

# Анимация в JavaScript

---

- События и узлы DOM
- Элемент Canvas
- Управление анимацией

# События и узлы

---

## DOM

У каждого элемента DOM есть свой метод **addEventListener**, позволяющий прослушивать события, происходящие с НИМ.

**Назначение обработчика на DOM кнопки:**

```
<button id = "b1">Кликни!</button>
```

```
<script>
```

```
var btn = document.getElementById("b1");
```

```
btn.addEventListener("click", function() {
```

```
    //если был клик – выводим сообщение
```

```
    alert("Вы нажали на кнопку");
```

```
});
```

```
</script>
```

# События и узлы

---

## DOM

`addEventListener` позволяет добавлять любое количество обработчиков.

Метод `removeEventListener`, вызванный с такими же аргументами, как `addEventListener`, удаляет обработчик.

```
<button>Act-once button</button>
<script>
  var btn = document.querySelector("button");
  function once() {
    alert("Done");
    btn.removeEventListener("click", once);
  }
  btn.addEventListener("click", once);
</script>
```

# Объекты событий

---

В объектах событий хранится дополнительная информация о событии.

```
<button>Нажатие клавиши</button>
```

```
<script>
```

```
var btn = document.querySelector("button");
```

```
btn.addEventListener("mousedown", function(event) {  
  if (event.which == 1) alert("Левая кнопка");  
  else if (event.which == 2) alert("Средняя кнопка");  
  else if (event.which == 3) alert("Правая кнопка");  
});
```

```
</script>
```

# Действия по

---

## умолчанию

- При клике на ссылку – переход по ней.
- При нажатии на стрелку вниз – прокрутка страницы вниз.
- По правому клику мыши – вывод контекстного меню.
- И т.п.

Обработчики событий вызываются до того, как сработает действие по умолчанию.

Если нужно отменить действие по умолчанию можно вызвать метод **preventDefault** объекта события.

```
var link = document.querySelector("a");
link.addEventListener("click", function(event) {
  console.log("Переход по ссылке отменен!");
  event.preventDefault();
});
```

# События кнопок

---

## клавиатуры

`<p>Нажмите на клавишу V </p>`

`<script>`

```
addEventListener("keydown", function(event) {  
  if (event.keyCode == 86)  
    document.body.style.background = "blue"; //голубой фон  
});
```

```
addEventListener("keyup", function(event) {  
  if (event.keyCode == 86)  
    //а теперь фиолетовый  
    document.body.style.background = "violet";  
});
```

`</script>`

# События кнопок

---

**клавиатуры**  
**// Распознаем коды кнопок клавиатуры:**

```
<script>  
console.log("3".charCodeAt(0)); // → 51  
console.log("qwerty".charCodeAt(0)); // → 113  
console.log("Привет".charCodeAt(0)); // → 1055  
</script>
```

**// какие символы были напечатаны**

**<p>Переведите фокус на страницу и печатайте.</p>**

```
<script>  
addEventListener("keypress", function(event) {  
  console.log(String.fromCharCode(event.charCode));  
});  
</script>
```

# Кнопки

---

## МЫШИ

Получение точных координат места, где произошло событие мыши – свойства **pageX** и **pageY** – они содержат координаты в пикселях относительно верхнего левого угла.

```
.dot {  
  height: 8px; width: 8px;  
  background: blue;  
  position: absolute; }
```

```
addEventListener("click", function(event) {  
  var div = document.createElement("div");  
  div.className = "dot";  
  div.style.left = (event.pageX - 4) + "px";  
  div.style.top = (event.pageY - 4) + "px";  
  document.body.appendChild(div);  
});
```

# Движение

---

## МЫШИ

При движении курсора мыши запускается событие «mousemove». Его можно использовать для отслеживания позиции мыши.

```
p:hover { color: red }
```

```
<p>Наведите мышь на этот <b>параграф</b> </p>
```

События:

**mousemove** – сдвиг курсора мыши

**mouseover** или **mouseout** – уход с фокуса

**scroll** – прокручивание элемента

**focus** – получение фокуса элемента

**blur** – потеря фокуса

# Установка

## таймеров

---

Для отрисовки обновлений могут быть использованы методы для вызова некоторой функции, через заданный промежуток времени.

**setInterval(function, delay)** – начинает периодически исполнять функцию каждые *delay* миллисекунд.

**setTimeout(function, delay)** – запускает выполнение указанной функции через *delay* миллисекунд.

**requestAnimationFrame(callback)** – сообщает браузеру, что вы хотите выполнить анимацию, и запрашивает, чтобы браузер вызвал указанную функцию *callback* для обновления анимации перед следующей

# Установка таймеров

---

Функция **setTimeout** схожа с **requestAnimationFrame**. Она планирует запуск функции в будущем. Первый аргумент – что произойдет, второй – через какое количество секунд.

//через две секунды фон страницы станет жёлтым

```
setTimeout(function() {  
    document.body.style.background = "yellow";  
}, 2000);
```

# Установка

---

## таймеров

**setInterval** и **clearInterval** используются для установки таймеров, которые будут повторяться через каждое указанное количество миллисекунд.

```
var ticks = 0;
```

```
var clock = setInterval(function() {  
    console.log("tick", ticks++);  
    if (ticks == 10) {  
        clearInterval(clock);  
        console.log("stop.");  
    }  
}, 200);
```

# Элемент холста

---

## Canvas

**Canvas** (холст) – это элемент из стандарта HTML5, который даёт разработчикам возможность создания динамической графики при помощи **JavaScript**.

**context** – объект, чьи методы предоставляют интерфейс для рисования графики.

Два стиля рисования:

- **2d** для двумерной графики
- **webgl** для трёхмерной графики при помощи интерфейса OpenGL.

# Элемент холста

---

## Canvas

### HTML:

```
<p>Before canvas</p>  
<canvas width="120" height="60">  
</canvas>  
<p>After canvas.</p>
```

Before canvas



After canvas.

### JavaScript:

```
var canv = document.querySelector("canvas");  
var context = canv.getContext("2d");  
  
context.fillStyle = "green";  
context.fillRect(10, 10, 100, 50);
```

# Элемент холста

---

## Canvas

Рисуем закрасенный треугольник:

```
<canvas></canvas>
```

```
var ct = document.querySelector("canvas").getContext("2d");  
ct.beginPath();  
ct.moveTo(50, 10);  
ct.lineTo(10, 70);  
ct.lineTo(90, 70);  
ct.fill();  
ct.closePath();
```

# Элемент холста

---

## Canvas

Метод **drawImage** позволяет выводить на холст пиксельные данные.

```
var cx = document.querySelector("canvas").getContext("2d");  
var img = document.createElement("img");
```

```
img.src = "img/hat.png";  
img.addEventListener("load", function() {  
  for (var x = 10; x < 200; x += 30)  
    cx.drawImage(img, x, 10);  
});
```

# Этапы рисования

---

## кадра

1. **Очистить canvas** – всё что было нарисовано ранее необходимо стереть. Например, при помощи метода `clearRect()`.
2. **Сохранить начальное состояние canvas** – если меняете настройки (стили, трансформации и т.п.), которые затрагивают состояние canvas – следует сохранить это оригинальное состояние.
3. **Нарисовать анимированные фигуры** – шаг на котором рисуете кадр.
4. **Восстановить состояние canvas** – если вы сохраняли состояние, восстановите его, прежде чем рисовать новый.

# Управление

## анимацией

---

```
var width = 100, height = 100, frames = 4, currentFrame = 0;
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");
var image = new Image();
image.src = 'sprite.png';

function draw() {
    ctx.clearRect(0, 0, width, height);
    ctx.drawImage(image, 0, height * currentFrame, width,
height, 0, 0, width, height);
    if (currentFrame == frames) currentFrame = 0;
    else currentFrame++;
}
setInterval(draw, 100);
```

# Лабораторная работа

---

11

Необходимо разработать игру «Кошки-мышки».

## **Описание:**

На игровом поле бегают мышки, у каждой своя скорость и траектория движения. Игрок должен кликать на мышек, после чего та исчезает, а игрок зарабатывает баллы.

На игровом поле могут находиться не менее 10 мышек. Мышки должны быть трех разных размеров. Клик на маленькой мышке – 30 баллов, на средней – 20, на большой – 10.

Игровое поле должно содержать блок с заработанными баллами.

Дизайн игры должен соответствовать ее логике.

# Передача данных в html- файл

---

Передать данные в html-файл можно с помощью URL-адреса.

После адреса ставиться знак вопроса (?), затем все аргументы, разделенные амперсандом (&).

```
"sample.html?value1&value2&value3"
```

Либо с помощью метода GET:

```
"sample.html?parametr1=value1&parametr2=value2"
```

# Чтение переданных

## данных

Сохраняем в переменную то, что в url-адресе после знака вопроса:

```
var url_args = location.search.substring(1);
```

Если параметров несколько, разделяем их:

```
var args = url_args.split("&"); // получаем массив
```

Если передавали с помощью GET-запроса, то разделяем '=':

```
var url_args = location.search.substring(1);
```

```
var values = new Array();
```

```
for (i in url_args) {
```

```
    var j = arg[i].split("=");
```

```
    values[j[0]]=unescape(j[1]);
```

```
}
```