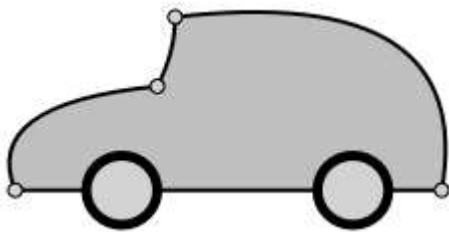


Animation

Кривые Безье

Виды кривых Безье

Кривая Безье задаётся опорными точками.



Математика

Координаты кривой описываются в зависимости от параметра $t \in [0, 1]$

- Для двух точек:

$$P = (1-t)P_1 + tP_2$$

- Для трёх точек:

$$P = (1-t)^2P_1 + 2(1-t)tP_2 + t^2P_3$$

- Для четырёх точек:

$$P = (1-t)^3P_1 + 3(1-t)^2tP_2 + 3(1-t)t^2P_3 + t^3P_4$$

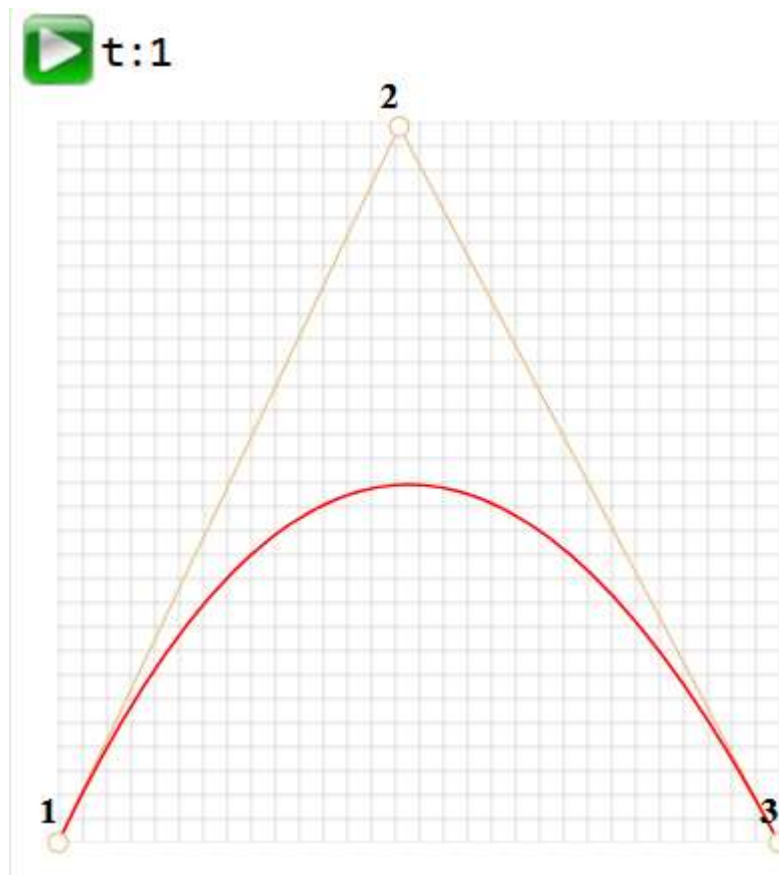
Вместо P_i нужно подставить координаты i -й опорной точки (x_i, y_i) .

Эти уравнения векторные, то есть для каждой из координат:

- $x = (1-t)^2x_1 + 2(1-t)tx_2 + t^2x_3$
- $y = (1-t)^2y_1 + 2(1-t)ty_2 + t^2y_3$

Вместо $x_1, y_1, x_2, y_2, x_3, y_3$ подставляются координаты трёх опорных точек, и в то время как t пробегает множество от 0 до 1 , соответствующие значения (x, y) как раз и образуют кривую.

Рисование «де Кастельжо»



Алгоритм построения кривой по «методу де Кастельжо»

1. Рисуем опорные точки. В примере выше это 1, 2, 3.
2. Строятся отрезки между опорными точками $1 \rightarrow 2 \rightarrow 3$. На рисунке выше они коричневые.
3. Параметр t пробегает значения от 0 до 1. В примере выше использован шаг 0.05, т.е. в цикле 0, 0.05, 0.1, 0.15, ... 0.95, 1.

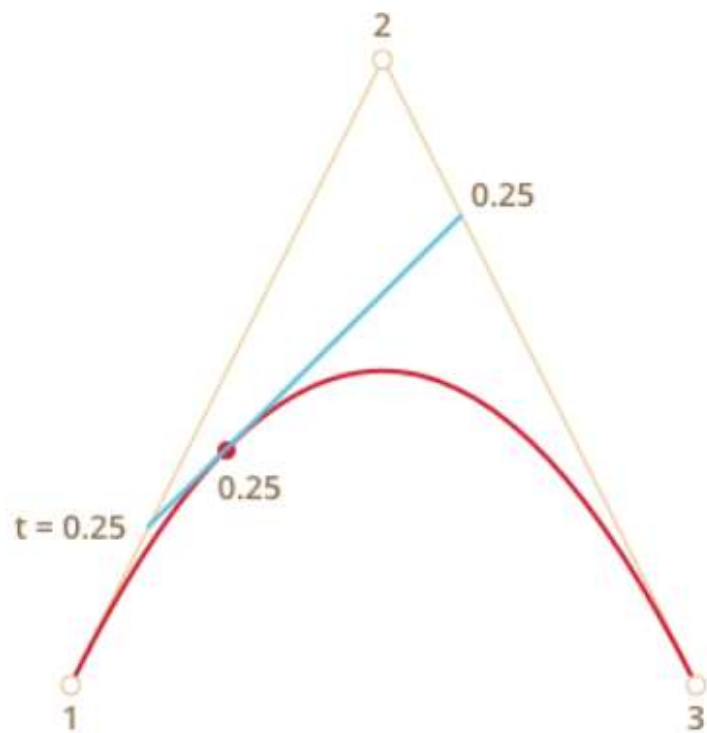
Для каждого из этих значений t :

- На каждом из коричневых отрезков берётся точка, находящаяся от начала на расстоянии от 0 до t пропорционально длине. Так как коричневых отрезков – два, то и точек две штуки.

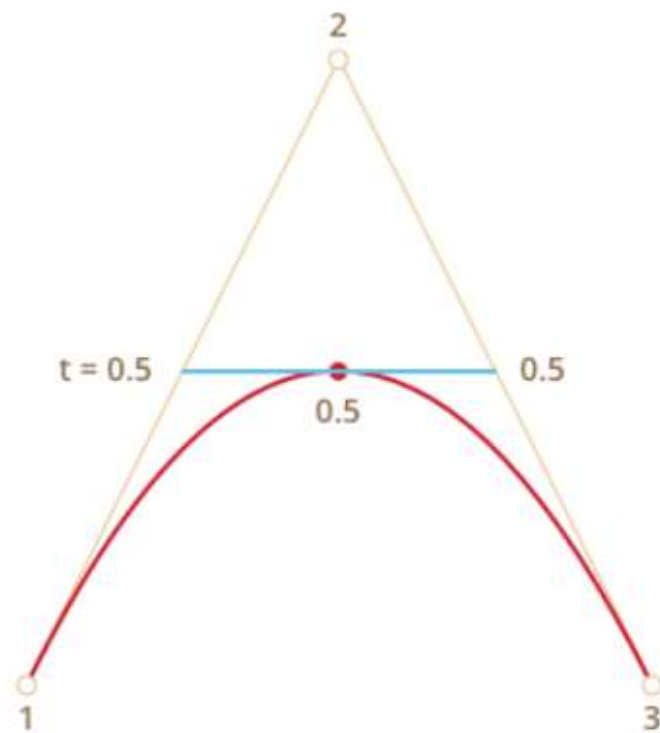
Например, при $t=0$ – точки будут в начале, при $t=0.25$ – на расстоянии в 25% от начала отрезка, при $t=0.5$ – 50%(на середине), при $t=1$ – в конце отрезков.

- Эти точки соединяются. На рисунке ниже соединяющий их отрезок изображён синим.

При $t=0.25$



При $t=0.5$



4. На **получившемся** отрезке берётся точка на расстоянии, соответствующем t . То есть, для $t=0.25$ (первый рисунок) получаем точку в конце первой четверти отрезка, для $t=0.5$ (второй рисунок) – в середине отрезка. На рисунках выше эта точка отмечена **красным**.
5. По мере того как t пробегает последовательность от 0 до 1 , каждое значение t добавляет к красной кривой точку. **Совокупность таких точек для всех значений t образуют кривую Безье.**

CSS-анимации

CSS transitions

```
1 <button id="color">Кликни меня</button>
2
3 <style>
4   #color {
5     transition-property: background-color;
6     transition-duration: 3s;
7   }
8 </style>
9
10 <script>
11   color.onclick = function() {
12     this.style.backgroundColor = 'red';
13   }
14 </script>
```

Кликни меня



Кликни меня

- transition-property
- transition-duration
- transition-timing-function
- transition-delay

transition: property duration timing-function delay

transition: font-size 3s, color 2s;

transition-property

- Список свойств, которые будут анимироваться, например: left, margin-left, height, color.
- Анимировать можно не все свойства, но многие. Значение all означает «анимировать все свойства».

transition-duration

- Продолжительность анимации, задаётся в секундах s или ms.

transition-delay

- Задержка до анимации. Например, transition-delay: 1s.
- Возможны отрицательные значения, при этом анимация начнётся с середины.

transition-timing-function

- Временная функция, которая задаёт, как процесс анимации будет распределён во времени, например начнётся ли анимация медленно, чтобы потом ускориться или наоборот.
- У него есть два основных вида значения: кривая Безье и по шагам.

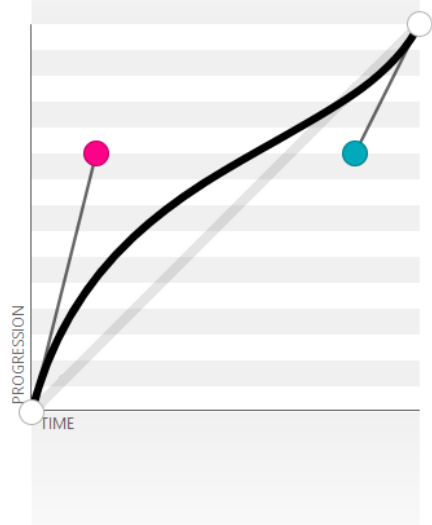
Кривая Безье

- Синтаксис для задания кривой Безье в CSS: `cubic-bezier(x2, y2, x3, y3)`.
- В нём указываются координаты только двух точек: второй и третьей, так как первая и последняя фиксированы.
- <http://cubic-bezier.com/>

cubic-bezier(.17,.67,.83,.67) [SAVE](#)

Preview & compare [GO!](#)

Duration: 1 second



[Tweet](#)

Library [IMPORT](#) [EXPORT](#)

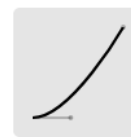
Click on a curve to compare it with the current one.



ease



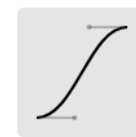
linear



ease-in



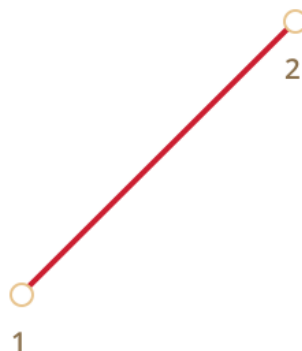
ease-out



ease-in-out

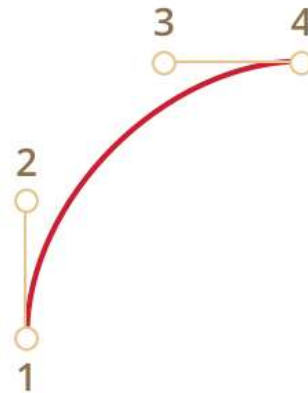
Tip: Right click on any library curve and select "Copy Link Address" to get a permalink to it which you can share with others

Линейная анимация



```
1 .train {  
2   left: 0;  
3   transition: left 5s cubic-bezier(0, 0, 1, 1);  
4   /* JavaScript ставит значение left: 450px */  
5 }
```

Параболическая анимация



```
1 .train {  
2   left: 0;  
3   transition: left 5s cubic-bezier(0, .5, .5, 1);  
4   /* JavaScript ставит значение left: 450px */  
5 }
```

Существует несколько стандартных обозначений кривых: linear, ease, ease-in, ease-out и ease-in-out.

ease*

(0.25, 0.1, 0.25, 1.0)



ease-in

(0.42, 0, 1.0, 1.0)



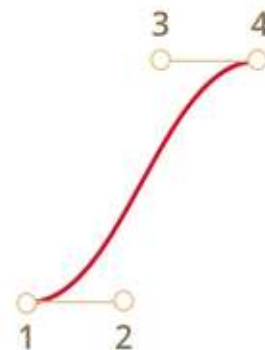
ease-out

(0, 0, 0.58, 1.0)



ease-in-out

(0.42, 0, 0.58, 1.0)



Шаги steps

Временная функция

steps(количество шагов[, start/end])

позволяет разбить анимацию на чёткое количество шагов.

```
1 #stripe.animate {  
2   margin-left: -174px;  
3   transition: margin-left 9s steps(9, start);  
4 }
```

Событие **transitionend**

- На конец CSS-анимации можно повесить обработчик на событие **transitionend**.
- Это широко используется, чтобы после анимации сделать какое-то действие или объединить несколько анимаций в одну.

Свойства объекта transitionend

propertyName

Свойство, анимация которого завершилась.

elapsedTime

Время (в секундах), которое заняла анимация, без учета transition-delay.


CSS animations

- Более сложные анимации делаются объединением простых при помощи CSS-правила **@keyframes**.
- <http://css.yoksel.ru/css-animation/>
- <https://albiebrown.github.io/gravify/>

```
1 <div class="progress"></div>
2
3 <style>
4   /* Современные браузеры, кроме Chrome, Opera, Safari */
5   @keyframes go-left-right { /* назовём анимацию: "go-left-right" */
6     from {
7       left: 0px; /* от: left: 0px */
8     }
9     to {
10      left: calc(100% - 50px); /* до: left: 100%-50px */
11    }
12  }
13
14  /* Префикс для Chrome, Opera, Safari */
15  @-webkit-keyframes go-left-right {
16    from {
17      left: 0px;
18    }
19    to {
20      left: calc(100% - 50px);
21    }
22  }
23
24  .progress {
25    /* применить анимацию go-left-right */
26    /* продолжительность 3s */
27    /* количество раз: бесконечное (infinite) */
28    /* менять направление анимации каждый раз (alternate) */
29    animation: go-left-right 3s infinite alternate;
30    -webkit-animation: go-left-right 3s infinite alternate;
31
32    position: relative;
33    border: 2px solid green;
34    width: 50px;
35    height: 20px;
36    background: lime;
37  }
```


Canvas

Canvas(холст) –прямоугольная область для рисования растровой графики с помощью JavaScript.

Canvas (basic support)  - LS									
Global									
95.02% + 2.87% = 97.89%									
Method of generating fast, dynamic graphics using JavaScript.									
Current aligned Usage relative Date relative Show all									
IE	Edge *	Firefox	Chrome	Safari	iOS Safari *	Opera Mini *	Chrome for Android	UC Browser for Android	Samsung Internet
			49						
			61		10.2				4
	15	56	62	10.1	10.3				5
11	16	57	63	11	11.2	² all	62	11.4	6.2
	17	58	64	TP					
		59	65						
		60	66						

Начало системы координат



Получение контекста

```
var canvas = document.getElementById( 'canvas' );  
var context = canvas.getContext( '2d' );
```

Работа с путями

beginPath() – сигнализирует о начале создания новой фигуры

moveTo() – перемещение в новую точку без прорисовки линии

lineTo() – перемещение текущей позиции в новую точку с прорисовкой линии соединяющую обе точки

closePath() – замыкает фигуру соединяя последнюю точку с начальной.

stroke() – прорисовка контуров созданной фигуры.

fill() – заполнение созданной фигуры.

Заливка прямоугольника

`fillRect(x, y, ширина, высота);`

`fillRect()` – заливка прямоугольной области сплошным цветом.

`strokeRect()` – вычерчивание прямоугольника по указанным параметрам.

`clearRect()` – очистка прямоугольной области холста от любого содержимого.

SVG

Scalable Vector Graphics

```
<!-- Определение элемента с графикой -->
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">

    <!-- Определение примитивов -->
    <rect width="300" height="100"
        style="fill: rgb(0,0,255);
        stroke-width: 1;
        stroke: rgb(0,0,0)" />

</svg>
```


Примитивы для построение графики

- rect
- circle
- Ellipse
- line

JS-Анимация

JS timers

<http://ejohn.org/blog/how-javascript-timers-work>

- `var id = setTimeout(fn, delay);`
- `var id = setInterval(fn, delay);`
- `clearInterval(id);, clearTimeout(id);`

setInterval()

Alert "Hello" every 3 seconds:

```
setInterval(function(){ alert("Hello"); }, 3000);
```

Return Value: A Number, representing the ID value of the timer that is set. Use this value with the `clearInterval()` method to cancel the timer

setInterval()

```
<!DOCTYPE html>
<html>
<body>

<p>A script on this page starts this clock:</p>
<p id="demo"></p>

<script>
var myVar = setInterval(function(){ myTimer() }, 1000);

function myTimer() {
    var d = new Date();
    var t = d.toLocaleTimeString();
    document.getElementById("demo").innerHTML = t;
}
</script>

</body>
</html>
```

A script on this page starts this clock:

23:04:54

clearInterval

```
<!DOCTYPE html>
<html>
<body>

<p>A script on this page starts this clock:</p>
<p id="demo"></p>

<button onclick="myStopFunction()">Stop time</button>

<script>
var myVar = setInterval(function(){ myTimer() }, 1000);

function myTimer() {
    var d = new Date();
    var t = d.toLocaleTimeString();
    document.getElementById("demo").innerHTML = t;
}

function myStopFunction() {
    clearInterval(myVar);
}
</script>

</body>
</html>
```

A script on this page starts this clock:

23:07:08

Stop time

Using setInterval() and clearInterval() to create a dynamic progress bar:

```
function move() {  
    var elem = document.getElementById("myBar");  
    var width = 0;  
    var id = setInterval(frame, 10);  
    function frame() {  
        if (width == 100) {  
            clearInterval(id);  
        } else {  
            width++;  
            elem.style.width = width + '%';  
        }  
    }  
}
```

Toggle between two background colors once every 300 milliseconds:

```
var myVar = setInterval(function(){ setColor() }, 300);

function setColor() {
    var x = document.body;
    x.style.backgroundColor = x.style.backgroundColor == "yellow" ? "pink" : "yellow";
}

function stopColor() {
    clearInterval(myVar);
}
```


setTimeout()

Display an alert box after 3 seconds:

```
setTimeout(function(){ alert("Hello"); }, 3000);
```

Return Value: A Number, representing the ID value of the timer that is set. Use this value with the `clearTimeout()` method to cancel the timer

Display a timed text:

```
<!DOCTYPE html>
<html>
<body>

<p>Click on the button below. The input field will tell you
when two, four, and six seconds have passed.</p>

<button onclick="timedText()">Display timed text</button>
<input type="text" id="txt">

<script>
function timedText() {
    var x = document.getElementById("txt");
    setTimeout(function(){ x.value="2 seconds" }, 2000);
    setTimeout(function(){ x.value="4 seconds" }, 4000);
    setTimeout(function(){ x.value="6 seconds" }, 6000);
}
</script>

</body>
</html>
```

Click on the button below. The input field will tell you when two, four, and six seconds have passed.

Display timed text

Open a new window and close the window after three seconds:

```
var myWindow = window.open("", "", "width=200, height=100");  
myWindow.document.write("<p>This is 'myWindow'</p>");  
setTimeout(function(){ myWindow.close() }, 3000);
```

clearTimeout()

При наведении мышкой на элемент необходимо через две секунды показать сообщение. Однако если в течение этих двух секунд указатель мыши был убран с элемента, то сообщение показывать не нужно.

```
var timerId;
element.onmouseover = function() {
    timerId = setTimeout(function() {
        alert(1);
    }, 2000);
};
element.onmouseout = function() {
    clearTimeout(timerId);
};
```

setTimeout, setInterval

```
// начать повторы с интервалом 2 сек  
var timerId = setInterval(function() {  
    alert( "тик" );  
}, 2000);
```

```
// через 5 сек остановить повторы  
setTimeout(function() {  
    clearInterval(timerId);  
    alert( 'стоп' );  
}, 5000);
```

Рекурсивный `setTimeout`

Рекурсивный `setTimeout` – более гибкий метод тайминга, чем `setInterval`, так как время до следующего выполнения можно запланировать по-разному, в зависимости от результатов текущего.

следующее выполнение планируется сразу после окончания предыдущего.

```
1  /** ВМЕСТО:
2  var timerId = setInterval(function() {
3      alert( "ТИК" );
4  }, 2000);
5  */
6
7  var timerId = setTimeout(function tick() {
8      alert( "ТИК" );
9      timerId = setTimeout(tick, 2000);
10 }, 2000);
```

- **Рекурсивный `setTimeout` гарантирует паузу между вызовами, `setInterval` – нет.**

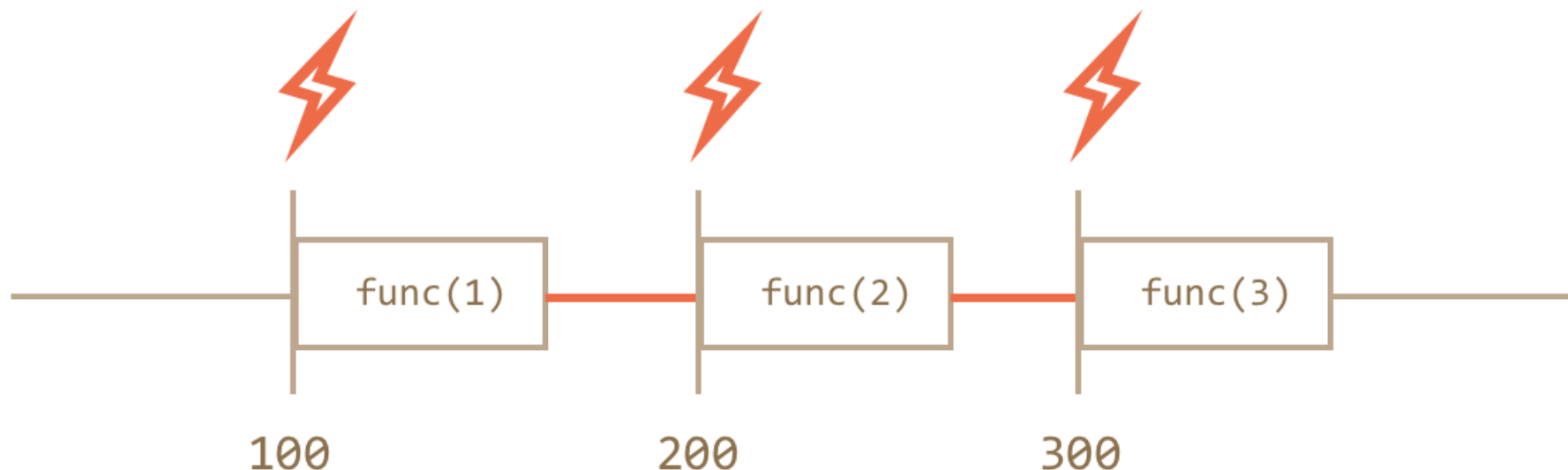
Давайте сравним два кода. Первый использует `setInterval`:

```
1 var i = 1;
2 setInterval(function() {
3     func(i);
4 }, 100);
```

Второй использует рекурсивный `setTimeout`:

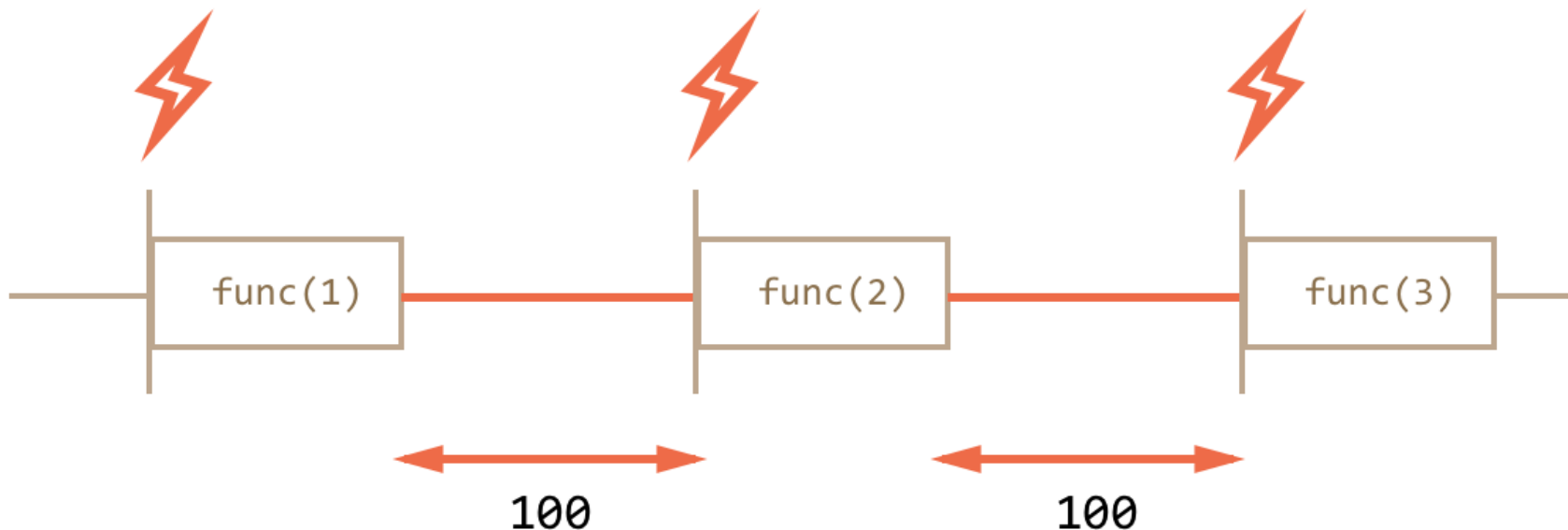
```
1 var i = 1;
2 setTimeout(function run() {
3     func(i);
4     setTimeout(run, 100);
5 }, 100);
```

- **Реальная пауза между вызовами func при `setInterval` меньше, чем указана в коде!**



- **Если функция выполняется дольше, чем пауза `setInterval`, то вызовы будут происходить вообще без перерыва.**

Кроме IE



- При рекурсивном `setTimeout` задержка всегда фиксирована и равна 100 мс.

Что такое коллбэк?

- любая функция, которая передается как аргумент, называется **callback-функцией**
- коллбэк – это функция, которая должна быть выполнена после того, как другая функция завершила выполнение

```
function first(){  
    console.log(1);  
}  
function second(){  
    console.log(2);  
}  
first();    // 1  
second();  // 2
```

```
function first(){  
    // Как будто бы запрос к API  
    setTimeout( function(){  
        console.log(1);  
    }, 500 );  
}  
function second(){  
    console.log(2);  
}
```

```
first();  
second();  
// 2  
// 1
```

```
function doHomework(subject, callback) {  
  alert(`Starting my ${subject} homework.`);  
  callback();  
}  
  
doHomework('math', function() {  
  alert('Finished my homework');  
});
```

```
function doHomework(subject, callback) {  
  alert(`Starting my ${subject} homework.`);  
  callback();  
}  
  
function alertFinished(){  
  alert('Finished my homework');  
}  
  
doHomework('math', alertFinished);
```

```
T.get('search/tweets', params, function(err, data, response) {  
  if(!err){  
    // Происходит какая-то магия  
  } else {  
    console.log(err);  
  }  
})
```


Чем плох setInterval

Разность момента установки таймера для следующего запуска.

Тест: будем отмерять количество времени, прошедшее с начала выполнения предыдущего запуска и с его окончания.

```
var d1 = new Date(), d2 = new Date();
setInterval(function() {
    var d = new Date();
    document.body.innerHTML += (d - d1) + ' ' + (d - d2) + '<br>';

    // Ставим метку в начале функции
    d1 = new Date();
    while (new Date() - d1 < 200); // ничего не делаем 200 миллисекунд
    // И в конце функции
    d2 = new Date();
}, 1000);
```

Chrome и Opera

1000 800

1003 803

1004 804

1018 808

987 787

1003 803

1000 800

Safari

1035 835

1259 1059

841 641

1091 891

1048 848

1005 805

998 798

Firefox

988 788

985 785

1007 807

971 771

984 784

990 790

974 774

1010 810

1004 804

IE

1203 1000

1203 1000

1203 1000

1203 1000

1204 1000

1203 1000

1203 1000

Чем заменить setInterval

Создаём безымянную функцию, тут же её вызываем, и в последующем передаём её же функции setTimeout.

```
(function() {  
    // Выполняем периодические действия  
  
    setTimeout(arguments.callee, 500);  
})();
```

Типичная асинхронная функция,
выполняющая какие-то периодические
действия:

```
/**
 * @param {Function} callback Функция, вызываемая по окончании работы.
 */
function foo(callback) {
    // Инициализируем переменные

    (function() {
        // Выполняем действия

        if (/* больше ничего делать не надо */) {
            callback();
        } else {
            setTimeout(arguments.callee, 500);
        }
    })();
}
```

Функция, посимвольно выводящая строку в нужный DOM-элемент, и вызывающая callback-функцию по окончании работы

```
function typeString(elId, str, callback) {  
    var i = 1, el = document.getElementById(elId);  
    (function() {  
        el.innerHTML = str.substr(0, i++);  
        if (i > str.length) {  
            callback();  
        } else {  
            setTimeout(arguments.callee, 300);  
        }  
    }) ();  
}
```

```
typeString('t', 'Hello, World!', function() {  
    alert('Напечатали');  
});
```

Очень быстрый setTimeout

- setZeroTimeout

```
var i = 0, d = new Date(), N = 1000;
(function() {
    if (i < N) {
        i++;
        setZeroTimeout(arguments.callee);
    } else {
        alert((new Date() - d) / N);
    }
})();
```

Результаты будут примерно следующие: в Safari 2 мс, в Opera 0.04 мс (т.е. доли миллисекунды). В остальных браузерах результаты разнятся в зависимости от N: чем больше N, тем меньше среднее время. В Firefox — от 10 до 0.07 мс, в Chrome — от 1 до 0.02 мс.

Управление памятью

- Сборщик мусора в JavaScript не чистит функции, назначенные в таймерах, пока таймеры актуальны.

```
1 // Функция будет жить в памяти, пока не сработал (или не был очищен) таймер
2 setTimeout(function() {}, 100);
```

- Для `setTimeout` – внутренняя ссылка исчезнет после исполнения функции.
- Для `setInterval` – ссылка исчезнет при очистке таймера.