Lesson 4: Timers

🎯 Goal

Students will learn how to use JavaScript's `setTimeout()` and `setInterval()` to create time-based behavior. By the end of the class, they will build an interactive reaction timer mini-game.

⏱️ Total Duration: 90–120 minutes

Level: Intermediate (students have prior experience with variables, functions, and basic DOM)

🧠 1. Warm-Up & Review (10 minutes)

🎯 Learning Goals

- Get students thinking about where time matters in real life and games
- Refresh key JavaScript concepts from previous lessons (`function`, `let`, and DOM events)

🗣️ Discussion Questions (3 minutes)

Begin with open-ended, relatable questions to engage students:

1. **"Have you ever played a game where speed or timing matters?"**
    - Examples: rhythm games, clicker games, FPS games, obstacle courses
2. **"Where have you seen countdowns or timers on websites or apps?"**
    - Examples: YouTube ads, online sales countdowns, ticket booking timers, exam platforms

Encourage students to share answers verbally or via chat if online.

You can note a few of their answers on the board or screen to make it feel collaborative.

🧠 Quick Review of Key Concepts (7 minutes)

Remind students of three core topics that will be used today:

✅ `function` – Defining a block of code to run later

```
function greet() {
  console.log("Hello!");
}
greet();
```

Ask: "What keyword do we use to define a function?" (Answer: `function`)

✅ `let` – Declaring a variable

```
let score = 0;
score = score + 1;
```

Ask: "How is `let` different from `const`? Can we change the value later?"

(Answer: Yes, `let` allows reassignment)

✅ DOM Event – Listening for clicks or other user actions

```
document.querySelector("#btn").addEventListener("click", () => {
  console.log("Button clicked!");
});
```

Ask: "What happens when someone clicks the button here?"

(Answer: It logs a message to the console)

🧑‍🏫 Teacher Tips

- Use a live coding platform (like CodePen, JSFiddle, or Replit) to show examples.
- Let students try typing out one example if time allows.

- Keep the review snappy—this section is just to refresh, not reteach.

⏳ 2. Intro to `setTimeout()` (15 minutes)

🎯 Learning Goal

- Understand how to delay an action using `setTimeout()`
- Use it to build time-based interactions in a webpage

✅ Concept: One-time delayed execution

`setTimeout()` lets you schedule code to run *once* after a specified number of milliseconds.

Basic Syntax:

```
setTimeout(callbackFunction, delayInMilliseconds);
```

You can use:

- A named function
- An anonymous arrow function (most common)
- Optional extra parameters to pass to the callback

📖 Example:

```
setTimeout(() => {
  console.log("Boom!");
}, 2000); // Waits 2 seconds, then prints "Boom!"
```

Ask students:

- "What do you expect to see after running this?"
- "How long is 2000 milliseconds in seconds?"

🧪 Student Practice (5–7 minutes)

Give students these small tasks to try in their code editors:

Task 1: Show a message after 3 seconds

```
setTimeout(() => {
  alert("Hello after 3 seconds!");
}, 3000);
```

Task 2: Try changing the delay

- What happens if you change 3000 to 5000?
- Can you show different messages at different times?

🧠 Teacher Tips

- Remind students that the delay is *not exact*—it depends on the event loop.
- Explain that `setTimeout()` does **not block** other code—it runs *asynchronously*.
- If students forget the `()` in arrow functions, remind them about syntax: `() => {}`

🔍 Optional Deeper Exploration

> “Can we pass arguments into the function being called?”

```
function greet(name) {
  console.log("Hello, " + name + "!");
}
setTimeout(greet, 2000, "Alice"); // Hello, Alice!
```

🧩 Optional Challenge (for fast learners)

Create a button that says "Surprise me"—when clicked, a message appears after 2 seconds.

(Hint: Combine `addEventListener` with `setTimeout()`)

```
document.querySelector("#surpriseBtn").addEventListener("click", () => {
  setTimeout(() => {
    alert("🎉 Surprise!");
  }, 2000);
});
```

🔁 3. Intro to `setInterval()` + `clearInterval()` (20 minutes)

🎯 Learning Goal

- Understand how to repeat actions at fixed time intervals using `setInterval()`
- Learn how to stop an interval using `clearInterval()`
- Apply this to build an interactive number counter

✅ Concept: Repeating behavior with `setInterval()`

`setInterval()` lets you run a function **repeatedly** every X milliseconds until you stop it.

Basic Syntax:

```
let intervalId = setInterval(callbackFunction, intervalInMilliseconds);
```

To stop it:

```
clearInterval(intervalId);
```

📖 Example:

```
let count = 0;
let timer = setInterval(() => {
  count++;
  console.log(count);
  if (count === 5) clearInterval(timer); // Stop after 5
}, 1000); // Run every 1 second
```

Ask students:

- "How many times will this print?"
- "What would happen if we didn't add the `clearInterval()` line?"

⚠️ Common Mistakes to Watch For

- Forgetting to store the timer ID → can't stop the interval
- Running multiple intervals at the same time → overlapping behavior
- Not using `clearInterval()` → infinite loop

👨‍💻 Student Task: Number Counter (10 minutes)

Task Description:

Build a webpage that shows a number on screen, and has two buttons:

- ✅ **Start** → starts counting up every second
- 🛑 **Stop** → stops the counter

Starter HTML:

```
<p id="counter">0</p>
<button id="startBtn">Start</button>
<button id="stopBtn">Stop</button>
```

Sample JS Logic:

```
let count = 0;
let timerId;

document.getElementById("startBtn").addEventListener("click", () => {
  timerId = setInterval(() => {
    count++;
    document.getElementById("counter").textContent = count;
  }, 1000);
});

document.getElementById("stopBtn").addEventListener("click", () => {
  clearInterval(timerId);
});
```

🧠 Teacher Tips

- Ask: "What happens if you click 'Start' multiple times?"
  - ➜ Answer: multiple timers get created → fix by clearing before starting
- Encourage students to tweak: count down instead of up, or change the speed

🧩 Optional Challenge

- Add a **Reset** button to reset the counter to 0
- Add input to let user choose the interval (e.g., 500ms, 2000ms)

🧠 Summary Talking Points

- `setInterval()` = run repeatedly
- `clearInterval()` = stop it
- You must **store the timer ID** in a variable if you want to stop it later

⏰ 4. Mini Practice: Countdown Timer (15 minutes)

🎯 Objective

Create a simple countdown timer that starts from 10 and updates on screen every second.

When it reaches 0, display the message: **"Time's up!"**

🧩 Key Skills Used

- `setInterval()` to repeat the countdown
- `clearInterval()` to stop when it hits 0
- DOM manipulation to update the number and message

✨ Hint for Students

> "Use a variable to keep track of the time left.
>
> Each second, decrease it by 1.
>
> When the value hits 0, stop the timer and change the message."

✅ Step-by-Step Plan

HTML Starter:

```html
<p id="countdown">10</p>
<button id="startCountdown">Start Countdown</button>
```

Sample JS Logic:

```javascript
document.getElementById("startCountdown").addEventListener("click", () => {
  let timeLeft = 10;
  const countdownDisplay = document.getElementById("countdown");

  const timer = setInterval(() => {
    countdownDisplay.textContent = timeLeft;
    timeLeft--;

    if (timeLeft < 0) {
      clearInterval(timer);
      countdownDisplay.textContent = "Time's up!";
    }
  }, 1000);
});
```

🧠 Teacher Tips

- Remind students that `setInterval()` won't stop automatically
- Ask: "What happens if you click the button more than once?"
    - ➔ Answer: Multiple timers run at once → can fix later with a more advanced check

---

🔧 Optional Tweaks

- Allow users to set their own countdown start time using an `<input>` field
- Add sound or color changes when countdown ends

✅ Completion Checklist

By the end of this activity, students should:

- Successfully show a countdown from 10 to 0
- Display "Time's up!" at the end
- Understand how to stop a timer using `clearInterval()`

🎮 5. Mini Project: Reaction Timer Game (30–40 minutes)

🎯 Project Goal

Create a simple reaction game where:

1. The user clicks a **"Start"** button.
2. After a random delay (2–5 seconds), a colored box appears.

3. The user clicks the box as fast as possible.
4. The app calculates and displays the reaction time in milliseconds.

🔧 Concepts Practiced

- `setTimeout()` – create random delay
- `Date.now()` – track time in milliseconds
- `DOM` – dynamically show/hide elements and update text

🧠 Step-by-Step Breakdown

✅ 1. HTML Structure (Starter)

```html
<button id="startBtn">Start</button>
<div id="box" style="width: 100px; height: 100px; background: red; display: n
one; position: relative; margin-top: 20px;"></div>
<p id="message"></p>
```

✅ 2. JavaScript Logic

```javascript
const startBtn = document.getElementById("startBtn");
const box = document.getElementById("box");
const message = document.getElementById("message");

let startTime;
let timeoutId;

startBtn.addEventListener("click", () => {
  message.textContent = "Wait for the box...";
  box.style.display = "none";

  const randomDelay = Math.floor(Math.random() * 3000) + 2000; // 2000–50
00ms
```

```
  timeoutId = setTimeout(() => {
    box.style.display = "block";
    startTime = Date.now(); // Record time when box appears
  }, randomDelay);
});

box.addEventListener("click", () => {
  const reactionTime = Date.now() - startTime;
  message.textContent = `Your reaction time: ${reactionTime} ms`;
  box.style.display = "none";
});
```

🧠 Teaching Tips

- Encourage students to break the project into **small parts**:
  1. Make a box appear after delay
  2. Click the box → show message
  3. Add reaction time tracking with `Date.now()`
- Walk through each part before letting students code on their own

---

🛠️ Optional Enhancements / Challenges

💥 Too Soon Detection

If the user clicks **before** the box appears, show a "Too soon!" warning:

```
let boxVisible = false;

startBtn.addEventListener("click", () => {
  boxVisible = false;
  // same as before...
  timeoutId = setTimeout(() => {
    box.style.display = "block";
    startTime = Date.now();
    boxVisible = true;
  }, randomDelay);
});

box.addEventListener("click", () ⇒ {
  if (!boxVisible) {
    message.textContent = "Too soon! Wait for the box.";
    clearTimeout(timeoutId);
  } else {
    const reactionTime = Date.now() - startTime;
    message.textContent = `Your reaction time: ${reactionTime} ms`;
    box.style.display = "none";
  }
});
```

🧠 Track Multiple Attempts

Let the player try **3 times**, then show average reaction time:

- Store times in an array
- After 3 rounds, calculate the average and display it

🎨 Add Style & Fun

- Randomize box color or position
- Use sound or animations
- Add a countdown before the game starts (e.g., “Get ready…”)

✅ Project Checklist

By the end of this activity, students should be able to:

- Use `setTimeout()` with a random delay
- Capture time using `Date.now()`
- Handle clicks and DOM element visibility

- Calculate and show time difference

🧩 Wrap-Up Discussion (2 min after project)

- "Was it harder than it looked?"
- "How could we add a score or leaderboard?"
- "Where else might you use a reaction timer in real life?"

---

6. Extension Challenge (Optional or Homework)

- Play 3 times → show average reaction time
- If user clicks before box appears, show "Too soon!" message

---

7. Recap & Q&A (10 minutes)

- Discuss the difference between `setTimeout` vs. `setInterval`
- Ask: Where could timers be useful in real life or games?
- Preview next lesson (e.g., Progress Bars or Local Storage)

✅ Key Takeaways

- `setTimeout()` = do something later, once
- `setInterval()` = do something repeatedly
- `clearInterval()` = stop the repeating
- Timers are essential for making interactive apps and games