Sometimes, you're going to need to build a JavaScript countdown clock. You may have an event, a sale, a promotion, or a game. You can build a clock in raw JavaScript rather than reaching for the nearest plugin. While there are many great clock plugins, here are the benefits you'll get from using raw JavaScript:

- Your code will be lightweight because it will have zero dependencies.
- Your website will perform better. You won't need to load external scripts and stylesheets.
- You'll have more control. You will have built the clock to behave exactly the way you want it to (rather than trying to bend a plugin to your will).

So, without further ado, here's how to make your own countdown clock in a mere 18 lines of JavaScript.

## Basic Clock: Count down to a Specific Date or Time

Here's a quick outline of the steps involved in creating a basic clock:

- Set a valid end date.
- Calculate the time remaining.
- Convert the time to a usable format.
- Output the clock data as a reusable object.
- Display the clock on the page, and stop the clock when it reaches zero.

### Set a Valid End Date

First, you'll need to set a valid end date. This should be a string in any of the formats understood by JavaScript's Date.parse() method. For example:

The ISO 8601 format:

```
var deadline = '2015-12-31';
```

The short format:

```
var deadline = '31/12/2015';
```

Or, the long format:

```
var deadline = 'December 31 2015';
```

Each of these formats allows you to specify an exact time and a time zone (or an offset from UTC in the case of ISO dates). For example:

```
var deadline = 'December 31 2015 23:59:59
GMT+0200';
```

### Calculate the Time Remaining

The next step is to calculate the time remaining. We need to write a function that takes a string representing a given end time (as outlined above). We then calculate the difference between that time and the current time. Here's what that looks like:

```
function getTimeRemaining(endtime){
```

```
  var t = Date.parse(endtime) - Date.parse(new
Date());
  var seconds = Math.floor( (t/1000) % 60 );
  var minutes = Math.floor( (t/1000/60) % 60 );
  var hours = Math.floor( (t/(1000*60*60)) % 24 );
  var days = Math.floor( t/(1000*60*60*24) );
  return {
    'total': t,
    'days': days,
    'hours': hours,
    'minutes': minutes,
    'seconds': seconds
  };
}
```

First, we're creating a variable `t`, to hold the remaining time until the deadline. The `Date.parse()` function converts a time string into a value in milliseconds. This allows us to subtract two times from each other and get the amount of time in between.

```
var t = Date.parse(endtime) - Date.parse(new
Date());
```

### Convert the Time to a Usable Format

Now we want to convert the milliseconds to days, hours, minutes, and seconds. Let's use seconds as an example:

```
var seconds = Math.floor( (t/1000) % 60 );
```

Let’s break down what’s going on here.

1. Divide milliseconds by 1000 to convert to seconds: `(t/1000)`
2. Divide the total seconds by 60 and grab the remainder. You don’t want all of the seconds, just those remaining after the minutes have been counted: `(t/1000) % 60`
3. Round this down to nearest whole number. This is because you want complete seconds, not fractions of seconds: `Math.floor( (t/1000) % 60 )`

Repeat this logic to convert the milliseconds to minutes, hours, and days.

### Output the Clock Data as a Reusable Object

With the days, hours, minutes, and seconds prepared, we’re now ready to return the data as a reusable object:

```
return {
  'total': t,
  'days': days,
  'hours': hours,
  'minutes': minutes,
  'seconds': seconds
};
```

This object allows you to call your function and get any of the calculated values. Here’s an example of how you’d get the remaining minutes:

```
getTimeRemaining(deadline).minutes
```

Convenient, right?

### Display the Clock and Stop It When It Reaches Zero

Now that we have a function that spits out the days, hours, minutes, and seconds remaining, we can build our clock. First we'll create the following HTML element to hold our clock:

```
<div id="clockdiv"></div>
```

Then we'll write a function that outputs the clock data inside our new div:

```
function initializeClock(id, endtime){
  var clock = document.getElementById(id);
  var timeinterval = setInterval(function(){
    var t = getTimeRemaining(endtime);
    clock.innerHTML = 'days: ' + t.days + '<br>' +
                      'hours: '+ t.hours + '<br>' +
                      'minutes: ' + t.minutes +
'<br>' +
                      'seconds: ' + t.seconds;
    if(t.total<=0){
      clearInterval(timeinterval);
    }
  },1000);
}
```

This function takes two parameters. These are the id of the element that contains our clock, and the countdown's end time. Inside the function, we'll declare a `clock` variable and use it to store a reference

to our clock container div. This means we don't have to keep querying the DOM.

Next, we'll use `setInterval` to execute an anonymous function every second. This function will do the following:

- Calculate the remaining time.
- Output the remaining time to our div.
- If the remaining time gets to zero, stop the clock.

At this point, the only remaining step is to run the clock like so:

```
initializeClock('clockdiv', deadline);
```

Congratulations! You now have a basic clock in just 18 lines of JavaScript.

## Prepare Your Clock for Display

Before styling the clock, we'll need to refine things a little.

- Remove the initial delay so your clock shows up immediately.
- Make the clock script more efficient so it doesn't continuously rebuild the whole clock.
- Add leading zeros as desired.

### Remove the Initial Delay

In the clock, we've used `setInterval` to update the display every second. This is fine most of the time, except in the beginning when

there will be a one-second delay. In order to remove this delay, we'll have to update the clock once before the interval starts.

Let's move the anonymous function that we're passing to `setInterval` into its own separate function. We can name this function `updateClock`. Call the `updateClock` function once outside of `setInterval`, and then call it again inside `setInterval`. This way, the clock shows up without the delay.

In your JavaScript, replace this:

```
var timeinterval = setInterval(function(){ ...
},1000);
```

With this:

```
function updateClock(){
  var t = getTimeRemaining(endtime);
  clock.innerHTML = 'days: ' + t.days + '<br>' +
                    'hours: '+ t.hours + '<br>' +
                    'minutes: ' + t.minutes +
'<br>' +
                    'seconds: ' + t.seconds;
  if(t.total<=0){
    clearInterval(timeinterval);
  }
}

updateClock(); // run function once at first to
avoid delay
```

```
var timeinterval = setInterval(updateClock,1000);
```

### Avoid Continuously Rebuilding the Clock

We need to make the clock script more efficient. We'll want to update only the numbers in the clock instead of rebuilding the entire clock every second. One way to accomplish this is to put each number inside a `span` tag and only update the content of those spans.

Here's the HTML:

```
<div id="clockdiv">
    Days: <span class="days"></span><br>
    Hours: <span class="hours"></span><br>
    Minutes: <span class="minutes"></span><br>
    Seconds: <span class="seconds"></span>
</div>
```

Now let's get a reference to those elements. Add the following code right after where the `clock` variable is defined

```
var daysSpan = clock.querySelector('.days');
var hoursSpan = clock.querySelector('.hours');
var minutesSpan = clock.querySelector('.minutes');
var secondsSpan = clock.querySelector('.seconds');
```

Next, we need to alter the `updateClock` function to update only the numbers. The new code will look like this:

```
function updateClock(){
    var t = getTimeRemaining(endtime);

    daysSpan.innerHTML = t.days;
```

```
    hoursSpan.innerHTML = t.hours;
    minutesSpan.innerHTML = t.minutes;
    secondsSpan.innerHTML = t.seconds;

    ...
}
```

Add Leading Zeros

Now that the clock is no longer rebuilding every second, we have one more thing to do: add leading zeros. For example, instead of having the clock show 7 seconds, it would show 07 seconds. One simple way to do this is to add a string of '0' to the beginning of a number and then slice off the last two digits.

For example, to add a leading zero to the "seconds" value, you'd change this:

```
secondsSpan.innerHTML = t.seconds;
```

to this:

```
secondsSpan.innerHTML = ('0' +
t.seconds).slice(-2);
```

If you'd like, you can add leading zeros to the minutes and hours as well. If you've come this far, congratulations! Your clock is now ready for display.

## Taking it Further

The following examples demonstrate how to expand the clock for certain use cases. They are all based on the basic example seen above.

## Schedule the Clock Automatically

Let's say we want the clock to show up on certain days but not others. For example, we might have a series of events coming up and don't want to manually update the clock each time. Here's how to schedule things in advance.

Hide the clock by setting its `display` property to `none` in the CSS. Then add the following to the `initializeClock` function (after the line that begins with `var clock`). This will cause the clock to only display once the `initializeClock` function is called:

```
clock.style.display = 'block';
```

Next we can specify the dates between which the clock should show up. This will replace the `deadline` variable:

```
var schedule = [

    ['Jul 25 2015', 'Sept 20 2015'],

    ['Sept 21 2015', 'Jul 25 2016'],

    ['Jul 25 2016', 'Jul 25 2030']
```

```
];
```

Each element in the `schedule` array represents a start date and an end date. As noted above, it is possible to include times and time zones, but I used plain dates here to keep the code readable.

Finally, when a user loads the page, we need to check if we are within any of the specified time frames. This code should replace the previous call to the `initializeClock` function.

```
// iterate over each element in the schedule

for(var i=0; i<schedule.length; i++){

  var startDate = schedule[i][0];

  var endDate = schedule[i][1];



  // put dates in milliseconds for easy comparisons

  var startMs = Date.parse(startDate);

  var endMs = Date.parse(endDate);

  var currentMs = Date.parse(new Date());
```

```
  // if current date is between start and end
dates, display clock

  if(endMs > currentMs && currentMs >= startMs ){

       initializeClock('clockdiv', endDate);

  }

}
```

Now you can schedule your clock in advance without having to update it by hand. You may shorten the code if you wish. I made mine verbose for the sake of readability.

## Set Timer for 10 Minutes from When the User Arrives

It may be necessary to set a countdown for a given amount of time after the user arrives or starts a specific task. We'll set a timer for 10 minutes here, but you can use any amount of time you want.

All we need to do here is replace the `deadline` variable with this:

```
var timeInMinutes = 10;

var currentTime = Date.parse(new Date());
```

```
var deadline = new Date(currentTime +
timeInMinutes*60*1000);
```

This code takes the current time and adds ten minutes. The values are converted into milliseconds, so they can be added together and turned into a new deadline.

Now we have a clock that counts down ten minutes from when the user arrives. Feel free to play around and try different lengths of time.

## Maintain Clock Progress across Pages

Sometimes it's necessary to preserve the state of the clock for more than just the current page. If we wanted to set a 10-minute timer across the site, we wouldn't want it to reset when the user goes to a different page.

One solution is to save the clock's end time in a cookie. That way, navigating to a new page won't reset the end time to ten minutes from now.

Here's the logic:

1. If a deadline was recorded in a cookie, use that deadline.
2. If the cookie isn't present, set a new deadline and store it in a cookie.

To implement this, replace the `deadline` variable with the following:

```
// if there's a cookie with the name myClock, use
that value as the deadline

if(document.cookie &&
document.cookie.match('myClock')){

  // get deadline value from cookie

  var deadline =
document.cookie.match(/(^|;)myClock=([^;]+)/)[2];

}



// otherwise, set a deadline 10 minutes from now
and

// save it in a cookie with that name

else{

  // create deadline 10 minutes from now

  var timeInMinutes = 10;

  var currentTime = Date.parse(new Date());
```

```
  var deadline = new Date(currentTime +
timeInMinutes*60*1000);



  // store deadline in cookie for future reference

  document.cookie = 'myClock=' + deadline + ';
path=/; domain=.yourdomain.com';

}
```

This code makes use of cookies and regular expressions, both of which are separate topics in their own right. For that reason, I won't go into too much detail here. The one important thing to note is that you'll need to change `.yourdomain.com` to your actual domain. If you have any questions concerning this, let me know in the comments.

## An Important Caveat about Client-Side Time

JavaScript dates and times are taken from the user's computer. That means the user can affect a JavaScript clock by changing the time on their machine. In most cases, this won't matter. But in the case of something super sensitive, it will be necessary to get the time from the server. That can be done with a bit of PHP or Ajax, both of which are beyond the scope of this tutorial.

After getting the time from the server, we can work with it using the same techniques from this tutorial.

## Conclusion

We've covered how to make a basic countdown clock and prepare it for efficient display. We've covered scheduling, absolute versus relative times, and preserving state.

### What's Next?

Play around with your clock code. Try adding some creative styles, or new features (such as pause and resume buttons).