# DOM Events, часть 1

JS COURSE ORT DNIPRO

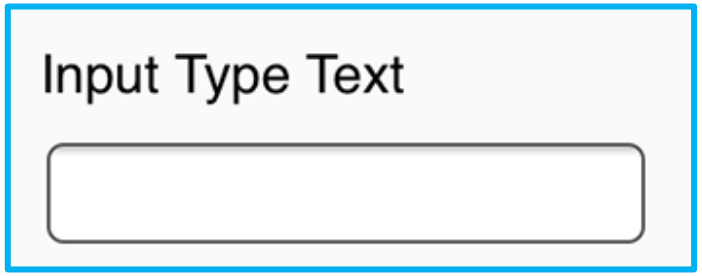
ORTDNIPRO.ORG/JS

# 1. Событийная модель

# Событийно-ориентированная система управления

Каждая из этих вещей делает что-то в ответ на действия пользователя. Можно сказать каждое действие пользователя это **событие**, и на него нужно как-то **отреагировать**.

# События возможные для одних элементов, могут не существовать для других




Поддерживает **ввод с клавиатуры**, события «**фокус**» и «**потеря фокуса**».


Не поддерживает **ввод с клавиатуры**, и событий «**фокус**» и «**потеря фокуса**» для него тоже быть не может.

Однако есть набор событий который поддерживают все элементы: **клик**, **наведение курсора мыши** и т.д.

# 2. Подписка на события

# Как указать браузеру какую функцию и когда вызывать?

```
2
3   <h1 onclick="eventListener()">Some Content</h1>
4
5   <script>
6
7
8     function eventListener(){
9       console.log('Click detected!');
10    }
11
12  </script>
13
```

*Через соответствующие атрибуты тегов*

# Как указать браузеру какую функцию и когда вызывать?

```
2
3   <h1>Some Content</h1>
4
5   <script>
6
7       let h1Tag = document.querySelector('h1');
8
9       h1Tag.onclick = function(){
10          console.log('Click detected!');
11      }
12
13  </script>
14
```

*Через свойства объектов входящих в дерево документа*

# Как указать браузеру какую функцию и когда вызывать?

```
2
3   <h1>Some Content</h1>
4
5   <script>
6
7       let h1Tag = document.querySelector('h1');
8
9       function eventListener_1(){
10          console.log("I'm eventListener_1");
11      }
12
13      function eventListener_2(){
14          console.log("I'm eventListener_2");
15      }
16
17      h1Tag.addEventListener('click', eventListener_1);
18      h1Tag.addEventListener('click', eventListener_2);
19
20      //h1Tag.removeEventListener('click', eventListener_1);
21      //h1Tag.removeEventListener('click', eventListener_2);
22
23  </script>
24
```

При помощи метода **.addEventListener()** можно на одно событие повесить множество обработчиков. А при необходимости и снять обработчик при помощи **.removeEventListener()**.

# Вспоминаем this

```
2
3   <h1>Some Content</h1>
4   <p>Some P tag</p>
5
6   <script>
7
8       let h1Tag = document.querySelector('h1');
9       let pTag = document.querySelector('p');
10
11      function eventListener(){
12          console.log('this in eventListener:', this);
13      }
14
15      h1Tag.addEventListener('click', eventListener);
16      pTag.addEventListener('click', eventListener);
17
18
19  </script>
20
```

Функция обработчик становиться частью объекта-элемента, и вызывается как его метод. Поэтому ключевое слово **this** в обработчике ссылается на объект который вызвал обработчик события.

# 3. События

onLoad,

onDOMContentLoaded

# Событие window.onload

```
3
4   window.addEventListener('load', function(e){
5       console.log("Event Window.onLoad", e);
6   });
7
8   document.addEventListener('DOMContentLoaded', function(e){
9       console.log("Event Document.DOMContentLoaded", e);
10  });
11
```

Событие **onload** (объекта **window**, он же **globalThis**) срабатывает тогда когда загружен (и обработан) HTML документ и все подключаемые файлы, в т.ч изображения, стили т.д.

# Событие document.DOMContentLoaded

```
3
4   window.addEventListener('load', function(e){
5       console.log("Event Window.onLoad", e);
6   });
7
8   document.addEventListener('DOMContentLoaded', function(e){
9       console.log("Event Document.DOMContentLoaded", e);
10  });
11
```

Событие **DOMContentLoaded** доступно для объекта **document** через **.addEventListener()** и срабатывает тогда когда загружен HTML документ и JS файлы (завершилась ли загрузка изображений и css-файлов неважно).

# 3. Информация о событии

# Информация о событии

*Чтобы обработать событие, недостаточно знать о том, что это – «клик» или «нажатие клавиши». Могут понадобиться детали: координаты курсора, введённый символ и другие, в зависимости от события.*

*Браузер может дать много полезной информации о событии, для этого он создаёт объект, в свойства которого записывает детали произошедшего события. И передаёт этот объект функции обработчику события.*

# Информация о событии

```
2
3   <h1>Some Content</h1>
4
5   <script>
6
7       let h1Tag = document.querySelector('h1');
8
9       function eventListener(e){
10          console.log('Event info:', e);
11      }
12
13      h1Tag.addEventListener('click', eventListener);
14
15  </script>
16
```

Браузер записывает информацию о событии в объект т.н. «объект события», который передаётся первым аргументом в функцию обработчик события. Если она принимает параметры, т.к. это является необязательным.

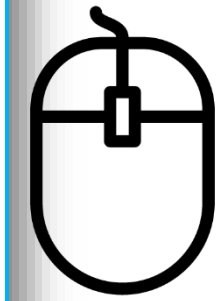
# Информация о событии

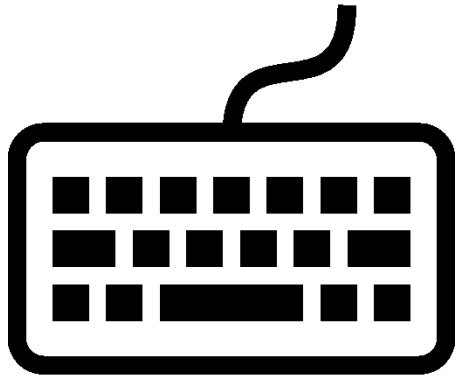
*Разные события – разные объекты с информацией о них.*

*В зависимости от типа события, объект с детальной информацией о событии содержит разные наборы полей, например: для событий мыши он содержит координаты курсора, а события клавиатуры он содержит данные о нажатых клавишах.*

# Информация о событии

*Разные события – разные объекты с информацией о них.*

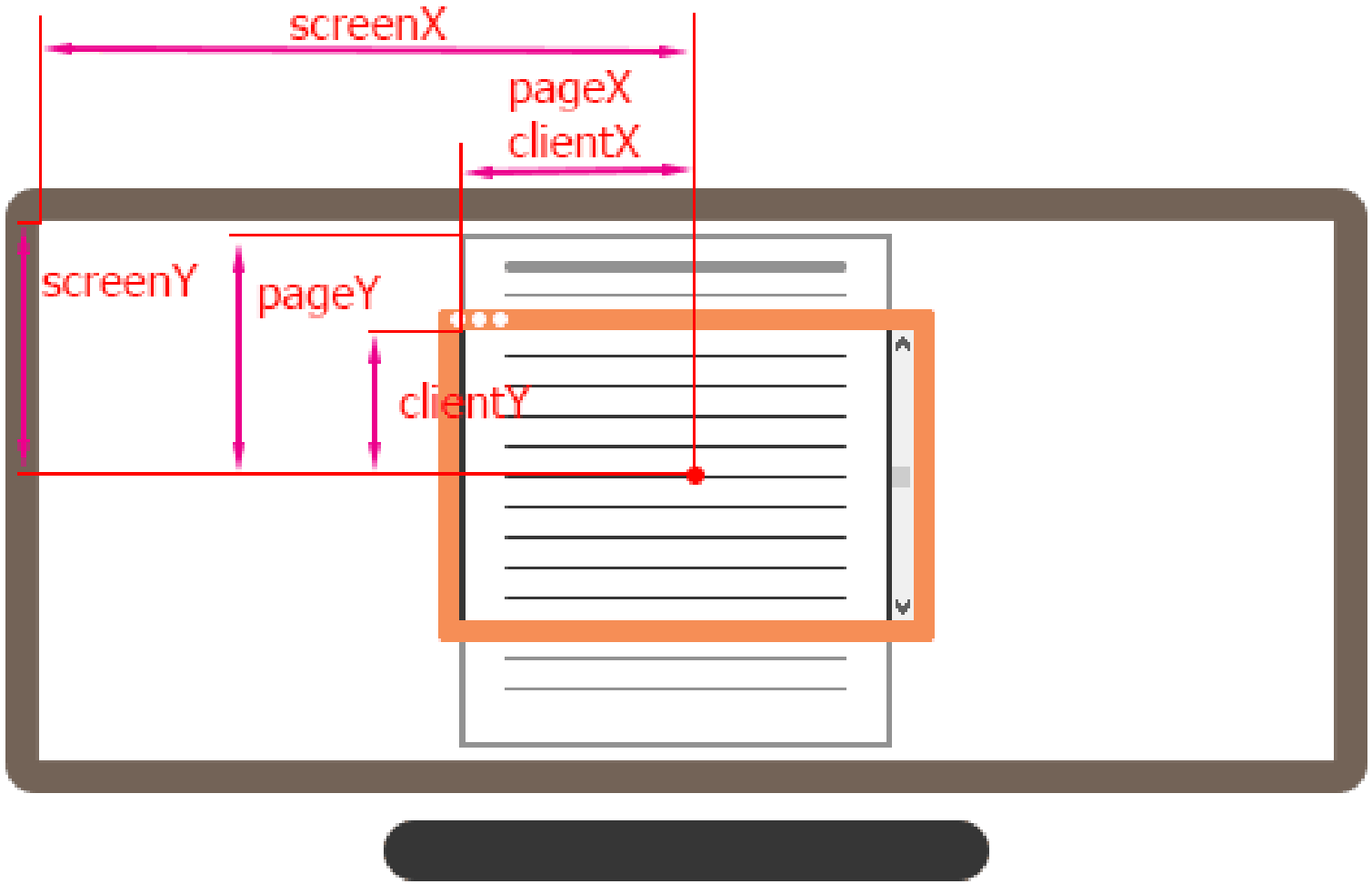


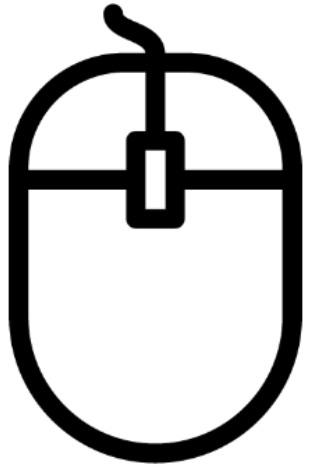


```
altKey: false
bubbles: true
button: 0
buttons: 0
cancelBubble: false
cancelable: true
clientX: 83
clientY: 17
ctrlKey: false
currentTarget: null
defaultPrevented: false
detail: 1
eventPhase: 0
fromElement: null
isTrusted: true
isTrusted: true
layerX: 83
layerY: 17
metaKey: false
movementX: 0
movementY: 0
offsetX: 75
offsetY: 9
pageX: 83
pageY: 17
▶ path: Array[5]
relatedTarget: null
returnValue: true
screenX: 2003
screenY: 102
shiftKey: false
▶ sourceCapabilities: InputDeviceCapabilities
▶ srcElement: p
▶ target: p
timeStamp: 1314.7900000000002
▶ toElement: p
type: "click"
▶ view: Window
which: 1
x: 83
y: 17
```

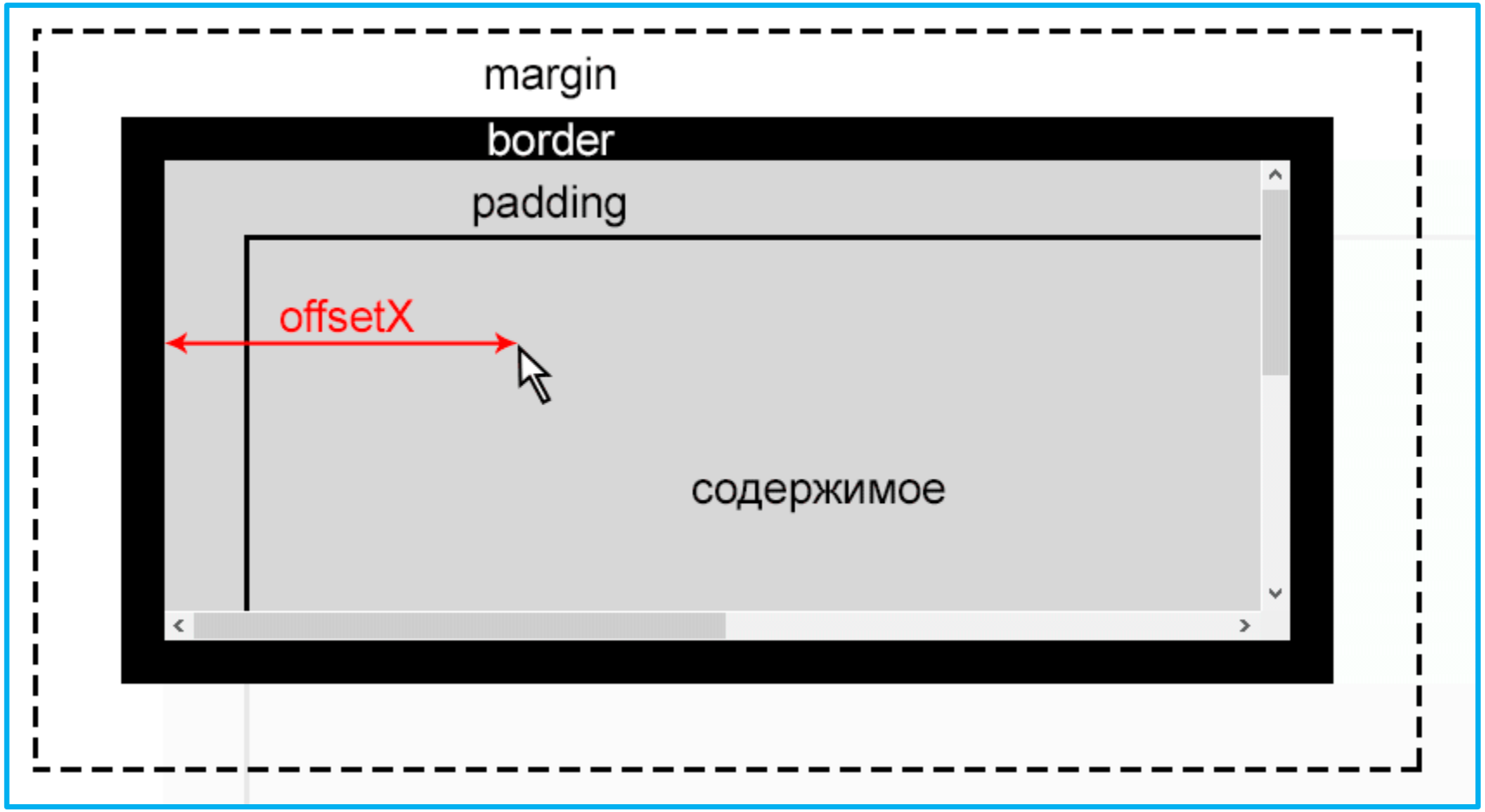
```
altKey: false
bubbles: true
cancelBubble: false
cancelable: true
charCode: 97
code: "KeyA"
ctrlKey: false
currentTarget: null
defaultPrevented: false
detail: 0
eventPhase: 0
isTrusted: true
isTrusted: true
keyCode: 97
keyIdentifier: "U+0041"
keyLocation: 0
location: 0
metaKey: false
▶ path: Array[5]
repeat: false
returnValue: true
shiftKey: false
▶ sourceCapabilities: InputDeviceCapabilities
▶ srcElement: input
▶ target: input
timeStamp: 2079.225
type: "keypress"
▶ view: Window
which: 97
```

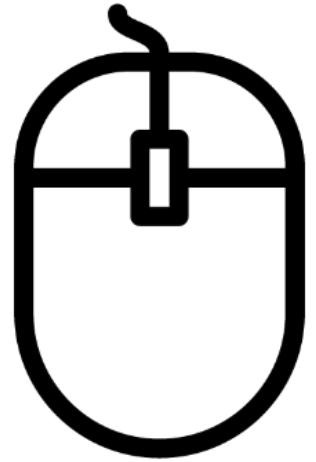
# Информация о позиции курсора (пальца)



# Информация о позиции курсора (пальца)



# 4. Немного практики

# Рисование, Графика, Canvas

```
<canvas id="paint-canvas"></canvas>
<script>

    var canvas      = document.getElementById("paint-canvas");
    canvas.width    = canvas.clientWidth;
    canvas.height   = canvas.clientHeight;

    var context     = canvas.getContext("2d");

    context.moveTo(200, 200);
    context.lineTo(300, 250);
    context.lineTo(200, 300);
    context.closePath();
    context.stroke();

</script>
```

Тег **canvas** – представляет собой «**холст**», прямоугольную область в которой можно рисовать. **Контекст canvas'а** – объект который содержит множество методов для рисования на «**холсте**».

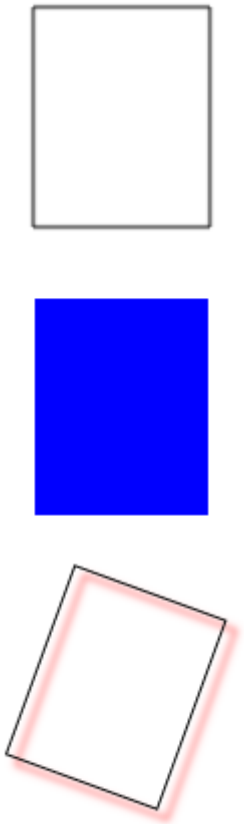
# Рисование, Графика, Canvas

Рисование на **canvas’**е основано на отрисовке примитивов.

1) Штриховых (контурных фигур) – в названии методов и свойств есть слово **stroke**;

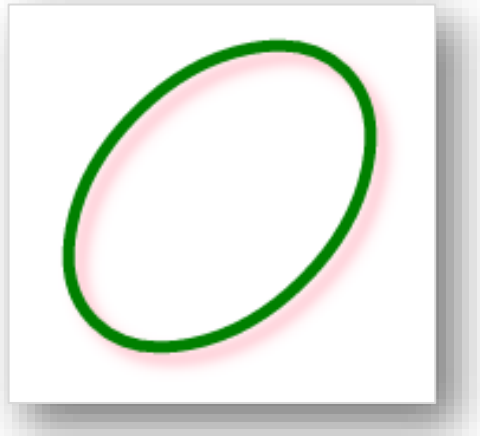
2) Заполненных фигур, в названии методов и свойств есть слово **fill**;

3) Наложении спецэффектов (тени, развороты, искажения и т.п.).

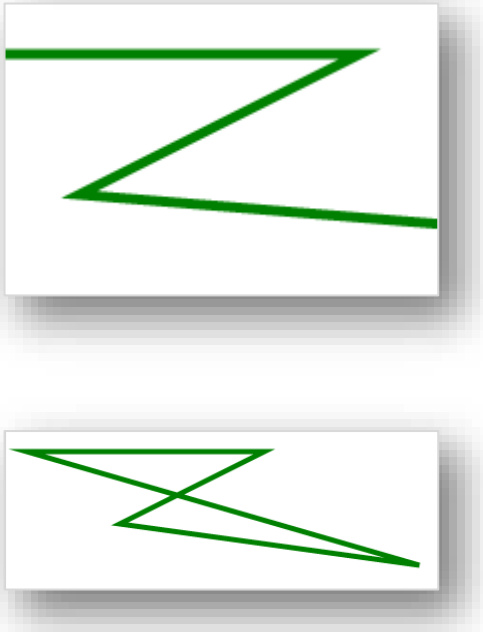


# Рисование, Графика, Canvas

Примитивы можно рисовать при помощи функций-заготовок: прямоугольник (**rect()**), эллипс (**ellipse()**) и т.п.



Либо самостоятельно задав контур фигуры состоящей из множества линий. Для этого есть функции **beginPath()** и **closePath()** – для случаев когда нужно замкнуть контур (между первой и последней точкой фигуры).

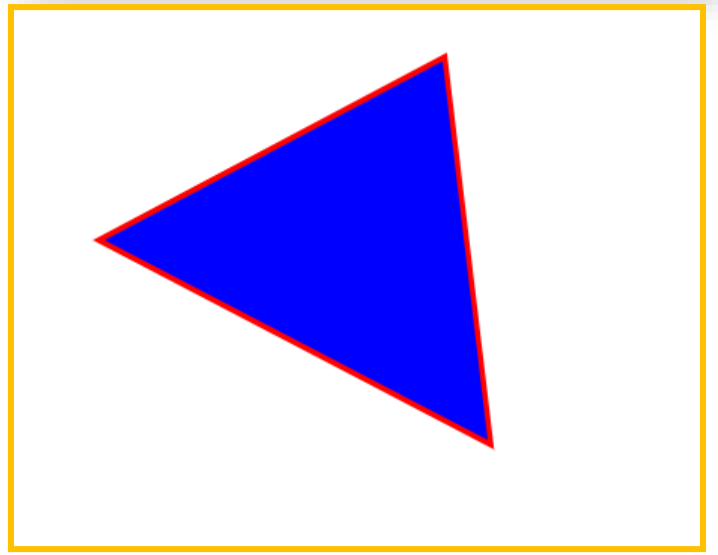


# Рисование примитивов

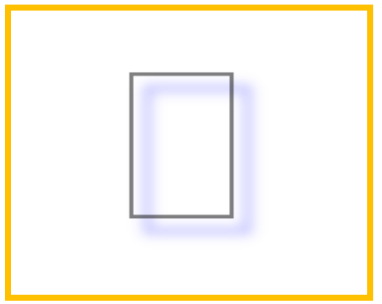
```
context.beginPath();
context.moveTo(175, 225);
context.lineTo(400,113);
context.lineTo(430, 350);
//context.closePath();
context.stroke();
```

```
context.beginPath();
context.moveTo(175, 225);
context.lineTo(400,113);
context.lineTo(430, 350);
//context.closePath();
context.fill();
```

```
context.beginPath();
context.moveTo(175, 225);
context.lineTo(400,113);
context.lineTo(430, 350);
context.closePath();
context.lineWidth = 7;
context.strokeStyle = "red";
context.fillStyle = "blue";
context.stroke();
context.fill();
```



```
context.rect(300, 200, 50, 80);
context.shadowBlur = 10;
context.shadowOffsetY = 8;
context.shadowOffsetX = 8;
context.shadowColor = "blue";
context.stroke();
```



# Свойства (графические атрибуты «холста»)

## Paths

| Method | Description |
|---|---|
| fill() | Fills the current drawing (path) |
| stroke() | Actually draws the path you have defined |
| beginPath() | Begins a path, or resets the current path |
| moveTo() | Moves the path to the specified point in the canvas, without creating a line |
| closePath() | Creates a path from the current point back to the starting point |
| lineTo() | Adds a new point and creates a line to that point from the last specified point in the canvas |
| clip() | Clips a region of any shape and size from the original canvas |
| quadraticCurveTo() | Creates a quadratic Bézier curve |
| bezierCurveTo() | Creates a cubic Bézier curve |
| arc() | Creates an arc/curve (used to create circles, or parts of circles) |
| arcTo() | Creates an arc/curve between two tangents |
| isPointInPath() | Returns true if the specified point is in the current path, otherwise false |

## Transformations

| Method | Description |
|---|---|
| scale() | Scales the current drawing bigger or smaller |
| rotate() | Rotates the current drawing |
| translate() | Remaps the (0,0) position on the canvas |
| transform() | Replaces the current transformation matrix for the drawing |

**Подробнее: http://www.w3schools.com/tags/ref_canvas.asp**

# 5. «Paint» на JavaScript

# «Paint» на JavaScript

Простой аналог программы «**Paint**» на базе **JavaScript** и **canvas**.

Воспользуйтесь шаблоном в репозитории занятия:
**./src/paint-example**

# На следующем занятии

# На следующем занятии

## Обработка событий (DOM Events), часть 2