JavaScript Timers — Full Guide With Explained Examples + Interview Tricks

This document is written in simple English, with clear definitions, explained examples, expected output, and interview-style trick questions (including for-loops + setTimeout). At the end, you can download this document.

---

1. setTimeout

Definition (Simple English)

`setTimeout` waits for a number of milliseconds and then runs a function **one time**. Your program does NOT stop while waiting.

Why do we use setTimeout?

- To delay something
- To run a task later
- To show messages after a pause
- To simulate loading
- To wait before retrying something

Syntax

```
const id = setTimeout(callback, delayInMilliseconds, arg1, arg2);
```

Example 1: Basic Delay

```
setTimeout(() => {
  console.log("Runs after 1 second");
}, 1000);
```

Explanation:
- Input: delay = 1000 ms (1 second)
- After 1 second, it prints the message.

Output:

```
Runs after 1 second
```

Example 2: Passing Arguments

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

setTimeout(greet, 2000, "Arun");
```

Explanation:
- After 2 seconds, it calls `greet("Arun")`
- So the output appears after 2 seconds.

Output:

```
Hello Arun
```

Example 3: Canceling a Timeout

```
const id = setTimeout(() => {
  console.log("This will NOT run");
}, 3000);

clearTimeout(id);
```

Explanation:
- The timeout is scheduled, but immediately cancelled.
- Nothing prints.

Output:
(No output)

---

2. clearTimeout

Definition

`clearTimeout` stops a timeout from running.

Syntax

```
clearTimeout(timeoutId);
```

Example 1

```
const id = setTimeout(() => console.log("Hello"), 2000);
clearTimeout(id);
```

Explanation: Cancelled immediately.

Output: No output.

Example 2: Cancel on Button Click

```
const id = setTimeout(() => console.log("Sent"), 3000);
button.onclick = () => clearTimeout(id);
```

Explanation: If the user clicks, the timeout is cancelled.

Output:
Depends → If clicked before 3 seconds, nothing prints.

---

3. setInterval

Definition

Repeats a function every X milliseconds **until you stop it**.

Syntax

```
const id = setInterval(callback, delayInMilliseconds);
```

Example 1: Simple Repeating

```
setInterval(() => {
  console.log("tick");
}, 1000);
```

Explanation: Prints "tick" every 1 second.

Output:

```
tick
tick
tick ... (forever)
```

Example 2: Counter

```
let count = 1;
setInterval(() => {
  console.log(count);
  count++;
}, 500);
```

Explanation: Every 0.5 seconds prints a number.

Output:

```
1
2
3
4 ...
```

Example 3: Passing Arguments

```
function show(name) {
  console.log("Hi", name);
}
setInterval(show, 1000, "Dev");
```

Explanation: Every second prints "Hi Dev".

---

4. clearInterval

Definition

Stops a repeating interval.

Syntax

```
clearInterval(intervalId);
```

Example 1: Stop After 3 Ticks

```
let count = 0;
const id = setInterval(() => {
  count++;
  console.log(count);
  if (count === 3) clearInterval(id);
}, 1000);
```

Explanation:
- Prints 1, 2, 3
- Stops

Output:

```
1
2
3
```

Example 2: Stop After 5 Seconds

```
const id = setInterval(() => console.log("tick"), 1000);
setTimeout(() => clearInterval(id), 5000);
```

Explanation:
- Interval starts ticking every second
- After 5 seconds timeout cancels it

Output:

```
tick
tick
tick
tick
tick
```

(Stops after 5)

---

5. INTERVIEW TRICK QUESTIONS

These are VERY common.

Question 1: What is the output?

```
for (var i = 1; i <= 5; i++) {
  setTimeout(() => console.log(i), 1000);
}
```

Explanation:

- `var` is function-scoped, not block-scoped
- By the time the timeout runs, loop is finished
- So `i` becomes 6

Output:

```
6
6
6
6
6
```

Question 2: Fix the above code to print 1 2 3 4 5

Solution A: Use `let`

```
for (let i = 1; i <= 5; i++) {
  setTimeout(() => console.log(i), 1000);
}
```

Output:

```
1
2
3
4
5
```

Solution B: IIFE

```
for (var i = 1; i <= 5; i++) {
  (function(i){
    setTimeout(() => console.log(i), 1000);
  })(i);
}
```

Question 3: What is the output?

```
console.log("A");
setTimeout(() => console.log("B"), 0);
console.log("C");
```

Explanation:

Even with 0 delay, timeout happens later.

Output:

```
A
C
B
```

Question 4: Explain why setTimeout is asynchronous.

Because JavaScript hands the callback to the browser timer API, and the event loop picks it later.

---

6. INTERVIEW TASK: Create a Timer Using setInterval (Common)

Requirement:
Create a countdown from 5 to 0.

```
let num = 5;
const id = setInterval(() => {
  console.log(num);
  num--;
  if (num < 0) clearInterval(id);
}, 1000);
```

Output:

```
5
4
3
2
1
0
```

---

7. INTERVIEW TASK: Write Your Own setInterval Using setTimeout

```
function myInterval(fn, delay) {
  function repeat() {
    fn();
    setTimeout(repeat, delay);
  }
  setTimeout(repeat, delay);
}

myInterval(() => console.log("Hello"), 1000);
```

---

8. Canvas — Explanation + Code

Interviewers sometimes ask about the `<canvas>` API.

Basic Canvas Example

```
<canvas id="box" width="300" height="300"></canvas>
<script>
const c = document.getElementById("box");
const ctx = c.getContext("2d");

ctx.fillStyle = "blue";
ctx.fillRect(20, 20, 150, 100); // x, y, width, height
</script>
```

Timer + Canvas (Interview Level)

```
<canvas id="c" width="300" height="300"></canvas>
<script>
const c = document.getElementById("c");
const ctx = c.getContext("2d");
let x = 0;

setInterval(() => {
  ctx.clearRect(0, 0, 300, 300);
  ctx.fillStyle = "red";
  ctx.fillRect(x, 50, 50, 50);
  x += 5;
}, 100);
</script>
```

Explanation:
- Every 100 ms, the red square moves slightly.

---

9. Summary

- `setTimeout` → run once later
- `setInterval` → run repeatedly
- Always clear timers
- Common interview traps: for-loop + var + setTimeout
- Canvas often combined with timers for animations

If you want, I can also add: - React versions of all examples - Node.js versions - More tricky interview puzzles - A downloadable PDF version