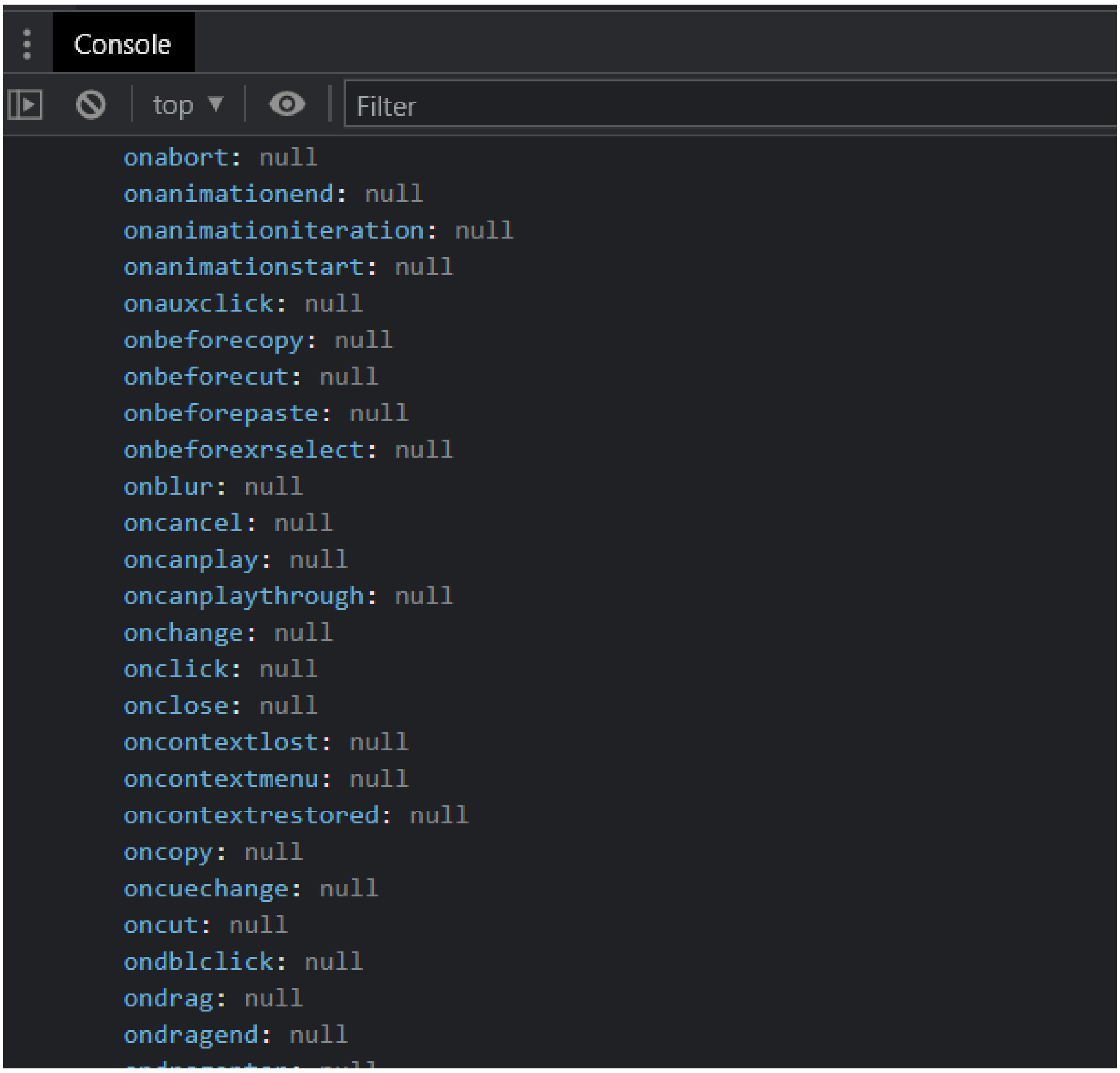
You will see that we have a NodeList array with the single paragraph object that we named "test".

When we expand the NodeList by clicking on the arrow, we get the following:



```
> document.getElementsByName("test")
<· ▼ NodeList [p] i
    ▶ 0: p
      length: 1
    ▶ [[Prototype]]: NodeList
```

If we click on the arrow next to the 0 index element (our paragraph object), we can see all of the attributes and methods attached to the paragraph object. It is very important to understand that every element in the DOM is an object. Here is a snippet of the available methods:



```
onabort: null
onanimationend: null
onanimationiteration: null
onanimationstart: null
onauxclick: null
onbeforecopy: null
onbeforecut: null
onbeforepaste: null
onbeforexrselect: null
onblur: null
oncancel: null
oncanplay: null
oncanplaythrough: null
onchange: null
onclick: null
onclose: null
oncontextlost: null
oncontextmenu: null
oncontextrestored: null
oncopy: null
oncuechange: null
oncut: null
ondblclick: null
ondrag: null
ondragend: null
```

You will notice that all of the event methods have the value null. This is because there are no functions attached to them as yet.

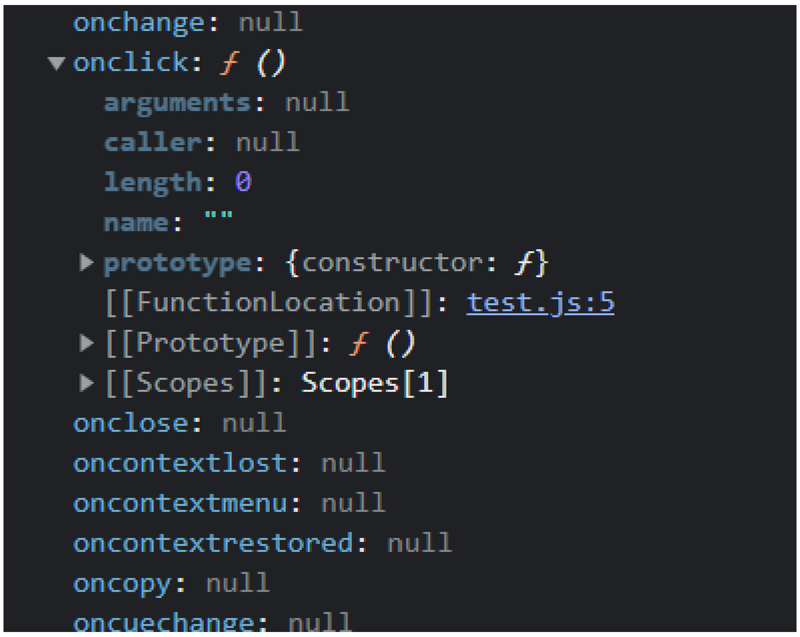
Let's attach a function to one of these event methods. We can create a JS file named "test.js" with the following code:

```
// grabbing the object at the 0 index of the array and seeting to avariable.
var selectedElement = document.getElementsByName('test')[0];

// assigning an anonymous function to the onclick method as a callback
function
selectedElement.onclick = function() {
    console.log("The paragraph was clicked");
};
```

We can then change your HTML file to the following as we import the external Javascript:

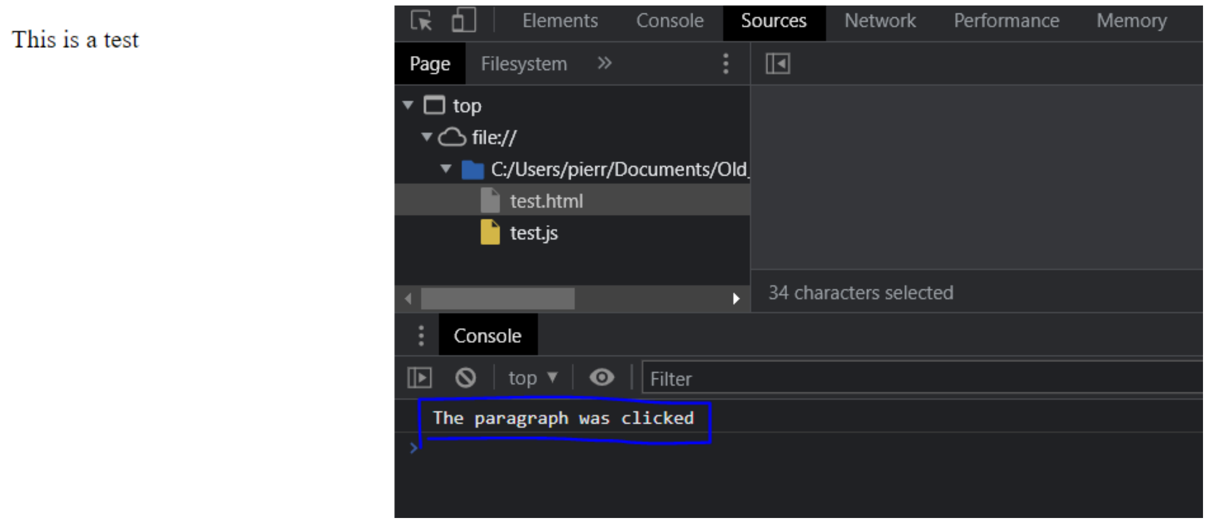
```html
<!DOCTYPE html>

<html>

<body>
    <p name="test">This is a test</p>

    <script type="text/javascript" src = "test.js"></script>
</body>



</html>
```

If we open the HTML file now and look again at the paragraph object in the console, we will see that a function has now been assigned to the onclick method as a callback function. When the method is called, the callback function is executed.



```
    onchange: null
  ▼ onclick: ƒ ()
      arguments: null
      caller: null
      length: 0
      name: ""
    ▶ prototype: {constructor: ƒ}
      [[FunctionLocation]]: test.js:5
    ▶ [[Prototype]]: ƒ ()
    ▶ [[Scopes]]: Scopes[1]
    onclose: null
    oncontextlost: null
    oncontextmenu: null
    oncontextrestored: null
    oncopy: null
    oncuechange: null
```

If we click on the paragraph in the browser, we will get the following output in the console:



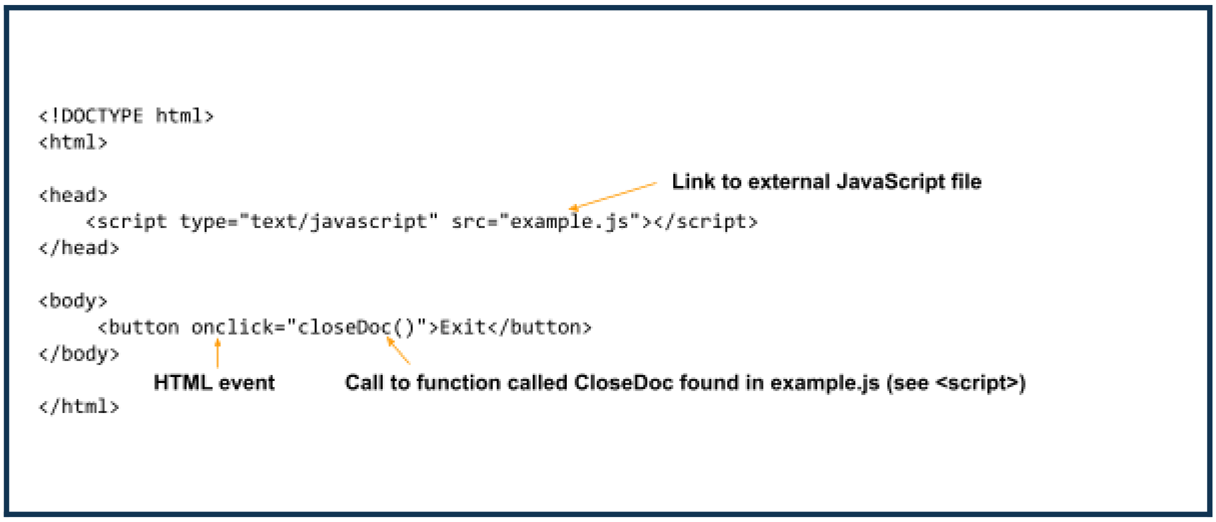
```
The paragraph was clicked
```

We can write JavaScript code that tells your browser what you want it to do when it realises that a certain event has occurred. This code could be written in a JavaScript function.

JAVASCRIPT FUNCTIONS AS EVENT HANDLERS

As stated, JavaScript is often used to handle events that occur on a website. To do this, simply identify the DOM event and specify the name of the JavaScript function that you want to call when the event is triggered. See the example below:

```html
<!DOCTYPE html>
<html>

<head>
    <script type="text/javascript" src = "example.js"></script>
</head>

<body>
    <button onclick="closeDoc()">Exit</button>
</body>


</html>
```



```
<!DOCTYPE html>
<html>

<head>
    <script type="text/javascript" src="example.js"></script>
</head>

<body>
     <button onclick="closeDoc()">Exit</button>
</body>

</html>
```

Link to external JavaScript file
HTML event
Call to function called CloseDoc found in example.js (see <script>)

The above is an example of an HTML file where we want our browser to listen for a certain event. Notice that in the `<head>` element, there is a `<script>` element that refers to the external JavaScript file called **example.js**. Also, notice the button element: here we instruct the browser to call the function named `closeDoc()` when the button is clicked. The `closeDoc()` function is declared in the external Javascript file, **example.js**.

The JavaScript file called **example.js** would contain the following function which will be called and executed when the button mentioned in the HTML page is clicked:

```
function closeDoc(){
      alert("You are closing this page!");
      window.close();
}
```

Copy the above HTML code into a file named “test.html” and the Javavscript code into a file named example.js in the same folder. Open the HTML file in the browser, click on the exit button and observe what happens. You will first see the alert. Once the alert is closed, the window.close() function is called. This is a built-in Javascript function that closes the current tab that you are working in.

We could also add a function to an event in an element object in the following manner using DOM manipulation and event listeners:

Create an HTML file named “example.js” and add the follwing code to the file. We are creating 2 elements. A button and a paragraph element. When the button element is clicked, we want the paragraph to display some text.

```html
<!DOCTYPE html>
<html>

<body>
    <button id="showMoreButton">Show More</button>
    <p id="info"></p>

    <script type="text/javascript" src = "example.js"></script>
</body>

</html>
```

We can then put the following code in a file named “example.js” and this is where the magic happens.

```javascript
//grabbing the button and paragraph elements and assigning those objects to
variables
let showMoreButton = document.getElementById('showMoreButton');
let information = document.getElementById('info');

//this function will append text to the paragraph element when called
function showMoreDetails(){
    information.innerHTML += "some more information";
}

/*We use the addEventListener method to assign the function to the
button element to be triggered when it is clicked.
Note that when using an event listener, we do not use the "on" portion
of the onclick method name, but just 'click'*/
showMoreButton.addEventListener('click', showMoreDetails);
```

THE EVENT OBJECT

JavaScript stores events as Event objects with their related data and functionalities as properties and methods. When an event is triggered, the event object can be passed as an argument to the event handler function. This object has many useful properties that are important to know about and explore. It can prove useful for debugging such as identifying what element triggered an event or even for other things such as taking separate actions based on whether a button was clicked whilst holding another key or not for example.

It is important to note that the event object is an optional argument and that the event handler function does not need it to trigger logic.

In codebases, you may see the event object argument named as one of the following:

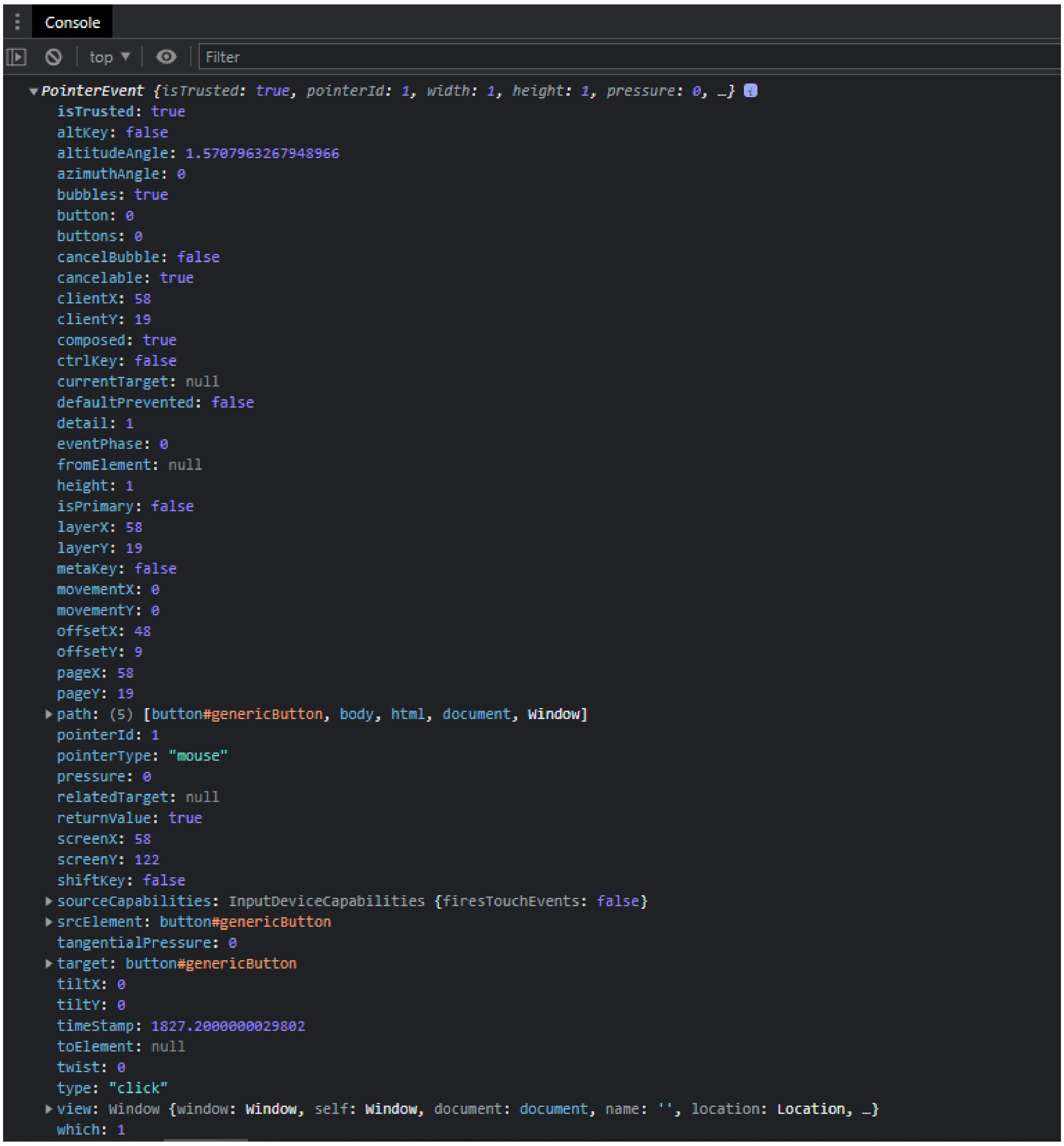
- “e”
- “evt”
- or “event”

These are all just synonyms for the same thing, but the usual convention that is most commonly found is simply the letter e.

Let’s take a look at a code example:

```
<html>
    <head>
        <title>My Webpage</title>
    </head>

    <body>
        <button id="genericButton">Click This</button>

        <script>
            //retrieving the button element
            let button = document.querySelector("#genericButton");
            //passing event object "e" as argument to event handler function
            button.addEventListener("click", function (e) {
                console.log(e); //outputting the event object to the console
            })
        </script>
    </body>
</html>
```

In the code snippet above, we are passing an event object as an argument to an anonymous function being used to trigger logic when the button is clicked. If you paste this code into an html file, open in the browser, click the button and open the developer tools, you will find the following output in the console:



```
▾PointerEvent {isTrusted: true, pointerId: 1, width: 1, height: 1, pressure: 0, …}
   isTrusted: true
   altKey: false
   altitudeAngle: 1.5707963267948966
   azimuthAngle: 0
   bubbles: true
   button: 0
   buttons: 0
   cancelBubble: false
   cancelable: true
   clientX: 58
   clientY: 19
   composed: true
   ctrlKey: false
   currentTarget: null
   defaultPrevented: false
   detail: 1
   eventPhase: 0
   fromElement: null
   height: 1
   isPrimary: false
   layerX: 58
   layerY: 19
   metaKey: false
   movementX: 0
   movementY: 0
   offsetX: 48
   offsetY: 9
   pageX: 58
   pageY: 19
  ▸path: (5) [button#genericButton, body, html, document, Window]
   pointerId: 1
   pointerType: "mouse"
   pressure: 0
   relatedTarget: null
   returnValue: true
   screenX: 58
   screenY: 122
   shiftKey: false
  ▸sourceCapabilities: InputDeviceCapabilities {firesTouchEvents: false}
  ▸srcElement: button#genericButton
   tangentialPressure: 0
  ▸target: button#genericButton
   tiltX: 0
   tiltY: 0
   timeStamp: 1827.2000000029802
   toElement: null
   twist: 0
   type: "click"
  ▸view: Window {window: Window, self: Window, document: document, name: '', location: Location, …}
   which: 1
```

There are many properties in this object that tell us about the event such as where the event occurred on the screen, the type of event, the time of the event, whether the control key or shift key was pressed at the time of the event etc.

You can find an explanation of common properties in the event object and what they mean in this resource.

Let's take a quick look at an example of what can be done with the event object:

```
<html>
    <head>
        <title>My Webpage</title>
    </head>

    <body>
        <button id="genericButton">Click This</button>

        <script>
            //retrieving the button element
            let button = document.querySelector("#genericButton");
            //passing event object "e" as argument to event handler function
            button.addEventListener("click", function (e) {
                /* check whether the shift key was pressed or not
                shiftKey property would be true if it was*/
                if (e.shiftKey) {
                    console.log("The shift key was pressed")
                } else {
                    console.log("The shift key was not pressed")
                }
            })
        </script>
    </body>
</html>
```

Copy and paste the above code in an HTML file, open it in the browser and check the console for what happens if a shift key is pressed while clicking on the button or not. You can imagine the possibilities here.

For an advanced understanding of events in Javascript, look at this **resource** on event bubbling and capturing.

Instructions

Open **example.js** in Visual Studio Code and read through the comments before attempting these tasks.

Getting to grips with JavaScript takes practice. You will make mistakes in this task. This is completely to be expected as you learn the keywords and syntax rules of this programming language. It is vital that you learn to debug your code. To help with this remember that you can:

- Use either the JavaScript console or Visual Studio Code (or another editor of your choice) to execute and debug JavaScript in the next few tasks.
- Remember that if you really get stuck, you can contact an expert code reviewer for help.

Compulsory Task

Follow these steps:

- In this task, you will be required to modify your files from the previous task.
- Follow these steps:
    - **main.js:** Modify your function which displays each item as list elements as follows:
        - Each created list element should contain a span element with a class named `close`. Each span element should have a value of '\u00D7', which represents the character 'x' in unicode.
        - As the last line of this function, call a function (described below) to delete an item from the shopping list.
        - Create a function which will update the grocery items array by getting the value of the text in the `<input>` tag and adding it to the array.
        - If the input text field is empty, display an alert to the user indicating that they should insert an item. Else, add the input text to the array.
        - Once the item has been added to the array or the alert displayed, reset the input text's value to an empty string.
        - As the last line of this function, call the function created in the previous task to display the updated array items.
        - Create a function to delete items from both the array and the Shopping List display.
        - Add a click event listener to each `<span>` element with a `close` class — when the event is triggered, delete the item from the array and set the display style to `none` for the specific list parent element.
        - Add a click event listener to the html element with the ID of `itemList`.
        - If the event's tag name is a list tag, toggle a `checked` class on the event element.
        - Add a key up event listener to the HTML element with an `input` ID.

        - If the event key code is equal to '13' (the key code for the 'Enter' key), trigger a `click()` function on the element with an `addButton` ID.
    - **CSS:**
        - All elements with a `checked` class should have styling which indicates that the item has been checked off the shopping list.
    - **HTML:**
        - Add an `onclick` attribute to the HTML element with an `addButton` ID. When this element is clicked, the function which updates the shopping list should be called.

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we've done a good job?

Click here to share your thoughts anonymously.