# 13.2 Timer

*Die `setInterval()` Methode ruft eine Funktion in bestimmten Intervallen auf. Jedes Intervall wird in Millisekunden angegeben. Lässt man den Timer alle 1000 ms ticken, dann wird zu jeder Sekunde die Funktion aufgerufen. 1000 ms = 1 Sekunde!*

## JS `setInterval(`*Funktion, Millisekunden*`)`

Der Timer kann als Objekt dimensioniert werden. Mit der Vereinbarung wird der Timer auch gestartet. Will man der Funktion Werte übergeben, dann werden die Parameter nach den Millisekunden hinzugefügt.
z. B.: `setInterval(`*Funktion, Millisekunden, **param1**, **param2**, ...*`);`

```
<script>
  var meinTimer = setInterval(meineFunktion, 1000);

  function meineFunktion() {
     var aktuellesDatum = new Date();
     aktuellesDatum = aktuellesDatum.toLocaleTimeString('de-DE');
     document.getElementById("ausgabe").innerHTML = aktuellesDatum;}
</script>

<p id="ausgabe"></p>
```

## JS `clearInterval();`

Stoppt den Timer. Der Methode wird das Timer Objekt übergeben.
z. B. `clearInterval(`*meinTimer*`);`

```
<script>
    var lauf = 10;
    var meinTimer = setInterval(meineFunktion, 500);

    function meineFunktion() {
         lauf = lauf - 1;
         document.getElementById("ausgabe").innerHTML = lauf;
         if (lauf == 0) {clearInterval(meinTimer);}}
</script>
<p id="ausgabe"></p>
```

*Im Beispiel wird von 10 bis 0 hinunter gezählt. Bei 0 wird der Timer gestoppt. Die Variable `lauf` und das Timer-Objekt `meinTimer` sind global und stehen damit jeder Funktion zur Verfügung.*

## JS `setTimeout(`*Funktion, Millisekunden*`)`

Die Methode ruft eine Funktion nach einer bestimmten Anzahl von Millisekunden auf. Die Funktion wird nur einmal aufgerufen.
Mit `clearTimeout()` kann man das Aufrufen verhindern.

```
<script>
    var meinTimer = setTimeout(meineFunktion, 10000);

    function meineFunktion() {
       alert("Ich habe 10 Sekunden gewartet!");}
</script>
```