**DOM EVENTS WITH JAVASCRIPT**

**JavaScript and HTML**

HTML defines the structure of a web page by using page elements as the building blocks. However, HTML by itself can not produce web page interactivity, that's where JavaScript comes in.

Below, we see a post-it with a typical stick figure on it. We can think of this as the HTML, with the head, body, and limbs as the elements on the page:



In web development, CSS provides the style to our HTML structure. Below, the stick figure is now dressed in a nice tuxedo:



If HTML and CSS provide structure and style in this analogy, JavaScript provides interactivity, allowing our stick figure to move. Below, the stick figure moves, swaying up and down, thanks to JavaScript: now, it’s a moveable.



**The <script> tag**

The <script> element allows you to add JavaScript code inside an HTML file. Below, the <script> element embeds valid JavaScript code:

<html>

<head>

<title></title>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>

</head>

<body>

<h1>This is an embedded JS example</h1>

<script>

var myFunction = function()

{

alert ( "Hello World");

}

window.onload= function(){

myFunction();

}

</script>

</body>

</html>

# The src attribute

**Linking code** is preferable because of a programming concept called Separation of Concerns (SoC). Instead of having messy code that is all in the same file, web developers separate their code into different files, making each “concern” easier to understand and more convenient when changes must be made.

Now, instead of writing JavaScript in our HTML file, we are going to write it in its own file, and then reference this code with a file path name. We will do this using the src attribute!

If the file is in the same project folder, the srcvalue will be a relative path name. Below is an example of a relative pathname to a JavaScript file.

<script src=”/exampleScript.js”> </script>

# How are scripts loaded?

A quick recap: the <script> element allows HTML files to load and execute JavaScript. The JavaScript can either go embedded inside of the <script> tag or the script tag can reference an external file. Before we dive deeper,

let’s take a moment to talk about **how browsers parse HTML files into web pages.** This informs **where to include a <script>** element inside your HTML file.

Browsers come equipped with HTML parsers that help browsers render the elements accordingly. ***Elements, including the <script> element, are by default, parsed in the order they appear in the HTML file.*** When the HTML parser encounters a <script> element, it loads the script then executes its contents before parsing the rest of the HTML. The two main points to note here are that:

* The HTML parser does NOT process the next element in *the HTML file* until it loads and executes the <script> element, thus leading to a delay in load time and resulting in a poor user experience.
* Additionally, scripts are loaded sequentially, **so if one script depends on another script, they should be placed in that very order** inside the HTML file.

The image below displays two scripts being loaded. The first script makes a Watering Can appear, the second script makes a Flower appear. This shows how scripts are loaded sequentially, and how they pause the HTML parser, which is why "Blooming" appears at the end.



# Defer attribute

When the HTML parser comes across a <script>element, it stops to load its content. Once loaded, **the JavaScript code is executed** and the **HTML parser proceeds to parse the next element in the file**. This can result in a **slow load time** for your website. HTML4 introduced **the defer and async attributes** of the <script> element to address the user wait-time in the website based on different scenarios.

The defer attribute **specifies scripts should be executed after the HTML file is completely parsed.** When the HTML parser encounters a <script> element with the defer attribute, it loads the script but **defers the actual execution of the JavaScript until after it finishes parsing the rest of the elements in the HTML file.**

Here is an example of the defer tag: ```html

```

When is defer useful?

When a script contains functionality that requires interaction with the DOM, the defer attribute is the way to go. This way, it ensures that the entire HTML file has been parsed before the script is executed.

Index.html

<!DOCTYPE html>

<html>

<head>

<script id="blue" src="turnBlue.js" ></script>

<script id="yellow" defer src="turnYellow.js"></script>

<style>

body {

background-color: white;

font-family: 'Raleway', sans-serif;

}

p {

text-align: center;

font-size: 10vw;

color: pink;

top: 50%;

left: 50%;

position: absolute;

transform: translateX(-50%) translateY(-50%);

}

</style>

</head>

<body>

 <p class="centered" id="logo">Codecademy</p>

</body>

</html>

Turnblue.js

var elem = document.getElementById('logo');

  elem.style.color = "cyan";

turnYellow.js

var elem = document.getElementById('logo');

  elem.style.color = "yellow";

# Async attribute

The async attribute loads and executes the script asynchronously with the rest of the webpage. This means that, similar to the defer attribute, the HTML parser will continue parsing the rest of the HTML as the script is downloaded in the background. However, with the async flag, the script will not wait until the entire page is parsed: it will execute immediately after it has been downloaded. Here is an example of the asynctag:

<script src="example.js" async> </script>```

When is it useful?

Async is useful for scripts that are independent of other scripts in order to function accordingly. Thus, if it does not matter exactly at which point the script file is executed, asynchronous loading is the most suitable option as it optimizes web page load time.

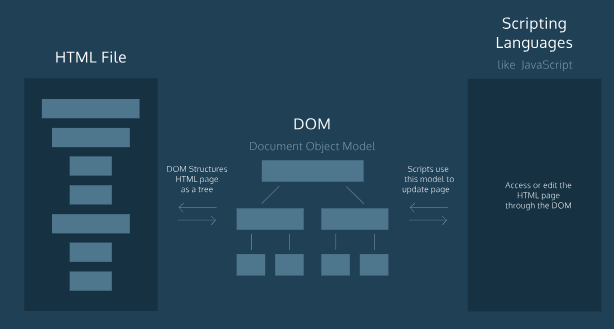
**What is the DOM?**

The *Document Object Model*, abbreviated DOM, is a powerful tree-like structure that allows programmers to conceptualize hierarchy and access the elements on a web page.

Note: There are other types of documents, such as XML and SVG, that are also modeled as DOM structures.

The DOM is a language-agnostic structure implemented by browsers to allow for web scripting languages, like JavaScript, to access, modify, and update the structure of an HTML web page in an organized way.

For this reason, we like to think of the DOM as the link between an HTML web page and scripting languages.



# The DOM as a Tree Structure

You’re already familiar with the concept of family trees: these charts represent the familial relationships amongst the descendants of a given family name.

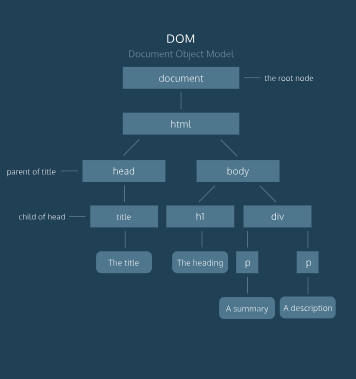
The DOM tree follows similar logic to that of a family tree. A family tree is made up of family members and their relationships to the family name. In computer science, we would call each family member a node.

We define a node as an intersecting point in a tree that contains data.

In the DOM tree, the top-most node is called the root node, and it represents the HTML document. The descendants of the root node are the HTML tags in the document, starting with the <html>tag followed by the <head> and <body> tags and so on.

# Parent Child Relationships in the DOM

A parent node is the closet connected node to another node in the direction towards the root.A child node is the closest connected node to another node in the direction away from the root.Knowing these terms will allow you to understand and discuss the DOM as a tree-like structure. In fact, you will also see this terminology used when referring to the nesting structure of HTML code.



Example : DOM Structure

<!DOCTYPE html>

<html>

    <head>

<title> the title </title>

    </head>

    <body>

<h1>The heading</h1>

        <div>

    <p>A summary</p>

    <p>A description</p>

</div>

    </body>

</html>

# Nodes and Elements in the DOM

A node is an intersecting point in a tree that also contains data.There are nine different types of node objects in the DOM tree.

In our diagram, the node objects with the sharp-edge **rectangles are of the type**[**Element**](https://developer.mozilla.org/en-US/docs/Web/API/Element)**,** while **the rounded edge rectangles are of type**[**Text**](https://developer.mozilla.org/en-US/docs/Web/API/Text), because they represent the text inside the HTML paragraph elements.

When trying to *modify a web page*, the script will **mostly interact with the DOM nodes of type element.**

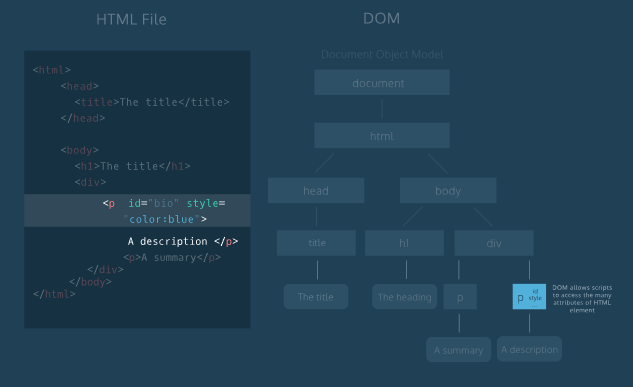
Elements are the building units of HTML web pages, they contain everything between an opening tag and a closing tag. If the tag is a self-closing tag, then that is the element itself.

# Attributes of Element Node

DOM element nodes model elements in an HTML document.

Much like an element in an HTML page, **the DOM allows us to access a node's attributes, such as its class, id, and inline style.**

If we were accessing that element node in our script, the DOM would allow us to tweak each of those attributes, or simply access them to check their value in the code.



# The document keyword

Now, we will focus on some of the most useful methods and properties of the [DOM Interface](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model) <https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model> in JavaScript. This interface is implemented by every modern browser.

First things first! **The document object** in JavaScript is the door to the DOM structure. The **document allows you to access the root node of the DOM tree.**

Before you can access a specific element in the page, first you must access the document structure itself. The document allows scripts to access children of the DOM as properties.

For example, if you wanted to access the <body>element in your script, you could access it as a property of the document by typing document.body. This property will return the body element of that DOM.

Similarly, you could access the <title> element with the .title property. See a <https://developer.mozilla.org/en-US/docs/Web/API/Document>

 of all document properties.

# Tweak an Element

When using the DOM in your script to access an HTML element, you also have access to all of that element's properties.

This includes the ability to modify the contents of the element as well as its attributes and properties— that can range from modifying the text inside a p element to assigning a new background color to a div.

You can **access and set the contents of an element** with the .innerHTML property.

For example, the following code reassigns the inner HTML of the body element to the text 'The cat loves the dog':

document.body.innerHTML = 'The cat loves the dog.';

The .innerHTML property can also add any valid HTML, including properly formatted elements. The following example assigns an h2 element as a child inside the <body> element:

document.body.innerHTML = '<h2>This is a heading</h2>';

# Select and Modify Elements

we accessed the webpage elements with the document keyword!

What if we wanted to select a specific element?

The DOM interface allows us to access a specific element with CSS selectors.  we can also use these same selectors to access DOM elements with our script! Selectors can include the name of the tag, a class, or an ID.

The .querySelector() method allows us to specify a **CSS selector and then returns the first element that matches that selector**. The following code would return the first paragraph in the document.

document.querySelector('p');

var a= document.querySelector('p');

a.innerHTML="HELLO";

You can also use other CSS selectors such as an element's . class or its # ID.

Another option, if you want to access elements directly by their id, you can use the aptly named .getElementByID() function:

document.getElementById('bio').innerHTML = 'The description';

The example chains so that it selects the element with an ID of 'bio' and set its .innerHML to the text 'The description'.

**Example:**

<html>

    <head>

        <title>Popular Google Searches</title>

    </head>

    <body>

<h1>Top 5 most searcheed TV shows in 2016</h1>

<ol>

<li id="first">Stranger Things</li>

<li id="second">Westworld</li>

<li id="third">Luke Cage</li>

<li id="fourth">Game of Thrones</li>

<li id="fifth">Black Mirror</li>

        </ol>

    </body>

<script src="./main.js"></script>

</html>

**Js file**

var a= document.querySelector('h1');

a.innerHTML="Most popular TV show searches in 2016";

document.getElementById('fourth').innerHTML = "Fourth element";

# Style an element

Another way to modify an element is by changing its CSS style. The .style property of a **DOM element provides access to the inline style of that HTML tag**.

The syntax follows an element.style.propertyformat, with the property representing a CSS property.

For example, the following code selects the first element with a class of blue and assigns blue as the background-color:

let blueElement = document.querySelector('.blue');

blueElement.style.backgroundColor = 'blue';

Unlike CSS, the DOM style property does not implement a hyphen such as background-color, but rather camel case notation backgroundColor.

The following chaining syntax would also work:

document.querySelector('.blue').style.fontFamily = 'Roboto';

document.querySelector('body').style.background-color="#201F2E";

# Create and Insert Elements

Just as the **DOM allows** scripts to **modify existing elements, it also allows for the creation of new ones.** The .createElement(tagName) method creates a new element based on the specified tag name. However, it does not append it to the document. It creates an empty element with no inner HTML.

In order to create an element and add it to the web page, you must assign it to be the child of an element that already exists on the DOM. We call this process appending. The .appendChild()method will add a child element as the last child node.

The following code creates a new paragraph element, adds text to the new element's innerHTML, and appends it to the body of the document:

let paragraph = document.createElement('p');

paragraph.innerHTML = 'The text inside paragraph';

document.body.appendChild(paragraph);

Unlike .innerHTML the .appendChild() method does not replace the content inside of the parent, in this case body. Rather, it appends the element as the last child of that parent.

**Ques:**

1. Create a li element in the document and save it to a variable.
2. Assign the new element an id of "oaxaca" by using the .id property on the new variable.
3. Assign the element the text "Oaxaca, Mexico" by using the .innerHTMLmethod.
4. Append the new element you created as the last child of the list with the ID more-destinations. Scroll to the bottom of the page in the browser to see your new element.

         let listItem = document.createElement('li');

         listItem.id="oaxaca"; //assign id to the new element

         listItem.innerHTML = "Oaxaca, Mexico"; //now, insert txt to element

         document.getElementById('more-destinations').appendChild(listItem);

//time to append in to the body of html element but, with the use of html document existing elemnt id we can add this new elemnt over there.

# Remove an Element

In addition to modifying or creating an element from scratch, the DOM also allows us by use of the .removeChild()method that removes a specified child from a parent.

Because the .querySelector() method returns the first paragraph, the following code would remove the first paragraph in the document:

let paragraph = document.querySelector('p');

document.body.removeChild(paragraph);

It's possible to also specify a different parent with the .querySelector() method, as long as you remove an element nested within that parent element.

If you want to hide an element because it does not need to be loaded initially, the .hiddenproperty allows you to hide it by assigning it as true or false:

document.getElementById('sign').hidden = true;

The code above did not remove the element from the DOM, but rather hid it. This is not the same as setting the CSS visibility property to hidden. For a list of the best use cases for this property, read a list in the <https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/hidden>

**Ques**

1. Select the element with the ID oaxaca and save it to a variable.
2. Select its parent, assigned an ID of more-destinations.
3. Remove the element using the .removeChild() method and passing in the variable containing the oaxacaelement.

    const parent = document.querySelector("#more-destinations");

         const child = document.querySelector("#oaxaca");

         parent.removeChild(child);

# Interactivity with onclick

You can add interactivity to DOM elements by assigning a function to run based on an [event](https://developer.mozilla.org/en-US/docs/Web/Events).

<https://developer.mozilla.org/en-US/docs/Web/Events>

Events can include anything from a click to a user mousing over an element.

The .onclick property allows you to assign a function to run on a click event on an element:

let element = document.getElementById('interact');

element.onclick = function() { element.style.backgroundColor = 'blue' };

**Example :**

* Modify the body of the turnButtonRed()function so that it modifies the button as follows:

1. Assigns the .style.backgroundColorto 'red'
2. Assigns the style.color to 'white'
3. Modifies the .innerHTML to 'Red Button'

* Add interactivity to the button element by adding a function that turns its .backgroundColor to 'red' and its font color to 'white' when the button is clicked.

**Way 1-**

         let element = document.querySelector("button");

         element.onclick = function turnButtonRed() {

         element.style.backgroundColor = 'red';

         element.style.color = 'white';

         element.innerHTML="Red Button"

         }

**Way-2**

         let element = document.querySelector("button");

         function turnButtonRed (){

             element.style.backgroundColor = "red";

         element.style.color = "white";

         element.innerHTML = "Red Button";

         }

         element.onclick = turnButtonRed;

**Project : I**

**Index.html**

<!DOCTYPE html>

<html>

<head>

<title>Chore Door!</title>

<link href="./style.css" rel="stylesheet" type="text/css">

<link href="https://fonts.googleapis.com/css?family=Work+Sans" rel="stylesheet" type="text/css">

</head>

<body>

<!-- HEADER WITH LOGO-->

<div class="header">

<img src="https://s3.amazonaws.com/codecademy-content/projects/chore-door/images/logo.svg">

</div>

<!-- TITLE WITH STARs-->

<div class="title-row">

<img src="https://s3.amazonaws.com/codecademy-content/projects/chore-door/images/star.svg">

<p class="instructions-title">Instructions</p>

<img src="https://s3.amazonaws.com/codecademy-content/projects/chore-door/images/star.svg">

</div>

<!-- table is a block element like div so we will make table outside div -->

<table class="instructions-row">

<tr>

<td class="instructions-number">1</td>

<td class="instructions-text">Hiding behind one of these doors is the ChoreBot.</td>

</tr>

<tr>

<td class="instructions-number">2</td>

<td class="instructions-text">Your mission is to open all of the doors without running into the ChoreBot.</td>

</tr>

<tr>

<td class="instructions-number">3</td>

<td class="instructions-text">If you manage to avoid the ChoreBot until you open the very last door, you win!</td>

</tr>

<tr>

<td class="instructions-number">4</td>

<td class="instructions-text">See if you can score a winning streak!</td>

</tr>

</table>

<!-- first main div-->

<div class="door-row">

<!-- first door-->

<img id="door1" class="door-frame" src="https://s3.amazonaws.com/codecademy-content/projects/chore-door/images/closed\_door.svg" >

<!-- Second door-->

<img id="door2" class="door-frame" src="https://s3.amazonaws.com/codecademy-content/projects/chore-door/images/closed\_door.svg" >

<!-- Third door-->

<img id="door3" class="door-frame" src="https://s3.amazonaws.com/codecademy-content/projects/chore-door/images/closed\_door.svg" >

</div>

<!--buttom create here -->

<div id="start" class="start-row">Good Luck</div>

<!--Score Board Created Here -->

<table class="score-row">

<tr>

<th class="score-text">Current streak:</th>

<th class="score-text">Best streak:</th>

</tr>

<tr>

<th class="score-box" id="score-number"></th>

<th class="score-box" id="high-score-number"></th>

</tr>

</table>

<!--make connection between html and js before closing of body tag -->

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

</body>

</html>

**Style.css**

body {

background-color: #010165;

margin: 0px;

}

/\*header selector\*/

.header{

background-color:#00ffff;

text-align:center;

}

/\*Two stars with title\*/

.title-row{

margin-top:42px;

margin-bottom:21px;

text-align:center;

}

.instructions-title{

display:inline;

font-size : 18px;

color : #00ffff;

font-family : 'Work Sans';

}

/\*TABLE --->INSTRUCTION NUMBERS AND TEXT\*/

.instructions-row{

margin : 0 auto;

width : 400px;

}

.instructions-number{

padding-right : 25px;

font-family : 'Work Sans';

font-size : 36px;

color : #00ffff;

}

.instructions-text{

padding : 10px;

font-family : 'Work Sans';

font-size : 14px;

color : #ffffff;

}

/\* FOR ALL DOORS\*/

.door-row{

text-align:center;

}

.door-frame{

cursor:pointer;

padding:10px;

}

/\*BUTTON\*/

.start-row{

margin :auto;

width : 120px;

height : 43px;

font-family : 'Work Sans';

background-color : #eb6536;

padding-top : 18px;

font-size : 18px;

text-align : center;

color : #010165;

margin-bottom : 21px;

cursor : pointer;

}

/\*ALL ABOUT SCORE BOX\*/

.score-row {

text-align: center;

width: 100%;

}

.score-text {

margin-right: 17px;

padding: 5px;

display: inline;

font-family: 'Work Sans';

font-size: 12px;

letter-spacing: 1.1px;

color: #ffffff;

}

.score-box {

margin-top: 9px;

display: inline-block;

padding: 8px;

margin: 18px;

margin-right: 29px;

width: 59px;

background-color: #00ffff;

font-family: 'Work Sans';

font-size: 45px;

}

**Script.js**

let closedDoorPath = "https://s3.amazonaws.com/codecademy-content/projects/chore-door/images/closed\_door.svg";

let currentlyPlaying = true;

//DOOR 1 CREATED

let doorImage1 = document.getElementById('door1');

//Now make the closed door image change when you click it so that you see an open door with the ChoreBot.

let botDoorPath="https://s3.amazonaws.com/codecademy-content/projects/chore-door/images/robot.svg";

doorImage1.onclick = () => {

if(currentlyPlaying && !isClicked(doorImage1)){//PART OF GAME SECURITY

//A variable's src value can be assigned a new value held by another variable:

//doorImage1.src=botDoorPath; (changes)

doorImage1.src=openDoor1;

playDoor(doorImage1); // check if door close return win and loose condition by further checking that inside that door there isBot or not ...

//PART OF GAME SECURITY

}}

//DOOR 2 CREATED

let doorImage2 = document.getElementById('door2');

let beachDoorPath = "https://s3.amazonaws.com/codecademy-content/projects/chore-door/images/beach.svg";

doorImage2.onclick = () => {

if(currentlyPlaying && !isClicked(doorImage2)){//PART OF GAME SECURITY

//doorImage2.src=beachDoorPath; (changes)

doorImage2.src=openDoor2;

playDoor(doorImage2); // check if door close return win and loose condition by further checking that inside that door there isBot or not

//PART OF GAME SECURITY

}}

//DOOR 3 CREATED

let doorImage3 = document.getElementById('door3');

let spaceDoorPath ="https://s3.amazonaws.com/codecademy-content/projects/chore-door/images/space.svg";

doorImage3.onclick = () => {

if(currentlyPlaying && !isClicked(doorImage3)){ //PART OF GAME SECURITY

//doorImage3.src=spaceDoorPath; (changes)

doorImage3.src=openDoor3;

playDoor(doorImage3); // check if door close return win and loose condition by further checking that inside that door there isBot or not

}

}

/\* \*\*\*\*\*Let's Make This ACT Like a Game! - Part III (Building a Loser)\*\*\*\*\* \*/

//You need to check if a door has the game-ending ChoreBot.

const isBot = (door) => {

if (door.src.trim() === botDoorPath.trim()) {

return true;

} else {

return false;

}

}

// Let's Make This ACT Like a Game! - Part II (Building a Winner)

// \*\*\*\*\*\*\*\*\*\*GAME SECURITY .... \*\*\*\*\*\*\*\*\*\*\*\*

let startButton= document.getElementById('start');

const gameOver = (status) => {

if(status==='win')

{

startButton.innerHTML="You win! Play again?";

}

else

{

startButton.innerHTML="Game over! Play again?";

}

currentlyPlaying =false;

}

// checks each door clickable only once. so the player not to cheat.

const isClicked = (door) => {

if (door.src == closedDoorPath) {

return false;

} else {

return true;

}

}

//checks door open or closed

const playDoor = (door) => {

numClosedDoors--;

if (numClosedDoors === 0) {

gameOver('win');

} else if (isBot(door)) {

gameOver('lose');

}

}

// RANDOM CHARBOT GENERATOR

let openDoor1;

let openDoor2;

let openDoor3;

let numClosedDoors=3;

const randomChoreDoorGenerator = () => {

const choreDoor = Math.floor(Math.random() \* numClosedDoors);

if(choreDoor === 0) {

openDoor1= botDoorPath;

openDoor2= beachDoorPath;

openDoor3= spaceDoorPath;

}

else if (choreDoor === 1) {

openDoor1= beachDoorPath;

openDoor2= botDoorPath;

openDoor3= spaceDoorPath

}

else {(choreDoor === 2)

openDoor1= spaceDoorPath;

openDoor2= beachDoorPath;

openDoor3= botDoorPath;

}

}

// RESET VALUES

startButton.onclick = () => {

if (!currentlyPlaying){

startRound();

}

}

const startRound = () => {

// Reset all the doors to be closed

door1.src = closedDoorPath;

door2.src = closedDoorPath;

door3.src = closedDoorPath;

numClosedDoors = 3;

currentlyPlaying = true;

startButton.innerHTML = 'Good luck!';

randomChoreDoorGenerator();

}

randomChoreDoorGenerator();

startRound();

**What is an Event?**

When you refresh your email inbox, double tap on a post, or scroll through a newsfeed — something cool happens in your browser. These actions are known as events!

*Events* on the web **are user interactions and browser manipulations** that you can program to trigger functionality. Some examples of events are:

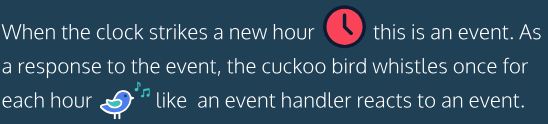
* A mouse clicking on a button
* Webpage files loading in the browser
* A user swiping right on an image

You can create interactivity on a website by assigning a function to respond to a specific **event *firing*, or triggering**.

# "Firing" Events

*After a specific event fires on a specific element* in the  DOM, an **event handler function** can be created to run as a response or create interactivity.

Let’s understand about **event handler functions** that modify and update DOM elements after an event fires.





# Event Handler Registration

JavaScript interprets **registered** events **as event objects** with properties and methods. When you create an **event handler function** on a specific event, **you create a property of the event object.**

An event handler function is **registered as a property** attached to the **DOM element** being interacted with, or the event target. syntax:

let eventTarget = document.getElementById('targetElement');

eventTarget.onclick = function() {

// this block of code will run

}

1. First, we accessed the DOM element that serves as the event target by its ID using document.getElementById('targetElement').
2. **Then we created the event handler property** which consists of the event target followed by the event name (the prefix on- and the event type.) In this example, we're using the **click event which fires** when the user presses and releases a mouse button on a DOM element.
3. Lastly, we assigned **an event handler function** to the property.

Note : Event handlers can also be registered as an HTML element attribute.

let readMore = document.getElementById('read-more')

let moreInfo = document.getElementById('more-info')

// Write your code here:

readMore.onclick = function() {

moreInfo.style.display = 'block';

}

**Adding Event Handlers**

**Way -1**

readMore.onclick = function() { // this is method

moreInfo.style.display = 'block'; // these are properties

}

// event is object 🡪 functionality is method and properties

It's best practice to create named event handler functions, instead of anonymous functions. Check out the syntax:

let eventHandlerFunction = function() {

// this block of code will run

}

Way-2

eventTarget.onclick = eventHandlerFunction;

The .addEventListener() method is **another common syntax for registering event handlers.** An *event listener* waits for a specific event to occur and calls a named event handler function to respond to it. This method requires two arguments:

1. The event type as a string
2. The event handler function

Check out the syntax of an .addEventListener() method with a click event:

Way-3

eventTarget.addEventListener('click', eventHandlerFunction);

You'll want to use the .addEventListener() method to **allow multiple event handlers to be registered to a single event** without changing its other event handlers.

Way-4

//Way - 4

view.addEventListener('click',() => {

open();

textChange();});

**Example:**

// Add the code you want to test below:

//Fetch ID's

let view = document.getElementById('view-button');

let close = document.getElementById('close-button');

let margo = document.getElementById('margo');

// Function Created

let open = function() {

margo.style.display = 'block';

close.style.display = 'block';

};

let hide = function() {

margo.style.display = 'none';

close.style.display = 'none';

};

let textChange = function() {

view.innerHTML = 'Hello, World!';

}

let textReturn = function() {

view.innerHTML = 'View';

}

// Adding events in DOM elements

//Way - 1

view.addEventListener('click',() => {

open();

textChange();});

//Way-2

close.onclick = hide;

close.addEventListener('click', textReturn);

**Removing Event Handlers**

The .removeEventListener() method is used reverse the .addEventListener() method. This method stops the code from "listening" for an event to fire when it no longer needs to. .removeEventListener also passes two arguments:

1. The event type as a string
2. The event handler function

Check out the syntax of a .removeEventListener()method with a click event:

eventTarget.removeEventListener('click', eventHandlerFunction);

Because this method **unregisters event handlers**, it needs to identify which function to remove from the event. The event handler function passed to the .removeEventListener() method **must** be the same function of the corresponding .addEventListener().

**Event Object Properties**

JavaScript stores events as *event objects* with their related data and functionality as properties and methods. There are pre-determined properties associated with event objects. You can call these properties to see information about the event, for example:

* the .target property to access the element that triggered the event.
* the .type property to access the name of the event.
* the .timeStamp property to access the number of milliseconds that passed since the document loaded and the event was triggered.

**Example:**

let social = document.getElementById('social-media');

let share = document.getElementById('share-button');

let text = document.getElementById('text');

// Write your code below

let sharePhoto = function(event) {

event.target.style.display = 'none';

text.innerHTML = 'You share the puppy in '

+ event.timeStamp + ' ms.';

}

share.onclick = sharePhoto;

# Event Types

Beyond the click event, there are all types of DOM events that can fire in a browser! It's important to know most events in the DOM take place without being noticed because there are no event handlers connected to them.

It's also **important to know some registered events don't depend on user interactions to fire.** For instance, the load event fires after website files completely load in the browser.

Browsers can fire many other events without a user — you can check out a list of events on the MDN Events . <https://developer.mozilla.org/en-US/docs/Web/Events>

Many events need user interaction with the DOM to fire. One user interaction event you've become familiar with is the click event. A click event fires when the user presses and releases a mouse button on an element in the DOM.

In the rest of this lesson, you'll explore more user interaction event types like the mouse and keyboard events. To explore more event types .Check out in the above link.

**Example:**

**Main.js**

// This variable stores the "Pick a Color" button

let button = document.getElementById('color-button');

// This variable stores the "Mystery Color" button

let mysteryButton = document.getElementById('next-button');

// This random number function that will creates color codes for the randomColor variable

function rgb(num) {

return Math.floor(Math.random() \* num);

}

// Write your code below

let colorChange = function() {

let randomColor = 'rgb(' + rgb(255) + ',' + rgb(255) + ',' + rgb(255) + ')';

// Not use id variable directly assign property using .event.target

event.target.style.backgroundColor = randomColor;

}

button.onclick = colorChange;

mysteryButton.onwheel = colorChange;

index.html

<body>

<section id='container'>

<img src='http://pngimg.com/uploads/rainbow/rainbow\_PNG5580.png'/>

<h1>Random Color Generator</h1>

<p>Find your new favorite color!</p>

<button id='color-button'>Pick a Color</button>

<button id='next-button'>Mystery Color</button>

</section>

<script src="main.js"></script>

</body>

# Mouse Events

This is different from a clickevent because mousedown doesn't need the mouse button to be released to fire.

The mouseup event is fired when the **user releases the mouse button.**

The mouseover event is fired when the **mouse enters the content of an element.**

The mouseout event is fired when the **mouse leaves an element.**

**Example:**

**.html**

<body>

<section id='container'>

<ul>

<li id='list-item-one'>The mouse must pass over to increase the box width</li>

<li id='list-item-two'>Release the mouse button to change the color</li>

<li id='list-item-three'>The mouse must leave the box to change the text</li>

<li id='list-item-four'>Click the mouse to make the fifth box appear</li>

<li id='list-item-five'>You found me!</li>

<button id='reset-button'>Reset</button>

</ul>

</section>

<script src="main.js"></script>

</body>

**.js file**

// These variables store the boxes on the side

let itemOne = document.getElementById('list-item-one');

let itemTwo = document.getElementById('list-item-two');

let itemThree = document.getElementById('list-item-three');

let itemFour = document.getElementById('list-item-four');

let itemFive = document.getElementById('list-item-five');

let resetButton = document.getElementById('reset-button');

// This function programs the "Reset" button to return the boxes to their default styles

let reset = function() {

itemOne.style.width = ''

itemTwo .style.backgroundColor = ''

itemThree.innerHTML = 'The mouse must leave the box to change the text'

itemFive.hidden = true;

};

resetButton.onclick = reset;

// Write code for the first list item

itemOne.onmouseover = function(){

itemOne.style.width = '430px';

};

// Write code for the second list item

itemTwo.onmouseup = function(){

itemTwo.style.backgroundColor = 'green'

};

// Write code for the third list item

itemThree.onmouseout = function(){

itemThree.innerHTML = 'The mouse has left the element.'

};

// Write code for the fourth list item

itemFour.onmousedown = function(){

itemFive.style.display = 'block';

};

# Keyboard Events

Another popular type of event is the keyboard event! keyboard events are triggered by user interaction with keyboard keys in the browser.

The keydown event is fired while a user presses a key down.

The keyup event is fired while a user releases a key.

The keypress event is fired when a user presses a key down and releases it. This is different from using keydown and keyup events together, because those are two complete events and keypress is one complete event.

Keyboard events have unique properties assigned to their event objects like the .key property that stores the values of the key pressed by the user. You can program the event handler function to react to a specific key, or react to any interaction with the keyboard.

**Example: 1**

Index.html

<body>

<div id='container'>

<h1>Ball Bounce</h1>

<p>Let's dribble the ball on the platform using any key on your </p>

<!--<button id='button'>Reset</button>-->

<div id='float-circle'></div>

<div id='platform'></div>

<div id='floor'></div>

</div>

<script src="main.js"></script>

</body>

**Main.js**

let ball = document.getElementById('float-circle');

// Write your code below

let up = function() {

ball.style.bottom = '250px';

};

let down = function() {

ball.style.bottom = '50px';

};

document.onkeydown = up;

document.onkeyup = down;

// it is not work by using target elemnt reason is

//event not able to focus on a paricular div . target elemnt work on any input elemnts .

**Example:2**

<body>

<p>A function is triggered when the user releases a key in the input field. The function transforms the character to upper case.</p>

Enter your name: <input type="text" id="fname" onkeyup="myFunction()">

<script>

function myFunction() {

var x = document.getElementById("fname");

x.value = x.value.toUpperCase();

}

</script>

</body>