前端網絡開發人員課程
（二）進階網絡程式設計

# 5. JS DOM V: Events I

Presented by Krystal Institute

## Learning Objective

- Understand what events are and how they work
- Know how to use events with Javascript

## Content

## 5.1 Revise on the previous lesson

## Attributes

- Standard attributes are converted into properties when a Dom object is created
- Using element.attributes returns a live collection of attributes in the specified element
- Attributes are always a string, and it will be converted to different data types when it converts into a DOM property

## Attributes

- data-* attribute are reserved for developer use, used in data collection and showing statistics
- There are 4 functions to manipulating attributes:
- setAttribute allows for overwriting and/or adding attributes

```
element.setAttribute(name, value);
```

- getAttribute returns the value of an attribute in the specified element

```
let value = element.getAttribute(name);
```

- removeAttribute removes an attribute from a specified element

```
element.removeAttribute(name);
```

## Attributes

- hasAAttribute checks if the specified element has the target attribute or not

- It returns a Boolean value: true if the attribute exist, and false otherwise

```
let result = element.hasAttribute(name);
```

## Styling

- Using element.style, CSS styles can be changed

```
element.style
```

- Multiple styles can be changed using either setAttribute or cssText method

```
div1.style.cssText = "color:black;display:block";
div2.setAttribute("style", "color:white;display:block ")
```

## Computed Style

- The computed style is the actual style property after every CSS modifiers has been applied

- Useful to see changes made with embedded or external styles

```
let style = window.getComputedStyle(element, "pseudoElement");
```

## Class Name & List

- className is a property that returns a list of all css classes of the element
- They are separated by a space as a long string
- Using operators like = and += can change or add classes

```
    <img class="image main" src="#">
    <div class="main panel"></div>
    <div class="panel"></div>
<script>
    let mainpanel = document.querySelector("div");
    mainpanel.className = "main";
    let img = document.querySelector("img");
    img.className += “ panel";
</script>
```

## Class Name & List

- ClassList is a read-only property of an element, returning a live collection of CSS classes
- add() adds one or more classes
- remove() removes one class
- replace() replaces an existing class with a new one
- contains() checks if the element contains a specific class
- toggle() adds the class if it is not inside an element, and removes it if it is

## 5.2 Events pt.1

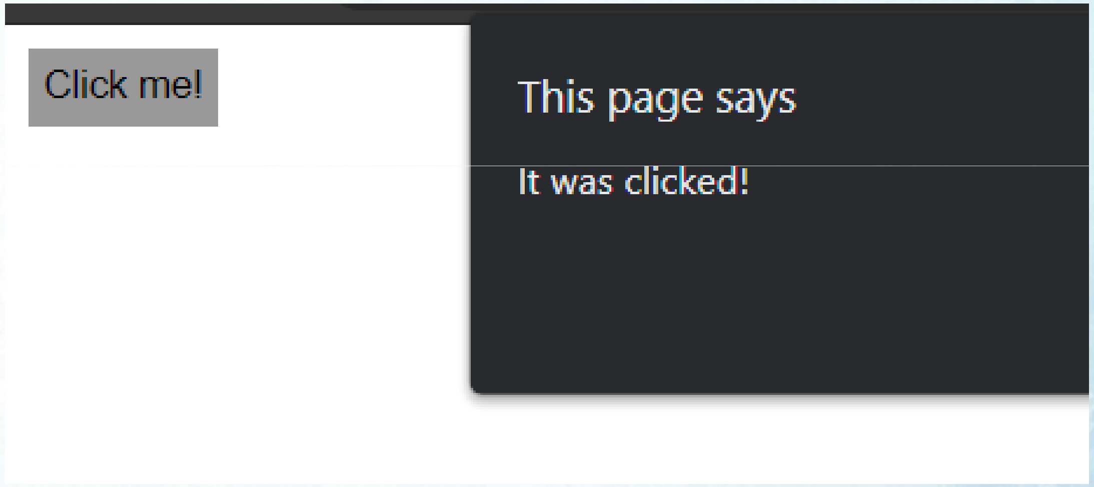
## JavaScript Events

- Events is an action that happens in the web browser
- Everything the user does in the website is an event
- That includes clicking a button, using your mouse and keyboard, or even moving your mouse!

## Event Listener

- A Event listener "listens" to any events that might happen, and can be programmed to perform some tasks when that event was triggered

- The example on the right displays a text when the button was clicked



It was clicked!

## Event Listener

- addEventListener(Event, function) is used to add a event listener to an element, and depending on the event added, it will run the function when it is triggered

```
element.addEventListener("Event", function);
```

## Event Flow

- When you click on a button, you're not just clicking the button, but everything that contains it, that means the <div>, <body>, and the whole HTML document

```
<html>
<head>
    <title>JSTutorial</title>
</head>
<body>
    <div id="container">
        <button id='btn'>Click Me!</button>
    </div>
</body>
</html>
```

## Event Flow

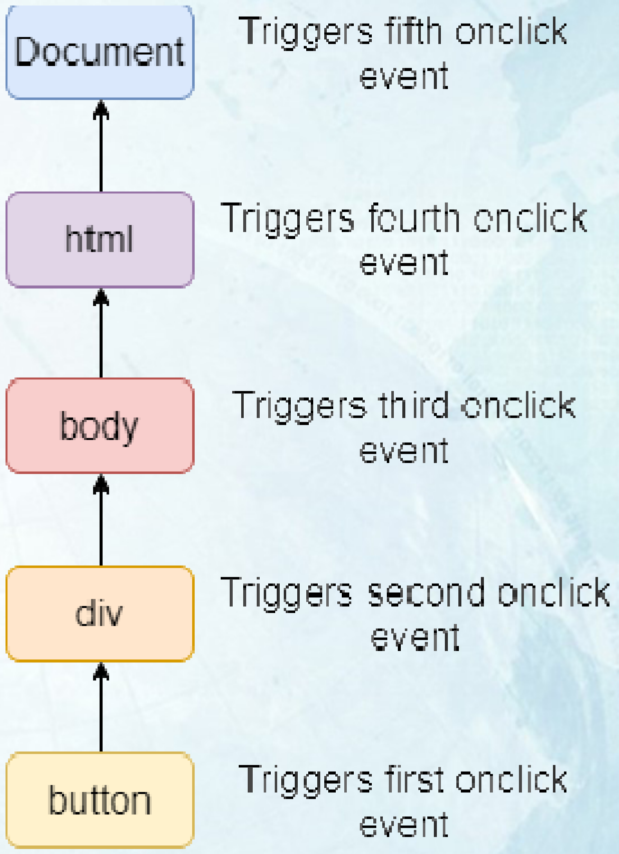
- The event flow shows the order in which events are received and triggered through the whole DOM Tree

- There are 2 main types of event models:
- Event bubbling and event capturing

## Event Bubbling

- A event bubbling model starts an event at the most specific element, and works its way up the Dom Tree

- Using the example from before, the event starts at button, then <div>, <body>, <html>, and lastly,  the document


Document
Triggers fifth onclick event
html
Triggers fourth onclick event
body
Triggers third onclick event
div
Triggers second onclick event
button
Triggers first onclick event

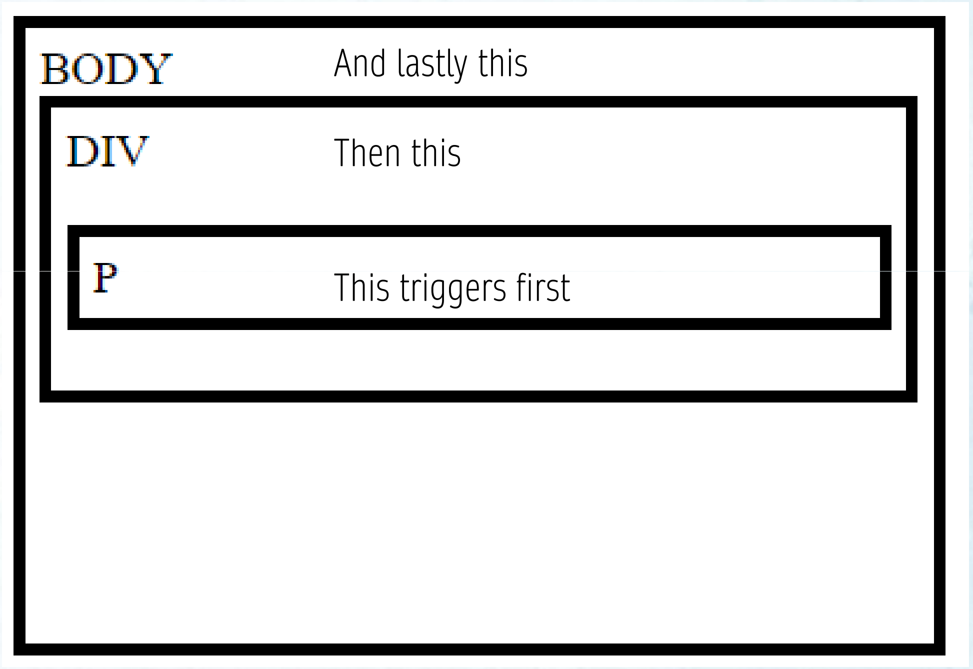
# Event Bubbling Activity

- Activity: try using the onclick event on every element container
- Create a <div> with <p> inside
- Set an onclick event on <body>, <div> and <p>
- Add outlines so its easier to visualize

```html
<html>
<head>
    <title>JSTutorial</title>
</head>
<body onclick="alert('body was clicked')"
      style="border: solid 4px black;padding: 5px;
      height: 200px;width: 300px">
    BODY
    <div onclick="alert('div was clicked')"
         style="border: solid 4px black;padding: 5px">
        DIV
        <p onclick="alert('p was clicked')"
           style="border: solid 4px black;padding: 5px">P</p>
    </div>
</body>
</html>
```

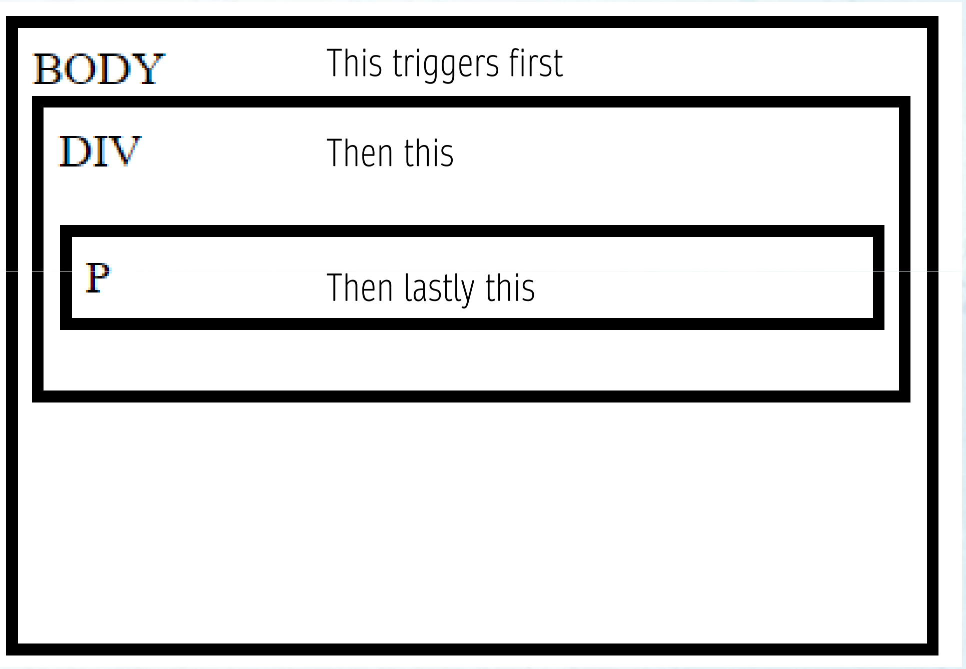
## Event Bubbling Activity

- When an element was clicked, the alert() function will trigger from the element you clicked on, then the one containing it, and so on



## Event Capturing

- Event capturing model is the opposite of the event bubbling model
- It starts from the least specific element, and work its way down to the most specific



## Event Capturing

- Event capturing is rarely used in browsers
- Event handlers added using addEventListener will not include event capturing, often only using the event bubbling model

## Event Handling

- There are 3 types of event handling

- The first type is HTML event attribute
- The onclick attribute has been used a lot in this course, it represents one of the event handlers

```
<p onclick="alert('p was clicked')">P</p>
```

## Event Handling

- HTML event attribute could have some issues
- If the webpage has been loaded and the event handler has not
- The showAlert will return undefined on click as no alert has been loaded yet
- This is rarely an issue if the page isn't too complex

```
<p onclick="showAlert()">P</p>
```

# Event Handling

- The next type is DOM Level 0 Event Handlers
- A better way to add event handlers is to use the onclick/on* property
- Assign it to a function to be performed when the onclick event is triggered

```
<body>
    <p>P</p>
<script>
    let p = document.querySelector("p");
    p.onclick = function() {
        alert("p was clicked")
    };
</script>
</body>
```

## Event Handling

- Using this, you can access the element's methods and properties

- To remove the event handler on a element, set their respective event to null

```
<body>
    <p id="main">P</p>
<script>
    let p = document.querySelector("p");
    p.onclick = function() {
        alert(this.id);
    };

    p.onclick = null;
</script>
</body>
```

## Event Handling

- The last type is DOM Level 2 Event Handlers

- 2 methods can be used to modify event listeners:
- addEventListener("Event", function)
- removeEventListener("Event", function)

```
<body>
    <p id="main">P</p>
<script>
    let p = document.querySelector("p");
    p.addEventListener("click", function() {
        alert("It was clicked.")
    });
</script>
</body>
```

## Event Handling

- addEventListener works similar as level 0 Event Handlers
- It accepts 3 arguments: the event name, the event handler function, and a boolean value
- The Boolean value depicts whether the event handler should be called, during event capturing (true) or event bubbling (false)

```
window.addEventListener("click", function(event) {})
```

- The function inside the listener can use events as a parameter, which represent the event itself

## Event Handling

- addEventListener also allows the element to have 2 of the same event types, with 2 different event listeners

```
<body>
    <p id="main">P</p>
<script>
    let p = document.querySelector("p");
    p.addEventListener("click", function() {
        alert("It was clicked.")
    });

    p.addEventListener("mousemove", function() {
        console.log("Your mouse moved")
    });
</script>
</body>
```

## Event Handling

- removeEventListener removes an event listener that was added via addEventListener

- The exact same arguments needs to be passed in case there are multiple event listeners

```
<body>
    <p id="main">P</p>
<script>
    let p = document.querySelector("p");
    function showalert() {
        alert("It was clicked")
    };

    p.addEventListener("click", showalert());
    p.removeEventListener("click", showalert());
</script>
</body>
```

## Event.target

- In the function following the addEventListener, event can be used as an parameter depicting the event that was triggered
- Using event.target returns the element that triggered the event listener, it could be any child nodes of the element

```
<body>
    <div>This is a wrapper div
        <p>Clicking on this will return p instead
            of div if using event.target</p>
    </div>
<script>
    let div = document.querySelector("div");
    div.addEventListener("click", function(event) {
        console.log(event.target.textContent)
    });
</script>
</body>
```

## Page Load Events

- There are 4 events that triggers when you load a webpage or leave a webpage
- They can be used as Event Handlers from the window or document object
- It is used for executing scripts as soon as possible

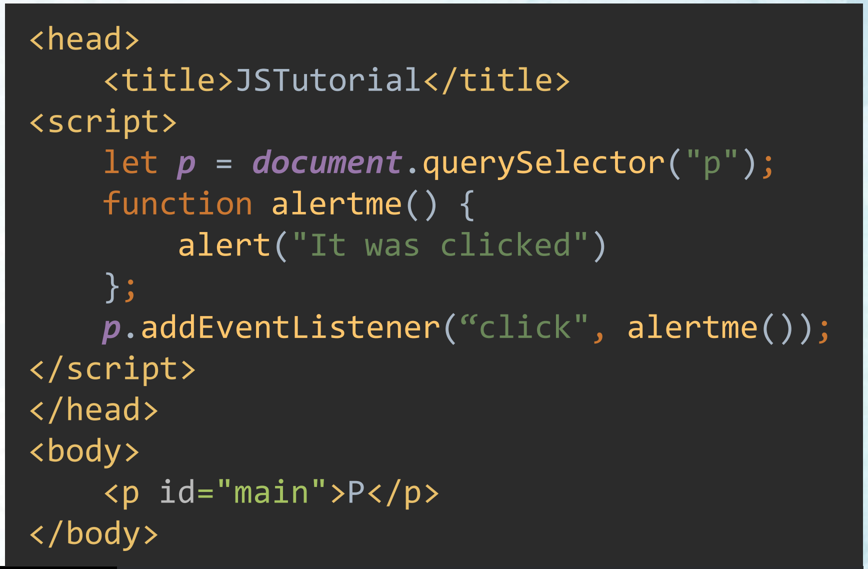
## Page Load Events

- DOMContentLoaded – this is triggered after the browser has loaded the HTML and completed building the DOM Tree, but has not loading external resources like images or stylesheets

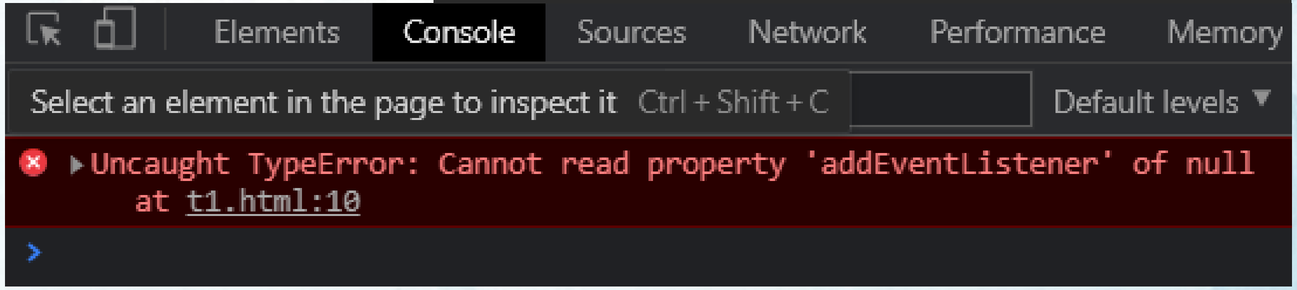
- load – this is triggered after the browser and all the external resources are loaded

## Page Load Events

- One example for using these is when you have a script in the <head> that references <body>
- Normally, this is return an error since <p> was assigned before it was even created in the DOM Tree (The script tag was created and ran before <p>)



```
<head>
    <title>JSTutorial</title>
<script>
    let p = document.querySelector("p");
    function alertme() {
        alert("It was clicked")
    };
    p.addEventListener(“click", alertme());
</script>
</head>
<body>
    <p id="main">P</p>
</body>
```


Elements
Console
Sources
Network
Performance
Memory
Select an element in the page to inspect it Ctrl + Shift + C
Default levels

```
Uncaught TypeError: Cannot read property 'addEventListener' of null
    at t1.html:10
```

## Page Load Events

- With load events, it is possible to reference elements that are created later than the script as the script is only run when the whole DOM Tree is created

```
<head>
    <title>JSTutorial</title>
<script>
    document.addEventListener("DOMContentLoaded", function() {
        let p = document.querySelector("p");
        p.addEventListener("click", function() {
            alert("It was clicked");
        });
    })
</script>
</head>
<body>
    <p id="main">P</p>
</body>
```

## Page Load Events

- Likewise, the other 2 load events triggers before the user closes the browser
- beforeunload triggers before the page and resources are unloaded
- unload triggers after everything is unloaded

```
window.addEventListener("beforeunload", function(event) {})
```

## Page Load Events Exercise

- Create a website that…
- Contains a button put in <body> and a empty <div>
- The <div> should display a text on click of the button
- The button and the <div> should be put inside <body> and the script should be put inside <head>

# Page Load Events Solution

- Set up a simple button and <civ> inside the <body>
- In the script, add a DOMContentLoaded event listener
- Locate the button and div, add another event listener that displays text on button click

```
<body>
    <button type="button">Click me!</button>
    <div id="display"></div>
</body>
```

```
<head>
    <title>JSTutorial</title>
<script>
    document.addEventListener("DOMContentLoaded", function() {
        let btn = document.querySelector("button");
        btn.addEventListener("click", function() {
            let div = document.querySelector("#display");
            div.textContent = "I'm clicked!"
        });
    })
</script>
</head>
```

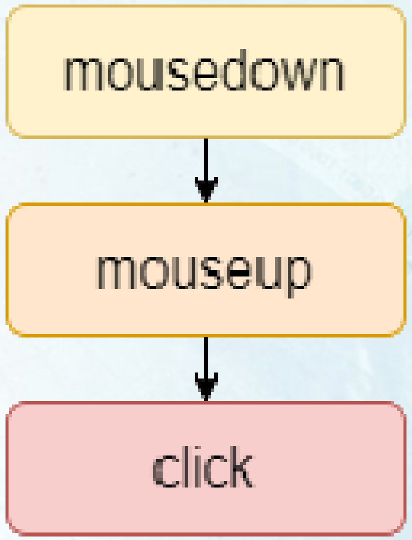
## Mouse Events

- Mouse events triggers when you use your mouse to interact with elements in the page

- In DOM Level 3, there are 9 mouse events.

## Mouse Events

- When you press a button, 3 events triggers in order:
- mousedown triggers when you press on the mouse button
- mouseup triggers when you release the mouse button
- click triggers after one mousedown and one mouseup



## Mouse Events

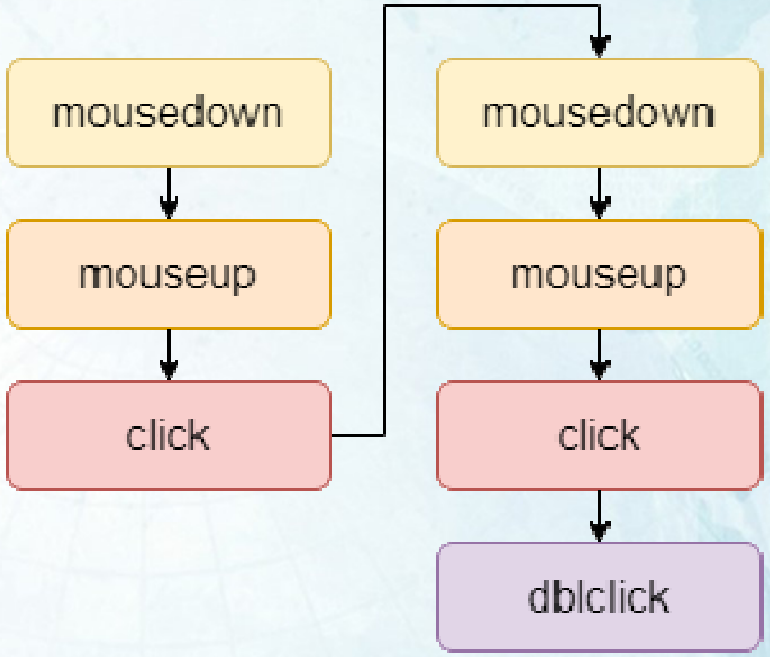
- mousedown can be individually triggered when you press the mouse button inside the element, and release the button outside of the element

- Likewise, mouseup can be triggered individually by pressing the mouse button outside the element, and releasing the button while inside the element

- In both cases, the click event never triggers

## Mouse Events

- dblclick triggers when the user double clicks on an element
- This will trigger the mouseup – mousedown – click loop twice before triggering the dblclick event

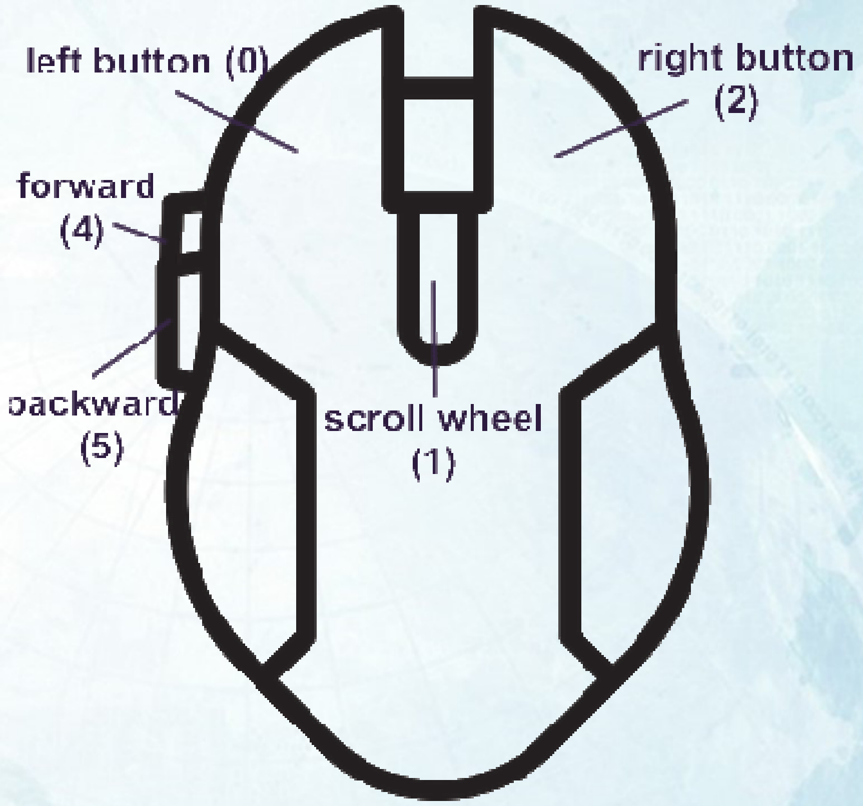


## Mouse Events

- mousemove triggers constantly as long as you're moving the mouse inside the element, no matter how small or big the movement are
- mouseover / mouseout triggers when the mouse enters / leaves the element, it also triggers every time it enters / leaves the element's children elements
- mouseenter / mouseleave triggers when the mouse enters / leaves the element, it doesn't trigger when it enters / leaves its child elements

## Mouse Event Handling

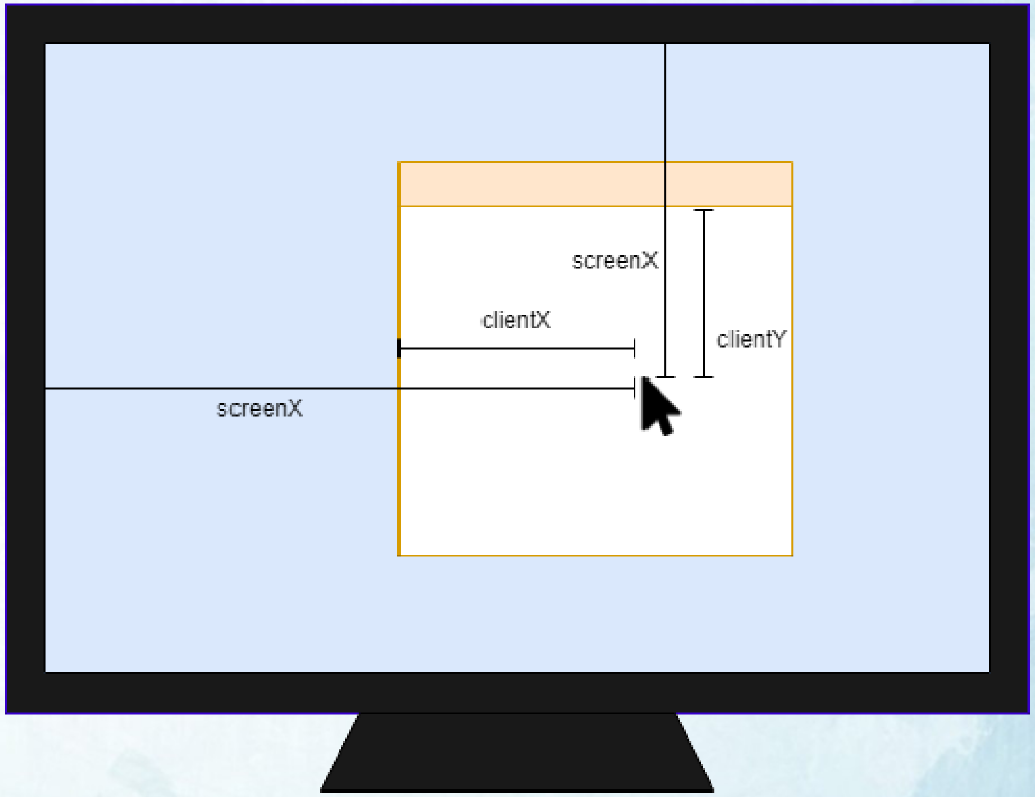
- When adding a mouse event listener, the mouse button event can be used to detect which mouse button they are using
- event.button is used for that matter
- event.button has 5 values 0 to 4, and they represent each button on a modern mouse



## Mouse Coordinates

- event.screenX / event.screenY returns the coordinate of the mouse cursor in relation to the monitor screen

- event.clientX / event.clientY returns the coordinate of the mouse cursor in relation to the web broswer



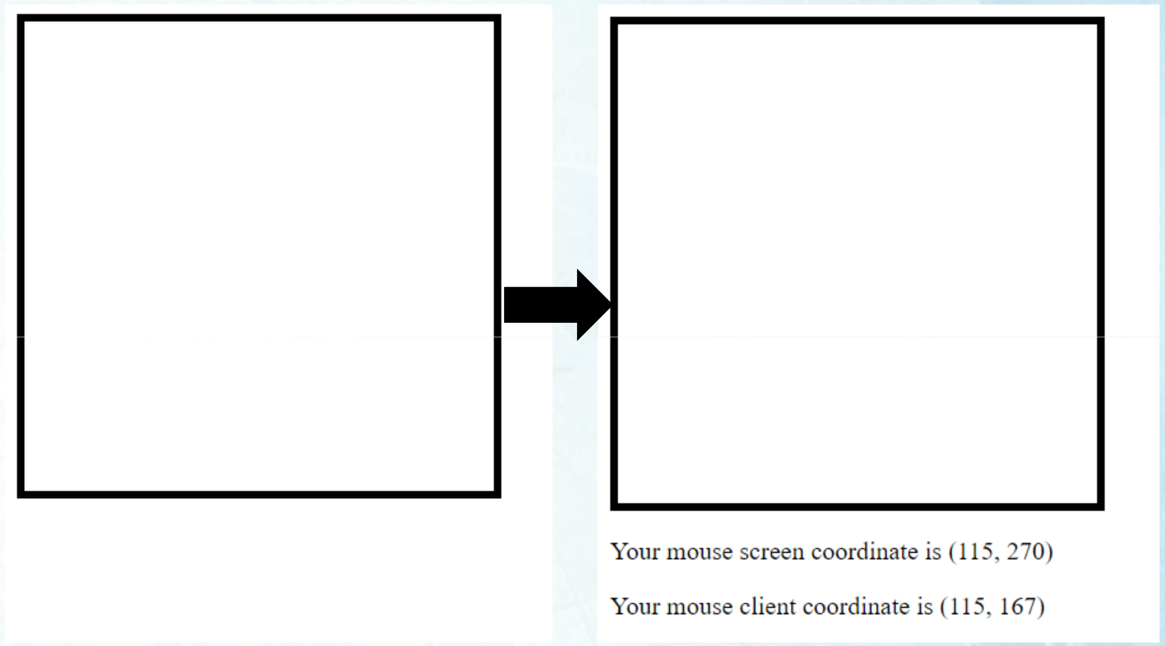
## Exercise

- Create a website that…

- Has a <div> representing a canvas, with black border around it
- When user’s mouse hover over it, it will show its client and screen coordinate in another div text
- Displaying coordinates should be constant as long as user’s mouse is inside the canvas
- Finish this exercise by the end of the lesson

## Exercise Example

- Upon moving the mouse on the div
- 2 texts show up, showing the client and the screen coordinates
- It should update on mouse move



Your mouse screen coordinate is (115, 270)

Your mouse client coordinate is (115, 167)

## References

- Use these if you need more explanations!
- https://www.javascripttutorial.net/es6/
- https://javascript.info/

- Use this if you need more specific answers!
- https://stackoverflow.com/