6-Interactivity

Chapter 6: Bringing Interactivity to Life

Interactivity is what makes web applications engaging and user-friendly. By harnessing JavaScript's capabilities, you can dynamically respond to user actions, create animations, and make your web page feel alive.

6.1 Adding Events to Elements

At the heart of interactivity is event handling, where JavaScript listens for user actions such as clicks, key presses, or mouse movements.

Example:

```
const button = document.querySelector("#myButton");

button.addEventListener("click", () => {
  alert("Button clicked!");
});
```

6.2 Using `setTimeout` and `setInterval`

JavaScript provides two essential functions to manage time-based actions:

1. `setTimeout`: Executes a function after a specified delay.
   Example:

   ```
   setTimeout(() => {
     console.log("Hello after 3 seconds!");
   }, 3000); // Delay of 3000ms (3 seconds)
   ```

2. `setInterval`: Executes a function repeatedly at a set interval.
   Example:

   ```
   setInterval(() => {
     console.log("This runs every 2 seconds!");
   ```

```
}, 2000);
```

6.3 Animating Elements

Animation brings your webpage to life. By manipulating CSS properties using JavaScript, you can create smooth, interactive animations.

6.3.1 Using `setInterval` for Animations

Example:

```
const box = document.querySelector("#box");
let position = 0;

function moveBox() {
  position += 5;
  box.style.left = position + "px";

  if (position > 300) {
    clearInterval(animation); // Stop animation
  }
}

const animation = setInterval(moveBox, 50); // Move every 50ms
```

6.3.2 Using `requestAnimationFrame`

For smoother animations, use `requestAnimationFrame`. This method adjusts to the browser's refresh rate, ensuring consistent performance.

Example:

```
const box = document.querySelector("#box");
let pos = 0;

function animate() {
  if (pos < 300) {
    pos += 5;
    box.style.left = pos + "px";
```

```
    requestAnimationFrame(animate); // Recursively call animate
  }
}

animate(); // Start animation
```

6.4 CSS Transitions and JavaScript

Instead of manually changing styles in tiny increments, CSS transitions allow you to animate style changes effortlessly. JavaScript enhances transitions by dynamically adding or modifying classes.

Example:

```
/* CSS */
.box {
  width: 100px;
  height: 100px;
  background-color: blue;
  transition: transform 0.5s, background-color 0.5s;
}

.box.active {
  transform: rotate(45deg);
  background-color: red;
}
```

```
// JavaScript
const box = document.querySelector(".box");

box.addEventListener("click", () => {
  box.classList.toggle("active"); // Toggles the "active" class
});
```

6.5 Real-Time Updates

Dynamic user interactions often require real-time updates to the DOM. Here are a few techniques:

1. **Dynamic Content Update**
   Example:

```
const text = document.querySelector("#text");

text.addEventListener("input", (event) => {
  const output = document.querySelector("#output");
  output.textContent = event.target.value; // Update in real-time
});
```

2. **Changing Styles Dynamically**
   Example:

```
const button = document.querySelector("#themeButton");

button.addEventListener("click", () => {
  document.body.style.backgroundColor =
    document.body.style.backgroundColor === "black" ? "white" : "black";
  document.body.style.color =
    document.body.style.color === "white" ? "black" : "white";
});
```

6.6 Practical Examples

1. **Interactive Counter**
   HTML:

```
<button id="decrement">-</button>
<span id="counter">0</span>
<button id="increment">+</button>
```

   JavaScript:

```
let count = 0;

const counter = document.querySelector("#counter");
const increment = document.querySelector("#increment");
const decrement = document.querySelector("#decrement");

increment.addEventListener("click", () => {
  count++;
  counter.textContent = count;
```

```
});

decrement.addEventListener("click", () => {
  count--;
  counter.textContent = count;
});
```

---

2. **Simple Drawing App**
   HTML:

```
<canvas id="drawingCanvas" width="500" height="500" style="border: 1px solid black;"></canvas>
```

JavaScript:

```
const canvas = document.querySelector("#drawingCanvas");
const ctx = canvas.getContext("2d");
let drawing = false;

canvas.addEventListener("mousedown", () => (drawing = true));
canvas.addEventListener("mouseup", () => (drawing = false));
canvas.addEventListener("mousemove", (event) => {
  if (!drawing) return;

  ctx.fillStyle = "black";
  ctx.beginPath();
  ctx.arc(event.offsetX, event.offsetY, 2, 0, Math.PI * 2);
  ctx.fill();
});
```

---

3. **Modal Pop-Up**
   HTML:

```
<button id="openModal">Open Modal</button>
<div id="modal" style="display: none; background: rgba(0, 0, 0, 0.8); position: fixed; inset: 0; justify-content: center; align-items: center;">
  <div style="background: white; padding: 20px; border-radius: 5px;">
    <p>This is a modal!</p>
```

```
    <button id="closeModal">Close</button>
  </div>
</div>
```

JavaScript:

```
const modal = document.querySelector("#modal");
const openModal = document.querySelector("#openModal");
const closeModal = document.querySelector("#closeModal");

openModal.addEventListener("click", () => {
  modal.style.display = "flex";
});

closeModal.addEventListener("click", () => {
  modal.style.display = "none";
});
```