## Higher Order Functions Examples

- **findIndex():** This method returns the index of the first element in the array that satisfies the provided testing function. If no element matches, it returns -1. Like other array methods, findIndex() takes a callback function as an argument, making it a higher-order function.

  Example

```
const numbers = [10, 20, 30, 40];
const index = numbers.findIndex(num => num > 25); // 2
```

- **findLastIndex():** Similar to findIndex(), but searches from right to left and returns the index of the first element that satisfies the testing function. It Accepts a predicate function as an argument.

  Example

```
const numbers = [1, 2, 3, 4, 5];
const lastEvenIndex = numbers.findLastIndex(num => num % 2 === 0);
console.log(lastEvenIndex); // Output: 3
```

- **findLast():** Similar to find(), but searches the array from right to left and returns the value of the first element that satisfies the provided testing function. It Accepts a predicate function as an argument.

  Example

```
const numbers = [1, 2, 3, 4, 5];
const lastEven = numbers.findLast(num => num % 2 === 0);
console.log(lastEven); // Output: 4
```

- **from():** The Array.from() method creates a new, shallow-copied Array instance from an array-like or iterable object. It also accepts a map function, making it a higher-order function.

  Example

```
const str = "hello";
const arr = Array.from(str, char => char.toUpperCase());
console.log(arr); // ['H', 'E', 'L', 'L', 'O']
```

- **reduceRight():** Similar to reduce(), but processes the array elements from right to left (from the last element to the first). It Accepts a reducer function as an argument.

  Example

```
const numbers = [1, 2, 3, 4];
const result = numbers.reduceRight((accumulator, currentValue) =>
accumulator - currentValue);
console.log(result); // Output: -2
```

- **toSorted():** Returns a new array with the elements sorted according to the provided compare function, without modifying the original array. It accepts a compare function as an argument.

  Example

```
const numbers = [3, 1, 4, 1, 5];
const sortedNumbers = numbers.toSorted((a, b) => a - b);
console.log(sortedNumbers); // [1, 1, 3, 4, 5]
console.log(numbers);       // [3, 1, 4, 1, 5] (original array is unchanged)
```

- **Custom Higher-Order Functions:** You can create your own higher-order functions in JavaScript to abstract behaviour.

  Example: A higher-order function that takes a function as an argument.

```
function repeat(n, action) {
  for (let i = 0; i < n; i++) {
    action(i);
  }
}
repeat(5, console.log); // Logs: 0, 1, 2, 3, 4
```

  Example: A higher-order function that returns a new function.

```
function multiplyBy(factor) {
  return function (number) {
    return number * factor;
  };
}
const double = multiplyBy(2);
console.log(double(5)); // 10
```

- **Other Higher-Order Function:** Some built-in functions and methods that allow functions as parameters.

  **addEventListener():** Registers an event handler for an event.

  Example

  ```
  document.addEventListener('click', () => console.log('Clicked!'));
  ```

## Event Handling

- Event handling is fundamental to creating interactive, responsive applications. It allows the program to wait for and react to user actions or browser events.
- Event handling in JavaScript refers to the process of capturing and responding to events triggered by user actions or the browser.
- These events can be things like mouse clicks, key presses, or changes in form input.
- JavaScript provides mechanisms to "listen" for these events and respond with specific actions, enabling interactive and dynamic web applications.

### What is an Event?

An event in JavaScript is a signal that something has occurred. It can be triggered by:

- User actions (e.g., clicking a button, hovering over an element, pressing a key).
- Browser actions (e.g., page load, resizing the window).
- Programmatic actions (e.g., triggering events using JavaScript code).

Each event comes with an event object, containing useful information like:

- The type of event (e.g., click, keydown).
- The target element where the event occurred.
- Details about the event (e.g., mouse position, key pressed).

### Examples of common events

- click – Triggered when a user clicks an element.
- keydown – Triggered when a key is pressed.
- submit – Triggered when a form is submitted.
- mouseover – Triggered when the mouse hovers over an element.

### Event-Driven Programming

- Event-Driven Programming is a programming paradigm where the flow of the program is determined by events (i.e., changes in state or user inputs).
- Instead of executing code in a sequential manner, event-driven programs wait for events and then execute corresponding event handlers.
- The code doesn't run in a linear sequence from start to finish.
- JavaScript waits for an event (like a click) and only then runs the code associated with that event.
- This model is fundamental for creating responsive and interactive user interfaces, as JavaScript can react to user inputs without blocking the main execution flow.

**Key Concepts of Event-Driven Programming in JavaScript**

- **Event Listener:** An event listener is a function that waits for a specific event to happen on a particular element. When the event occurs, the event listener calls an event handler function to respond to the event.
  Example of adding an event listener

```
const button = document.getElementById('myButton');
button.addEventListener('click', function() {
    alert('Button was clicked!');
});
```

  The above code listens for a click event on the button, and when the button is clicked, the event handler displays an alert.

- **Event Handler:** An event handler is a function that defines what should happen when an event is detected. It is usually passed as a callback function in the addEventListener() method.
  Inline event handler example

```
<button onclick="alert('Button clicked!')">Click Me</button>
```

- **Event Propagation:** When an event occurs, it doesn't just trigger the event handler attached to the specific element; it can also propagate to other elements (parent or child elements) in a process known as event propagation.

  There are two forms of propagation:
  - **Event Bubbling:** The event starts from the target element and bubbles up through its ancestors (parents).
  - **Event Capturing:** The event is first captured by the outermost ancestor and propagates down to the target.

  Example of event propagation

```
document.getElementById('outerDiv').addEventListener('click', function() {
    console.log('Outer div clicked');
});
document.getElementById('innerDiv').addEventListener('click',
function(event) {
    console.log('Inner div clicked');
    event.stopPropagation(); }); // Prevents event bubbling to outer div
```

```
<div id="outerDiv" style="width: 300px; height: 300px; background-color: lightblue;">
    Outer Div
    <div id="innerDiv" style="width: 150px; height: 150px; background-color: lightcoral; margin: auto; margin-top: 50px;">
        Inner Div    </div>
</div>
```

- **Event Object:** Every event has an associated event object that contains information about the event, like which element was clicked, where the click occurred, etc.
  Example of using the event object

```
document.addEventListener('click', function(event) {
    console.log('Clicked element:', event.target);
    console.log('Mouse coordinates:', event.clientX, event.clientY);
});
```

**Use Cases of Event-Driven Programming**

- **User Interaction:** Detecting clicks, key presses, and form submissions to dynamically respond to user inputs.
  Example: Opening a modal when a button is clicked.

- **Real-Time Applications:** Events are crucial in applications like chat rooms, where updates must be handled based on real-time inputs.
  Example: A new message arrives, and the UI updates without reloading the page.

- **Dynamic Form Validation:** Events can trigger live form validation as the user fills out a form.
  Example: Showing validation messages after the user stops typing in an input field.

- **Interactive UI Components:** Reacting to hover events, drag-and-drop actions, and touch gestures.
  Example: Creating a drag-and-drop feature in a file upload interface.

- **Custom Event Handling:** You can create and dispatch custom events to decouple functionality in complex applications.
  Example: Triggering a custom event after a successful API call.

**Event Examples**

1. **Mouse Events:** Mouse events are triggered by user actions involving the mouse. These events are useful for handling clicks, hovering, and interacting with various UI elements.

   a. **click (Mouse Click Event):** The click event occurs when a user clicks an element (usually a button or a link).

   Example

```
<button id="myButton">Click me!</button>
<script>
        const button = document.getElementById('myButton');
        button.addEventListener('click', function() {
          console.log('Button clicked!');
        });
</script>
```

   b. **dblclick (Double Click Event):** The dblclick event is fired when the user clicks an element twice in quick succession.

   Example

```
<button id="myButton">Double-click me!</button>
<script>
        const button = document.getElementById('myButton');
        button.addEventListener('dblclick', function() {
          console.log('double-clicked!');
        });
</script>
```

   c. **mouseover (Mouse Hover Event):** The mouseover event occurs when the mouse pointer hovers over an element.

   Example

```
<div id="myDiv" style="width: 200px; height: 100px; background-color: lightblue;">Hover over me! </div>
<script>
        const div = document.getElementById('myDiv');
        div.addEventListener('mouseover', function() {
          div.style.backgroundColor = 'red';    });
</script>
```

d. **mouseout (Mouse Leave Event):** The mouseout event occurs when the mouse pointer leaves an element.

Example

```
<div id="myDiv" style="width: 200px; height: 100px; background-color: lightblue;"> Hover over me! </div>
<script>
        const div = document.getElementById('myDiv');
        div.addEventListener('mouseout', function() {
          div.style.backgroundColor = 'lightblue';
        });
</script>
```

2. **Keyboard Events:** Keyboard events are triggered when the user interacts with the keyboard.

a. **keydown (Key Pressed Down Event):** The keydown event is fired when a key is pressed down.

Example

```
<input type="text" id="myInput">
<script>
        const input = document.getElementById('myInput');
        input.addEventListener('keydown', function(event) {
          console.log('Key pressed:', event.key);
    });
</script>
```

b. **keypress (Character Key Press Event):** The keypress event is fired when a key that produces a character is pressed down. It does not detect non-character keys (like Shift or Alt).

Example

```
<input type="text" id="myInput">
<script>
        const input = document.getElementById('myInput');
        input.addEventListener('keypress', function(event) {
          console.log('Character key pressed:', event.key);     });
</script>
```

c. **keyup (Key Released Event):** The keyup event is fired when a key is released.

Example

```
<input type="text" id="myInput">
<script>
        const input = document.getElementById('myInput');
        input.addEventListener('keyup', function(event) {
            console.log('Key released:', event.key);
        });
</script>
```

3. **Form Events:** Form events are used to handle user input and interaction with forms.

a. **submit (Form Submission Event):** The submit event is triggered when a form is submitted.

Example

```
<form id="myForm">
    <input type="text" placeholder="Enter your name" required>
    <button type="submit">Submit</button>
</form>
<script>
        const form = document.getElementById('myForm');
        form.addEventListener('submit', function(event) {
            event.preventDefault(); // Prevents the form from refreshing the
    page
            console.log('Form submitted successfully!');    });
</script>
```

b. **change (Input Value Change Event):** The change event is fired when the value of an input element changes and loses focus.

Example

```
<input type="text" id="myInput" placeholder="Type something">
<script>
        const input = document.getElementById('myInput');
        input.addEventListener('change', function() {
            console.log('Input value changed to: ' + input.value);    });
</script>
```

c. **focus (Element Focus Event):** The focus event is triggered when an element gains focus (e.g., when a user clicks into an input field).

Example

```
<input type="text" id="myInput" placeholder="Click to focus">
<script>
        const input = document.getElementById('myInput');
        input.addEventListener('focus', function() {
          input.style.backgroundColor = 'red';
            input.style.color = 'white';
        });
</script>
```

d. **blur (Element Loses Focus Event):** The blur event is triggered when an element loses focus.

Example

```
<input type="text" id="myInput" placeholder="Click away to blur">
<script>
        const input = document.getElementById('myInput');
        input.addEventListener('blur', function() {
          input.style.backgroundColor = 'white';
        });
</script>
```

4. **Window Events:** Window events handle interactions with the browser window.

a. **load (Page Load Event):** The load event is triggered when the whole page (including images and external resources) is fully loaded.

Example

```
<script>
        window.addEventListener('load', function() {
          alert('Page is fully loaded successfully!');
        });
</script>
```

b. **resize (Window Resize Event):** The resize event is fired when the window is resized.

Example

```
<script>
        window.addEventListener('resize', function() {
            console.log('Window resized to: ' + window.innerWidth + 'x' +
        window.innerHeight);
        });
</script>
```

c. **scroll (Page Scroll Event):** The scroll event is triggered when the user scrolls the page.

Example

```
<script>
            window.addEventListener('scroll', function() {
                console.log('Page is scrolling! Scroll position: ' +
            window.scrollY);
            });
</script>
```

5. **Touch Events (For Mobile Devices):** Touch events are specific to touch-enabled devices, like mobile phones or tablets.

a. **touchstart (Finger Touches the Screen):** The touchstart event is triggered when a finger touches the screen.

Example

```
<div id="touchArea" style="width: 200px; height: 200px; background-color:
lightgrey;">     Touch me! </div>
<script>
        const touchArea = document.getElementById('touchArea');
        touchArea.addEventListener('touchstart', function() {
            console.log('div touched!');
        });
</script>
```

b. **touchend (Finger Removed from the Screen):** The touchend event occurs when the finger is removed from the screen.

Example

```
<div id="touchArea" style="width: 200px; height: 200px; background-color: lightgrey;">    Touch me! </div>
<script>
        const touchArea = document.getElementById('touchArea');
        touchArea.addEventListener('touchend', function() {
          console.log('div is untouched!');
        });
</script>
```

c. **touchmove (Finger Moved on the Screen):** The touchmove event is triggered when a finger moves across the screen while touching it.

Example

```
<div id="touchArea" style="width: 200px; height: 200px; background-color: lightgrey;">    Touch and move! </div>
<script>
        const touchArea = document.getElementById('touchArea');
        touchArea.addEventListener('touchmove', function() {
          console.log('Moving fingures on component!');
        });
</script>
```

## Removing Event Listeners

- **Removing Event Listeners with removeEventListener()**
  The removeEventListener() method allows you to remove an event listener that was previously added using addEventListener().
  This method requires the same function reference (not a new one) that was used when adding the listener.
  This means if you use an anonymous function, you cannot remove the event listener because there's no way to reference that function later.
  Example

```
<button id="myButton">Click Me</button>
  <button id="removeListener">Remove Click Listener</button>

  <script>
        // Named function to handle click event
        function handleClick() {
          console.log('Button Clicked!');
        }

        // Adding the event listener using the named function
        const button = document.getElementById('myButton');
        button.addEventListener('click', handleClick);

        // Remove event listener when the second button is clicked

    document.getElementById('removeListener').addEventListener('click'
    , function() {
          button.removeEventListener('click', handleClick);
          console.log('Click listener removed!');
        });
</script>
```

Example: Using Anonymous Function (Not Removable)

```
<button id="myButton">Click Me</button>
  <button id="removeListener">Try to Remove Listener</button>
  <script>
        // Using anonymous function for event listener
        const button = document.getElementById('myButton');
        button.addEventListener('click', function() {
          console.log('Button Clicked!');       });
        // Trying to remove the event listener
      document.getElementById('removeListener').addEventListener('click'
      , function() {
          button.removeEventListener('click', function() {
            console.log('Button Clicked!');          });
          console.log('Listener cannot be removed!');       });
  </script>
```

In the above code, the event listener is added using an anonymous function, and attempting to remove the listener fails because the anonymous function has no reference. Hence, the listener remains active even after attempting removal.

**preventDefault():** preventDefault() prevents the default action that is normally taken when the event occurs.

For example

- Preventing a form from submitting when the submit button is clicked.
- Preventing a link from navigating to a URL when clicked.

Example: Using preventDefault() to Stop Form Submission

```
<form id="myForm">
  <label for="name">Name:</label>
  <input type="text" id="name" name="name">
  <button type="submit">Submit</button>    </form>
<script>
  document.getElementById('myForm').addEventListener('submit', function(event) {
    event.preventDefault();  // Prevents the form from submitting
    console.log('Form submission prevented!');       });
</script>
```