

[JavaScript 14]

Roi Yehoshua 2018

[Our targets for today]

- setTimeout and setInterval functions
- How to clear created timeout or interval
- How to manage timers in Javascript
- How to use browser Local storage

[Scheduling: setTimeout and setInterval]

- We may decide to execute a function not right now, but at a certain time later
- That's called "scheduling a call"
- There are two methods for it:
 - **setTimeout()** allows to run a function once after the interval of time
 - **setInterval()** allows to run a function regularly with the interval between the runs
- These methods are supported in all browsers and Node.JS

[setTimeout]

→ The syntax:

```
let timerId = setTimeout(func|code, delay[, arg1, arg2...])
```

→ **func|code** – a function or a string of code to execute. Usually, that's a function.

→ **delay** - the delay before run, in milliseconds (1000 ms = 1 second)

→ **arg1, arg2...** - arguments for the function

→ For instance, this code calls sayHi() after one second:

```
function sayHi() { alert('Hello');  
}  
setTimeout(sayHi, 1000);
```

→ You can also use an arrow function:

```
setTimeout(() => alert('Hello'), 1000);
```

[setTimeout]

→ Example for passing arguments to the schedules function:

```
function sayHi(phrase, who) { alert(phrase + ', ' + who);  
}  
  
setTimeout(sayHi, 1000, "Hello", "John"); // Hello, John
```

→ Novice developers sometimes make a mistake by adding () after the function:

```
// wrong! setTimeout(sayHi(), 1000);
```

→ That doesn't work, because setTimeout expects a reference to function, and here sayHi() runs the function, and the *result of its execution* is passed to setTimeout

→ In our case the result of sayHi() is undefined (the function returns nothing), so nothing is scheduled

[Canceling with clearTimeout]

→ A call to `setTimeout` returns a “timer identifier” **timerId**, that we can use to cancel the execution

→ The syntax to cancel:

```
let timerId = setTimeout(...); clearTimeout(timerId);
```

→ In the code below, we schedule the function and then cancel it

→ As a result, nothing happens:

```
let timerId = setTimeout(() => alert("never happens"), 1000); alert(timerId); //  
timer identifier  
  
clearTimeout(timerId);
```

[setInterval]

→ The **setInterval** method has the same syntax as **setTimeout**:

```
let timerId = setInterval(func|code, delay[, arg1, arg2...])
```

→ All arguments have the same meaning

→ But unlike **setTimeout** it runs the function not only once, but regularly after the given interval of time

→ To stop further calls, you can call **clearInterval**(timerId)

→ The following example shows a message every 2 seconds, and stops after 5 seconds:

```
// repeat with the interval of 2 seconds
let timerId = setInterval(() => alert('tick'), 2000);

// after 5 seconds stop
setTimeout(() => { clearInterval(timerId); alert('stop'); }, 5000);
```

→ In Chrome, Opera and Safari the internal timer becomes “frozen” while showing alert/prompt

→ So if you run the code above and don't dismiss the alert window after some time, then the next alert will be shown after 2 more seconds (timer did not tick during the alert)

[setTimeout(...,0)]

- There's a special use case: `setTimeout(func, 0)`
- This schedules the execution of `func` as soon as possible
- But scheduler will invoke it only after the current code is complete
- So the function is scheduled to run “right after” the current, i.e., *asynchronously*.
- For instance, this outputs “Hello”, then immediately “World”:

```
setTimeout(() => alert("World"), 0);  
  
alert("Hello");
```

- The first line “puts the call into calendar after 0ms”. But the scheduler will only “check the calendar” after the current code is complete, so “Hello” is first, and “World” – after it.

[Splitting CPU-Hungry Tasks]

- There's a trick to split CPU-hungry tasks using setTimeout
- Let's take a simpler example for consideration
- We have a function to count from 1 to 2000000000:

```
let i = 0;
let start = Date.now();

function count() {
  // do a heavy job
  for (let j = 0; j < 2e9; j++) {
    i++;
  }
  alert("Done in " + (Date.now() - start) + 'ms');
}
count();
```

- If you run it, the CPU will hang - the whole JavaScript actually is paused, no other actions work until it finishes

[Splitting CPU-Hungry Tasks]

→ Let's split the job using the nested setTimeout:

```
let i = 0;
let start = Date.now();

function count() {
  // do a piece of the heavy job
  do {
    i++;
  } while (i % 1e6 !== 0);

  if (i === 1e9) {
    alert("Done in " + (Date.now() - start) + 'ms');
  } else {
    setTimeout(count, 0); // schedule the new call
  }
}
count();
```

→ Now the browser UI is fully functional during the “counting” process

→ Pauses between count executions provide just enough “breath” for the JavaScript engine to do something else, to react to other user actions

[Local Storage]

- Similar to the cookies, but can be stored endless and capable of storing more data.
- Bind to the site and accessible across tabs and windows.
- Stored on local machine of the user.
- Preserved through sessions.

[Domains and Subdomains]

- Not dependent on protocol. “<https://example.com>” and “<http://example.com>” will be treated as a same domain.
- Domains local storage can be accessed by subdomains. “<https://example.com>” will be accessible to “<https://text.example.com>”.
- The same works for accessing domains from the subdomains.
- Subdomains can't access each other. So “<https://text.example.com>” won't be able to reach local storage of “<https://mail.example.com>”.

[Working with local storage data]

→ Setting value “Tom” to storage item “myCat”:

```
localStorage.setItem('myCat', 'Tom');
```

→ Getting value of storage item “myCat”:

```
localStorage.getItem('myCat');
```

→ Removing item “myCat” from the local storage:

```
localStorage.removeItem('myCat');
```

→ Clearing whole storage:

```
localStorage.clear();
```

[Storage quota]

- Firefox: 5mb
- Chrome: 10mb
- Internet Explorer: 10mb
- Safari: 5mb

[Control questions]

1. How can we execute a delayed code?
2. What is the difference between `setTimeout` and `setInterval`?
3. What should be avoided upon working with timers?
4. What is `localStorage`?
5. What differs `localStorage` from cookies?

[Materials]

Core materials:

<https://learn.javascript.ru/settimeout-setinterval>

<https://developer.mozilla.org/ru/docs/Web/API/Window/localStorage>

Additional materials:

<http://htmlbook.ru/html5/storage>

Video materials:

<https://www.youtube.com/watch?v=puxkpNbCh0w>