Практическая работа №6 «Анимация в CSS. Ключевые кадры @keyframes, свойства CSS-анимации, свойства transition и transform, группы функций трансформации. Свойство box-shadow»

11 Анимация в CSS

Первые анимации реализовывались при помощи Flash и JavaScript. Позже многие инструменты были внедрены в CSS.

CSS-анимации могут проигрываться без дополнительных действий со стороны пользователя и состоять из нескольких шагов.

Список свойств для создания CSS-анимаций:

- `animation-name`
- `animation-duration`
- `animation-iteration-count`
- `animation-direction`
- `animation-timing-function`
- `animation-delay`
- `animation-play-state`
- `animation-fill-mode`
- `animation`

Для создания *ключевых кадров* используется директива `@keyframes.`

11.1 Ключевые кадры `@keyframes`

Что из себя представляет любая анимация? Это переход от одного состояния элемента к другому состоянию.

Чтобы рассказать браузеру, с чего начать и чем закончить анимацию, используется директива `@keyframes`.

Представим, что у нас есть два блока: розовый круг и синий квадрат. Мы хотим написать анимацию так, чтобы розовый круг превращался в синий квадрат, а синий квадрат превращался в розовый круг (рисунок 11.1.1).

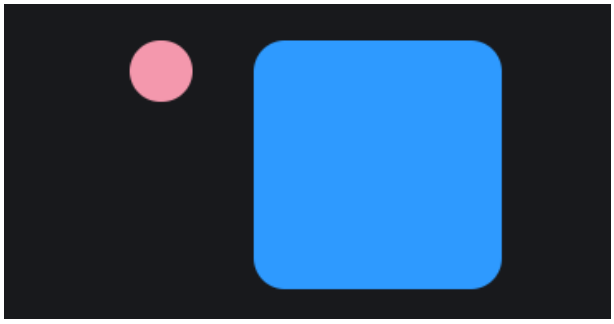


Рисунок 11.1.1 – Два блока для создания анимации

Начать создание анимации нужно с разложения её на шаги — *ключевые кадры*. Эта анимация простая - у неё будет всего два ключевых кадра.

Чтобы превратить розовый круг в синий квадрат, нужно будет поменять три свойства: `width`, `height` и `background-color`.

Чтобы прописать ключевые кадры используем директиву `@keyframes`.

После ключевого слова `@keyframes` необходимо написать имя анимации. Оно понадобится для того, чтобы связать анимацию для конкретного элемента с ключевыми кадрами. Желательно, чтобы имя анимации было уникальным (листинг 11.1.1).

Листинг 11.1.1 – Создание ключевых кадров `@keyframes`

```
@keyframes circle-to-square {
    from {
      width: 50px;
      height: 50px;
      background-color: #F498AD;
    }
    to {
      width: 200px;
      height: 200px;
      background-color: #2E9AFF;
    }
}
```

Примечание: если в коде встречается несколько директив с одинаковыми именами, то будет воспроизводиться последняя, стоящая ниже в коде анимация.

Ключевые кадры могут прописываться при помощи ключевых слов `from` (начальный кадр) и `to` (конечный кадр). Это удобно, если у вас всего два ключевых кадра. Если же кадров больше двух, то можно использовать проценты.

Добавим анимации промежуточный шаг, когда круг будет фиолетовым прямоугольником (листинг 11.1.2):

Листинг 11.1.2 – Промежуточный шаг анимации (50%)

```
@keyframes circle-to-square {
    from {
      width: 50px;
      height: 50px;
      background-color: #F498AD;
    }
    50% {
      width: 50px;
      height: 200px;
      background-color: #7F6EDB;
    }
    to {
      width: 200px;
      height: 200px;
      background-color: #2E9AFF;
    }
}
```

Браузер расшифровывает ключевое слово `from` как 0%, а ключевое слово `to` как 100%.

Далее, чтобы анимация начала проигрываться, нужно присвоить её какому-то элементу, чтобы браузер понимал, какой элемент на странице анимировать. Для этого существуют свойства анимации, представленные далее.

11.2 Свойства CSS-анимации

11.2.1 Свойство `animation-name`

Для присвоения анимации элементу нужно имя, которое мы задали ранее (как пример) (листинг 11.2.1).

Листинг 11.2.1 – Свойство `animation-name`

```
.child-one {
    animation-name: circle-to-square;
}
```

Теперь браузер знает, что ключевые кадры анимации с названием `circle-to-square` должны применяться к элементу с классом `child-one`.

Кроме имени анимации можно указать `none`, значение по умолчанию - означает отсутствие анимации. Удобно использовать для сброса анимации.

Например, можно указать это значение для элемента при наведении – с псевдоклассом `:hover`:

Листинг 11.2.2 – Свойство `animation-name` со значением `none`

```
.element {
    animation: some-animation;
  }

.element:hover {
    animation: none;
}
```

Далее, чтобы анимация заработала, необходимо задать время, за которое будет происходить изменение свойства элемента.

11.2.2 Свойство `animation-duration`

При помощи свойства `animation-duration` прописываем длительность одного цикла анимации. Значение этого свойства указывается в секундах `s` или миллисекундах `ms`.

Следующий код задает время на анимацию равное 5 секундам (листинг 11.2.3):

Листинг 11.2.3 – Свойство `animation-duration`

```
.child-one {
    animation-name: circle-to-square;
    animation-duration: 5s;
}
```

Примечание: если указать `0s`, то ключевые кадры будут пропущены, анимация применится мгновенно.

Таким образом, анимация теперь проигрывается, но только один раз.

11.2.3 Свойство `animation-iteration-count`

При помощи свойства `animation-iteration-count` можно указать, сколько раз анимация будет проигрываться.

В качестве значения указывается число, означающее количество повторений, или ключевое слово `infinite`. Если указано `infinite`, то анимация будет повторяться бесконечно. Это значение встречается чаще всего (листинг 11.2.4).

Листинг 11.2.4 – Свойство `animation-iteration-count` со значением `infinite`

```
.child-one {
    animation-name: circle-to-square;
    animation-duration: 5s;
    animation-iteration-count: infinite;
}
```

Теперь анимация будет проигрывается постоянно, но после последнего кадра происходит резкий скачок к исходному состоянию.

11.2.4 Свойство animation-direction

Свойство `animation-direction` сообщает браузеру, должна ли анимация проигрываться в обратном порядке.

Возможные значения:

- `normal` - значение по умолчанию, анимация воспроизводится от начала и до конца, после чего возвращается к начальному кадру.
- `reverse` - анимация проигрывается в обратном порядке, от последнего ключевого кадра до первого, после чего возвращается к последнему кадру.
- `alternate` - каждый нечётный повтор (первый, третий, пятый) анимации воспроизводится в прямом порядке, а каждый чётный повтор (второй, четвёртый, шестой) анимации воспроизводится в обратном порядке (листинг 11.2.5).
- `alternate-reverse` - аналогично значению `alternate`, но чётные и нечётные повторы меняются местами.

Листинг 11.2.5 – Свойство `animation-direction` со значением `alternate`

```
.child-one {
    animation-name: circle-to-square;
    animation-duration: 5s;
    animation-iteration-count: infinite;
    animation-direction: alternate;
}
```

Теперь анимация красиво проигрывается. Круг плавно становится квадратом, а потом снова плавно превращается в круг.

Дальнейшее улучшение анимации возможно посредством следующих свойств.

11.2.5 Свойство `animation-timing-function`

CSS-анимации по умолчанию проигрываются линейно, меняя свойства элемента на равные доли в равные промежутки времени. Такое поведение редко встречается в реальной жизни. Так, например, мячик, который вы подбрасываете в руках, начинает своё движение медленно и со временем ускоряется.

При помощи свойства `animation-timing-function` можно задать, как будет развиваться анимация между ключевыми кадрами: равномерно, или сначала быстро, потом медленно, или по каким-то сложным внутренним законам.

Возможные значения:

- `linear` - значение по умолчанию. Анимация проигрывается равномерно, без колебаний скорости.

- `ease` - анимация начинается медленно, затем быстро разгоняется и снова замедляется к концу.
- `ease-in` - анимация начинается медленно и плавно ускоряется к концу.
- `ease-out` - анимация начинается быстро и плавно замедляется к концу.
- `ease-in-out` - анимация начинается и заканчивается медленно, ускоряясь в середине.
- `cubic-bezier(x1, y1, x2, y2)` - временная функция, описывающая тип ускорения в виде кривой Безье. По оси *x* располагается временная шкала анимации, а по оси *y* — прогресс анимации. Это очень мощный инструмент для создания разнообразных анимаций со сложными внутренними законами.

  Чаще всего используется инструмент визуализации, позволяющий изменять значения и сразу видеть, как будет выглядеть анимация. Один из самых популярных инструментов — https://cubic-bezier.com/#.17,.67,.83,.67.
- `step-start` - задаёт пошаговую анимацию, разбивая её на отрезки, изменения происходят в начале каждого шага.
- `step-end` - пошаговая анимация - изменения происходят в конце каждого шага.
- `steps(количество шагов, положение шага)` - функция, указывающая, что анимация должна воспроизводиться шагами, резко переходя от одного состояния к другому.

  Первый параметр указывает на сколько отрезков нужно разбить анимацию. Значением должно быть целое положительное число больше 0.

  Второй параметр является необязательным и указывает позицию шага - момент, когда начинается анимация. Возможные значения:

  - `jump-start` - первый шаг происходит при значении 0.
  - `jump-end` - последний шаг происходит при значении 1.
  - `jump-none` - все шаги происходят в пределах от 0 до 1 включительно.
  - `jump-both` - первый шаг происходит при значении 0, последний - при значении 1.
  - `start` - ведёт себя как jump-start.
  - `end` - ведёт себя как jump-end.

Со значением `start` анимация начинается в начале каждого шага, со значением `end` - в конце каждого шага с задержкой. Задержка вычисляется как результат деления времени анимации на количество шагов. Если второй параметр не указан, используется значение по умолчанию `end`.

Наглядное представление каждого значения свойства `animation-timing-function` можно увидеть по ссылке: https://codepen.io/kamilniftaliev/full/XXmjER

Вернёмся к розовому кругу и укажем, что он должен превращаться в синий квадрат нелинейно, медленно разгоняясь к концу анимации (листинг 11.2.6).

Листинг 11.2.6 – Свойство `animation-timing-function` со значением `ease-in`

```
.child-one {
    animation-name: circle-to-square;
    animation-duration: 5s;
    animation-iteration-count: infinite;
    animation-direction: alternate;
    animation-timing-function: ease-in;
}
```

И, в дополнение, превратим синий квадрат в розовый круг. Используем практически те же самые свойства, что и для круга, только зададим другое значение для свойства `animation-direction`, чтобы шаги анимации воспроизводились в обратном порядке (листинг 11.2.7):

Листинг 11.2.7 – Свойство `animation-direction` со значением `alternate-reverse`

```
.child-two {
    animation-name: circle-to-square;
    animation-duration: 5s;
    animation-iteration-count: infinite;
    animation-direction: alternate-reverse;
    animation-timing-function: ease-in;
}
```

Теперь обе фигуры меняются синхронно. Далее добавим правой фигуре – квадрату - небольшую задержку.

11.2.6 Свойство `animation-delay`

Свойство `animation-delay` задаёт задержку воспроизведения анимации. Значением может быть любое число, как отрицательное, так и положительное.

Если значение положительное, то будет задержка перед началом анимации. Если значение отрицательное, то анимация начнётся как бы за кадром.

Пусть анимация второго элемента – квадрата - начнётся с задержкой -2.5 секунды. Так она будет начинаться с середины (листинг 11.2.8):

Листинг 11.2.8 – Свойство `animation-delay`

```
.child-two {
    animation-name: circle-to-square;
    animation-duration: 5s;
    animation-iteration-count: infinite;
    animation-direction: alternate-reverse;
    animation-timing-function: ease-in;
    animation-delay: -2.5s;
}
```

11.2.7 Свойство `animation-play-state`

Свойство, позволяющее ставить анимацию на паузу и запускать снова.

Возможные значения:

- `running` — анимация проигрывается (значение по умолчанию).
- `paused` — анимация ставится на паузу. При повторном запуске анимации она продолжается с того места, где была остановлена.

Добавим возможность взаимодействовать с анимацией. Пусть по наведению курсора анимация ставится на паузу, а если курсор убран, то продолжает проигрываться (листинг 11.2.9).

Листинг 11.2.9 – Свойство `animation-play-state` со значением `paused`

```
.child:hover {
    animation-play-state: paused;
}
```

11.2.8 Свойство `animation-fill-mode`

Свойство анимации `animation-fill-mode` сообщает браузеру нужно ли применять стили ключевых кадров до или после проигрывания анимации.

Возможные значения:

- `none` - стили анимации не применяются до и после проигрывания анимации (значение по умолчанию).
- `forwards` - после окончания анимации элемент сохранит стили последнего ключевого кадра.
- `backwards` - после окончания анимации к элементу будут применены стили первого ключевого кадра.
- `both` - до начала анимации к элементу применяется первый ключевой кадр, а после окончания анимации элемент останется в состоянии последнего ключевого кадра.

Наглядное представление каждого значения свойства `animation-fill-mode` можно увидеть по ссылке: https://codepen.io/kikiki_kiki/pen/YBQErx

11.2.9 Свойство `animation`

Свойство `animation` - это свойство, в котором можно указать значения для всех перечисленных выше свойств, начинающихся на `animation-`.

Значения указываются через пробел. Порядок указания значений не важен. Из-за того, что значения этих свойств очень разные, браузер сам догадывается, какое значение к какому свойству относится. Важно только помнить, что первое значение времени будет воспринято как значение `animation-duration` (длительность анимации), а второе — `animation-delay` (задержка воспроизведения).

Для работы анимации совсем не обязательно перечислять все значения. Достаточно указать имя анимации и её длительность. Для остальных свойств будут установлены значения по умолчанию (листинг 11.2.10).

Листинг 11.2.10 – Свойство `animation`

```
.child-two {
    animation: circle-to-square 2s infinite alternate-reverse ease-in 1s;
}
```

11.3 Использование нескольких CSS-анимаций

Есть возможность применить к одному элементу сразу несколько анимаций. Для этого нужно перечислить несколько значений через запятую. Возможно указать любое

количество значений для любого из свойств анимации. Анимации будут воспроизводиться одновременно.

Например, разобьём предыдущую анимацию на две отдельные. Одна будет отвечать за изменение формы элемента, а вторая за изменение цвета (листинг 11.3.1).

Листинг 11.3.1 – Использование двух отдельных анимаций

```
@keyframes circle-to-square {
    from {
      width: 50px;
      height: 50px;
    }
    50% {
      width: 100%;
      height: 50px;
    }
    to {
      width: 100%;
      height: 100%;
    }
}

@keyframes color-change {
    from {
      background-color: #F498AD;
    }
    50% {
      background-color: #7F6EDB;
    }
    to {
      background-color: #2E9AFF;
    }
}
```

Далее присвоим обе анимации одному элементу, при этом задав первой задержку, и указав разное время воспроизведения (листинг 11.3.2).

Листинг 11.3.2 – Использование двух отдельных анимаций для одного элемента

```
.child {
    animation:
      circle-to-square 10s infinite alternate ease-out 1s,
      color-change 5s alternate linear infinite;
}
```

В итоге форма меняется за 10 секунд, а цвет перетекает из розового в синий за 5 секунд. А потом обратно.

11.4 Свойство `transition`

Свойство `transition` используется, когда нам нужно плавно изменить CSS-свойства между двумя состояниями элемента. Например, при наведении мышкой.

Свойство `transition` это свойство-шорткат, т.е. объединяющее несколько других свойств. Как, например, `margin` или `background`. Оно включает в себя несколько подсвойств:

- `transition-property` - плавность изменения;
- `transition-duration` - длительность перехода;
- `transition-timing-function` - функция, описывающая скорость изменения свойства;
- `transition-delay` - задержка перед началом изменения.

1. Порядок записи свойства `transition` при применении его **к одному свойству**:

`имя свойства | длительность | временная функция | задержка`

(листинг 11.4.1).

Листинг 11.4.1 – Свойство `transition` к одному свойству – `margin`

```
.selector {
    transition: margin-left 4s ease-in-out 1s;
}
```

Можно использовать также либо `имя свойства | длительность`, либо `имя свойства | длительность | задержка`.

2. Порядок записи свойства `transition` при применении его **к двум свойствам через запятую** (листинг 11.4.2):

Листинг 11.4.2 – Свойство `transition` к двум свойствам – `margin` и `color`

```
.selector {
    transition: margin-left 4s, color 1s;
}
```

3. Порядок записи свойства `transition` при применении его **ко всем свойствам, которые будут меняться** (листинг 11.4.3):

Листинг 11.4.3 – Свойство `transition` ко всем свойствам

```
.selector {
    transition: all 0.5s ease-out;
}
```

Примечания:

1. С помощью `transition` можно плавно изменять любое свойство, у которого значение записывается с помощью чисел (например, `margin`). Исключения: `visibility`, `z-index`.
2. Длительность перехода может задаваться в секундах (`0.3s`) или в миллисекундах (`300ms`). Ноль перед точкой можно не писать (`.3s`).
3. Значение свойства `z-index` записывается числом, но его нельзя плавно изменить никаким способом.
4. Значение свойства `visibility` записывается строкой, но его в связке с `opacity` можно плавно изменять при помощи `transition`.

   Если использовать только `opacity`, то элемент станет невидимым, но будет доступен для взаимодействия с мышкой и клавиатурой.

   Если использовать только `visibility`, то скрытие и появление не будет плавным.

11.5 Свойство `transform`

Свойство `transform` используется, когда нужно применить к элементу какие-либо трансформации: искажение, поворот, смещение, масштабирование.

Часто бывает необходимо каким-то образом трансформировать визуальное представление элемента (масштабировать, повернуть, переместить) и при этом никак не затронуть соседние элементы в документе. Он как бы «приподнимается» над остальным содержимым. При этом он не уходит из потока документа, и остальные элементы располагаются так, как располагались до применения трансформаций. Для подобных преобразований используется свойство `transform`. В качестве значения выступают различные функции трансформации: `rotate`, `translate`, `scale`, `skew` и другие.

Примеры кода с использованием функций трансформации представлены ниже в листингах 11.5.1-11.5.2:

Листинг 11.5.1– Смещение визуального представления элемента на 120 пикселей вправо

```
.selector {
    transform: translateX(120px);
}
```

Листинг 11.5.2 – Несколько значений функций трансформации

```
.selector {
    transform: perspective(500px) translate(10px, 0, 20px) rotateY(3deg);
}
```

Группы функций трансформации

Функции перемещения

1. `translate(X, Y)`

Функция используется для смещения элемента вверх-вниз или влево-вправо. В целом, ту же работу выполняют CSS-свойства `top`, `right`, `bottom`, `left` — например, для абсолютно (`position: absolute`) или относительно (`position: relative`) спозиционированных элементов. Но есть ряд важных отличий: элемент позиционируется относительно соответствующих сторон родителя. То есть `left: 20px` сместит элемент на 20 пикселей относительно левой границы родителя, а `translate(20px)` сместит элемент вправо относительно того места, где тот находился до трансформации. В целом, позиционирование и `translate` прекрасно сочетаются друг с другом. Позиционирование лучше использовать для изначального расположения элемента на странице, а `translate` применять, если нужно добавить анимации движения.

Функция принимает два параметра: первый параметр отвечает за смещение вправо-влево, а второй параметр — вверх-вниз. Если передать только один параметр, тогда смещение будет только вправо-влево. Мы можем использовать любые единицы измерения расстояния из CSS, например, абсолютные `10px` или относительные `50%`. Абсолютное значение используется «как есть»: элемент сместится на 10 пикселей. Относительное значение считается относительно размеров самого элемента. При указании 50% элемент сместится на половину собственной ширины или высоты.

2. `translateX(X)`, `translateY(Y)`, `translateZ(Z)`

Когда нужно сместить элемент вдоль конкретной оси, можно применить соответствующие функции трансформации.

3. **`translate3d(X, Y, Z)`**

Если нужно сместить элемент по всем трём осям, можно всё собрать воедино и использовать эту функцию.

Функции масштабирования

1. **`scale(X, Y)`**

Функция для масштабирования элемента. Значения X и Y — это положительные числа, либо 0. Если в функцию передать 0, то элемент не будет виден. Единица соответствует нормальному масштабу. Числа от 0 до 1 — это уменьшенный масштаб. Числа больше единицы — увеличенный масштаб. Например, чтобы визуально увеличить элемент в 2 раза, нужно написать `transform: scale(2)`.

В отличие от `translate`, один параметр в функции `scale` работает несколько иначе. `scale(2)`- это то же самое, что `scale(2, 2)`, то есть одно число указывает на пропорциональное масштабирование по обеим осям одновременно.

2. **`scaleX(X), scaleY(Y), scaleZ(Z)`**

Используется, когда необходимо растягивать или сжимать элемент только по горизонтали, вертикали или третьей оси Z.

3. **`scale3d(X, Y, Z)`**

Если нужно масштабировать элемент по всем трём осям, можно всё собрать воедино и использовать эту функцию.

Функции наклона

`skewX(X), skewY(Y)`

Функции выполняют сдвиг одной стороны элемента относительно противолежащей. В результате элемент как бы наклоняется. Величина наклона зависит от положения точки применения трансформаций (`transform-origin`) и числа градусов, заданных в параметрах.

`skewX` сдвигает верхнюю сторону элемента относительно нижней.

`skewY` сдвигает правую сторону относительно левой.

Функции поворота

1. `rotateX(X), rotateY(Y), rotateZ(Z)`

Кроме сдвига или наклона, элемент можно вращать. В функцию передаём единицы измерения углов (*deg, rad, turn*), например, `45deg` или `0.5turn`. Обратите внимание, что обычное вращение элемента на странице — это вращение относительно оси Z. Если мы хотим вращать элемент относительно других осей, то нужно не забывать про перспективу. С ней повороты относительно X или Y будут выглядеть максимально естественно.

2. `rotate(Z)`

Функция аналогична `rotateZ(Z)`. Чтобы не запоминать ось для типового вращения элемента, можно использовать просто слово `rotate`.

3. `rotate3d(X, Y, Z)`

Если нужно повернуть элемент по всем трём осям, можно всё собрать воедино и использовать эту функцию.

Прочие функции

1. `matrix(a, b, c, d, tx, ty)`

`matrix()` — это функция, которой можно описать любую трансформацию в плоскости экрана. Она использует матричные преобразования и может заменить собой все вышеописанные функции. Но при этом она очень сложно читается. Например, глядя на функцию `matrix(0.707107, 0.707107, -0.707107, 0.707107, -0.707107, 34.6482)` невозможно сразу точно определить, что она аналогична записи `rotate(45deg) translate(24px, 25px)`. Зачем же она нужна, такая сложная, если проще описать трансформации соответствующими функциями? Например, с её помощью можно писать сложные динамические анимации. Популярные JS-библиотеки для анимации используют как раз матричные преобразования, а не конкретные функции трансформации.

2. `matrix3d(a1, b1, c1, d1, a2, b2, c2, d2, a3, b3, c3, d3, a4, b4, c4, d4)`

Если нужно произвести трансформации в трёхмерном пространстве, а не в плоскости экрана, то нужно использовать эту функцию.

3. `perspective(Z)`

Несмотря на то, что экран плоский, у нас всё равно есть возможность перемещать элемент вдоль оси Z. Она направлена перпендикулярно плоскости экрана в сторону пользователя. Если мы используем `translateZ` просто так, то никакого перемещения ближе или дальше мы не увидим. Чтобы было ощущение движения к нам или от нас, элемент должен становиться крупнее или мельче. Эта разница в размерах — следствие перспективы. Элемент на экране может вести себя подобно объектам в реальном мире и менять размер при перемещении к нам или от нас. Чтобы это заработало, нужно элементу задать свойство `perspective`. Это свойство необходимо применять при любых трансформациях, выходящих из плоскости экрана.

Функция `perspective()` принимает один параметр — расстояние до точки схождения перспективы. Плоскость экрана принимается за начало координат. Например, запись `perspective(500px)` означает, что точка схождения перспективы находится как бы на расстоянии 500 пикселей вглубь от плоскости экрана.

Примечания:

1. Если среди значений есть функция `perspective()`, то она должна быть первой среди всех значений.
2. В последних версиях спецификации появились отдельные CSS-свойства для трансформаций. Это `rotate`, `translate` и `scale`. Если раньше мы писали комплексные трансформации, применяя несколько функций, то теперь каждую трансформацию можем описать отдельным свойством.
3. Если свойство `transform` имеет значение, отличное от `none`, то создаётся новый контекст наложения. Это означает, что относительно этого элемента теперь будут позиционироваться все дочерние элементы, у которых `position: fixed` или `position: absolute`.

Наглядное представление каждого значения свойства `transform` можно увидеть по ссылке: https://codepen.io/vineethtrv/pen/XKKEgM

Дополнительная информация

1. Animatable CSS properties – URL: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_animated_properties (англ. яз.)
2. Animate.css - URL: https://animate.style/ (англ. яз.)
3. An Interactive Guide to CSS Transitions – URL: https://www.joshwcomeau.com/animation/css-transitions/ (англ. яз.)
4. The World of CSS Transforms – URL: https://www.joshwcomeau.com/css/transforms/ (англ. яз.)
5. An Interactive Guide to Keyframe Animations – URL: https://www.joshwcomeau.com/animation/keyframe-animations/ (англ. яз.)

12 Свойство box-shadow

Свойство, с помощью которого можно задать блоку тень. Создано, чтобы имитировать объекты реального мира и создавать иллюзию объёма для плоских элементов интерфейса.

Каждая тень состоит из следующих значений:

- Два, три или четыре значения размера с единицами измерения:
  - Если задано **два значения**, то они расшифровываются как смещение по оси *x* и по оси *y*.
  - Если задано **третье значение**, то оно интерпретируется как радиус размытия.
  - Если задано **четвёртое значение**, то оно отвечает за радиус распространения.
- Дополнительно (необязательно) можно указать ключевое слово `inset`, которое превратит тень из внешней во внутреннюю.
- Чаще всего, но не обязательно, нужно указывать цвет тени в любом доступном формате цвета.

1. Смещения по осям *x* и *y* - обязательные значения для тени. Могут принимать любые числовые значения, в том числе отрицательные. Значение по умолчанию равно 0 для обеих осей. Если первое значение положительное, то тень будет справа от элемента, если отрицательное — слева. Если второе значение положительное, то тень будет снизу, если отрицательное — сверху.

2. Радиус размытия - положительное числовое значение с единицами измерения. По умолчанию значение 0, что делает его указание необязательным. Если не

указывать его или задать 0, то край тени будет резким, без размытия. **Чем больше значение, тем шире область размытия и тем светлее сама тень.**

3. Радиус распространения - любое числовое значение с единицами измерения. По умолчанию равно 0, размеры тени совпадают с размерами элемента. Если указано отрицательное значение, то тень будет меньше, если положительное, то тень будет больше.

4. Цвет - опциональное, но на самом деле обязательное значение цвета тени. Если не указывать цвет, то решение остаётся за браузером. Как правило, браузер берёт значение свойства `color` того элемента, которому задана тень. Но Safari отрисует прозрачную тень. Если вам действительно нужен цвет тени, совпадающий с цветом текста элемента, то это можно указать явно при помощи ключевого слова `currentColor`.

5. `inset` - если ключевое слово не указано в значении, то тень располагается снаружи элемента. Если указать это ключевое слово, то элемент как будто будет вдавлен внутрь и его стенки будут отбрасывать тень внутрь.

Можно задавать несколько теней для одного элемента, перечисляя их через запятую (листинг 12.1.1).

Листинг 12.1.1 – Несколько теней для одного элемента

```
.button {
    box-shadow:
      0 5px 10px gray,
      -5px -10px 20px green;
}
```

Примечания:

1. Свойство `box-shadow` задаёт тень именно для блока. Тень будет совпадать с формой блока. Если вы сделали круглый блок, то тень тоже будет круглой. Если не менять форму элемента, то тень будет прямоугольной.
2. Если нужна **тень для букв в тексте**, используйте свойство `text-shadow`.
3. Тени можно использовать для hover-эффектов (например, псевдокласса `:active`). При помощи теней кнопка сделана выпуклой, а при нажатии она становится вдавленной за счёт изменения положения теней.

Практическое задание

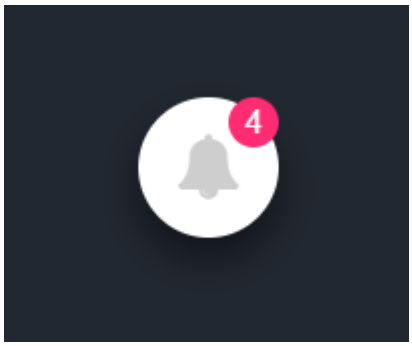
Примечание: в заданиях должна быть адаптивность (медиа-запросы, свойство `flex-wrap` или относительные единицы измерения: *%, rem, em, vw, vh* и т.д.). Контент страницы всегда должен пропорционально умещаться на экране устройств с шириной от 320px до 2560px.

1. Создать 4 различных кнопки (примеры можно выбрать из **видео** и модифицировать под себя) с применением свойств `transition` и `transform`, псевдокласса `:hover` и псевдоэлементов `::before` и `::after`.

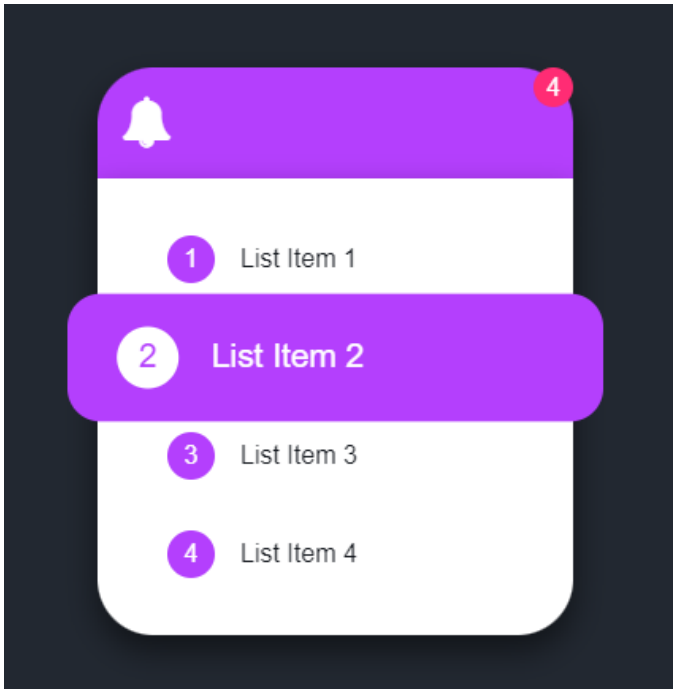
2. Создать элемент уведомлений. При наведении должен появляться список.

   Пример (модифицировать под тему своего сайта):

   До наведения:



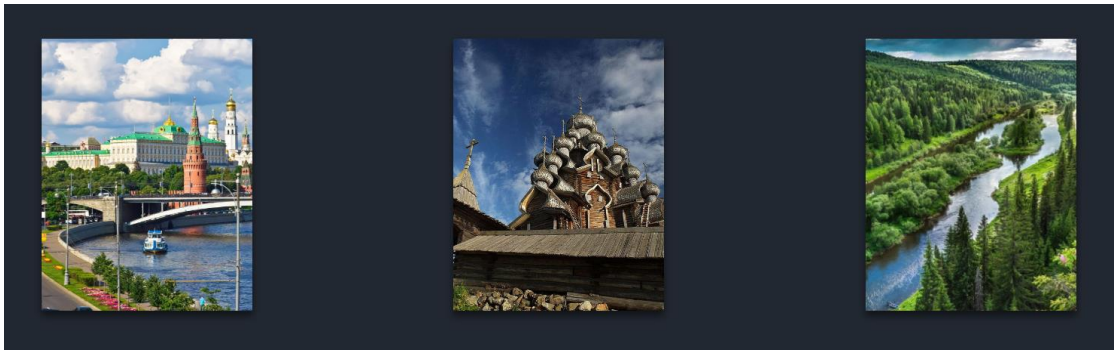
   После наведения:



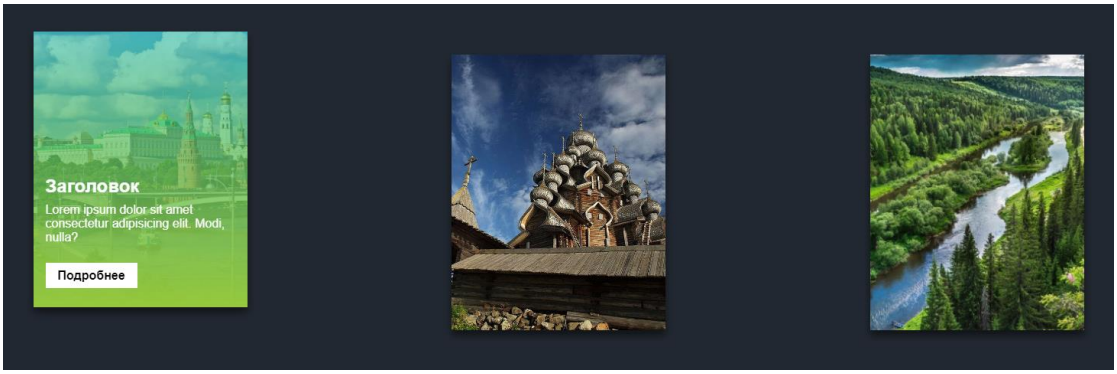
3. Создать три карточки **на выбор**. Использовать свойства flexbox (`transition`, `transform`) и псевдоэлемент `::before`. Добавить плавную анимацию при наведении.

Пример №1 (модифицировать под тему своего сайта):

До наведения:

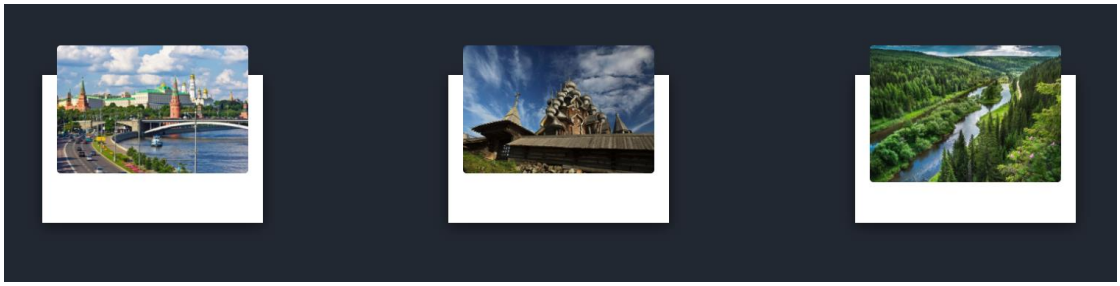


После наведения:

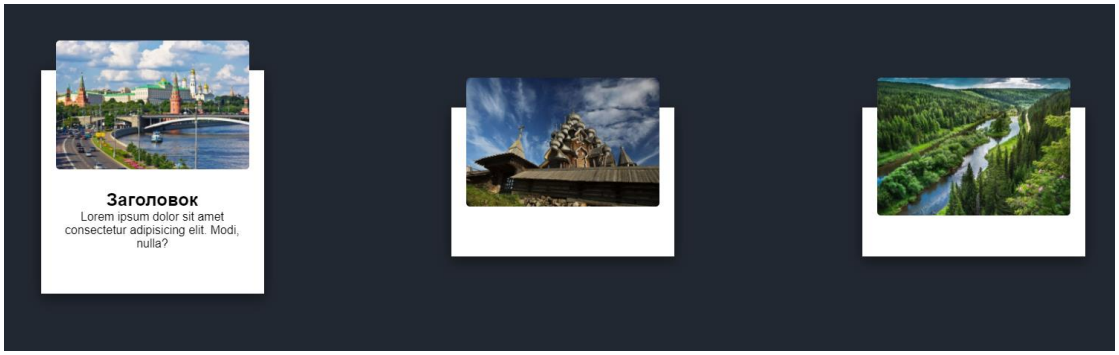


Пример №2 (модифицировать под тему своего сайта):

До наведения:



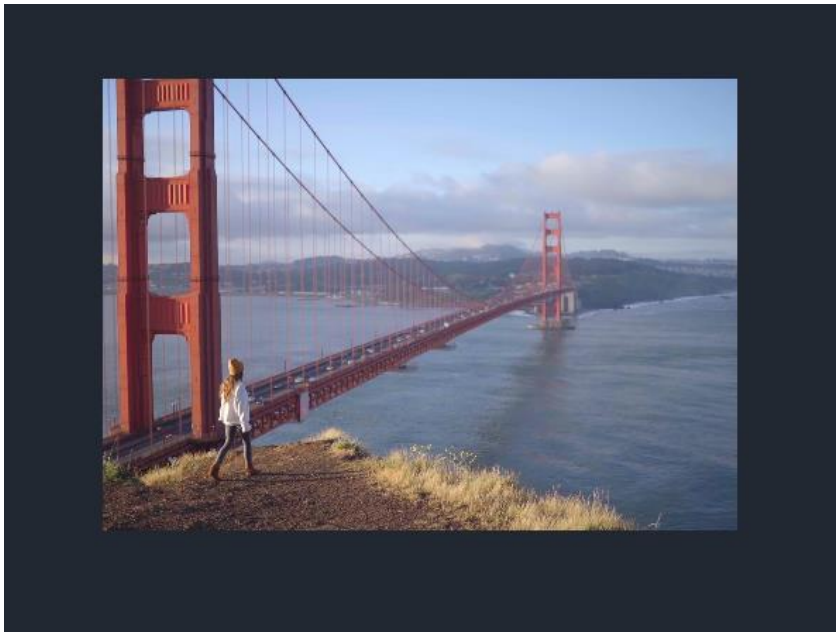
После наведения:



4. Создать контейнер, где будет находиться видео, и небольшую подпись. Использовать свойства flexbox (`transition`, `transform`) и псевдоэлемент `::before`. Добавить плавную анимацию при наведении.

Пример (модифицировать под тему своего сайта):

До наведения:



После наведения: