

Практическая работа №7 «Функции `calc()` и `var()` в CSS. Свойство `filter`, функции фильтров. Свойство `backdrop-filter`, эффект `glassmorphism`. Свойство `cursor`»

13 Функция `calc()`

Функция `calc()` - функция, позволяющая производить математические вычисления прямо в CSS.

Как самый простой пример использования функции `calc()` можно привести частую ситуацию в верстке, когда необходимо разделить контент на три колонки, ширина каждой из которых равна трети от 100%. Изначально указывалось значение $100\% / 3 = 33,33333\ldots\%$; при округлении, соответственно, получается 33%, но теперь можно использовать `calc()` и браузер сам рассчитает ширину колонок (листинг 13.1).

Листинг 13.1 – Добавление функции `calc()`

```
.selector {  
    width: calc(100% / 3);  
}
```

Примечание: при использовании этой функции следует учитывать, что на каждую операцию браузер тратит некоторую долю секунды и часть оперативной памяти. Если расчётов будет много, то это потенциально повлияет на скорость загрузки страницы. Используйте `calc()` с умом.

Синтаксис функции `calc()`

В круглых скобках можно писать любые математические операции с любыми единицами измерения, доступными в вебе (`%`, `px`, `rem`, `em`, `vw`, `vh`, `vmin` и т.д.). Доступны четыре стандартных операнда:

- `+` - сложение;
- `-` - вычитание;
- `/` - деление;
- `*` - умножение.

Операторы сложения и вычитания обязательно с двух сторон должны отбиваться пробелом. Иначе браузер воспримет их как часть числа. Хотя операторы деления и умножения не требуют такой строгости к себе, принято и их тоже отбивать пробелами для удобства чтения.

Так, например, выражение `calc(50% -брх)` будет интерпретировано как величина в процентах и следующее за ним отрицательное число в пикселях (не верное выражение), в то время как `calc(50% - брх)` - правильное выражение, будет интерпретировано как вычитание из процентов длины в пикселях.

Внутри скобок может быть больше одного вычисления, можно группировать операции при помощи скобок. Но не стоит увлекаться: чем короче вычисление, тем проще потом его прочитать и понять, что в нем прописано.

Функцию `calc()` можно использовать для любых свойств, значением которых должна быть цифра. Причём если свойство предполагает составное значение, то можно указать функцию как часть этого значения (листинг 13.2):

Листинг 13.2 – Функция `calc()` для CSS-свойства

```
.selector {  
  margin: calc(5vh / 4) 20px;  
  transition: transform calc(0.5s + 120ms);  
}
```

Также эту функцию можно использовать вместе с переменными `var()` (листинг 13.3):

Листинг 13.4 – Функция `calc()` с переменной

```
.selector {  
  width: calc(var(--variable-width) + 20px);  
}
```

Внутри круглых скобок можно складывать только числовые значения. Нельзя сложить число со строкой.

Также неудачным местом применения этой функции являются **медиавыражения**. Вот такая запись считается не валидной (листинг 13.4):

Листинг 13.4 – Не валидная функция `calc()`

```
@media (min-width: calc(465px + 1vw)) {  
  .wiz:  
}
```

Когда функция `calc()` используется для управления размером текста, убедитесь, что одно из значений включает относительную единицу длины, например (листинг 13.5):

Листинг 13.5 – Функция `calc()` для размера шрифта

```
h1 {  
  font-size: calc(1.5rem + 3vw);  
}
```

14 Переменные в CSS – функция `var()`

Пользовательские свойства (иногда называемые переменными CSS или каскадными переменными) — это объекты, определенные авторами CSS и содержащие определенные значения, которые можно повторно использовать во всем документе. Они задаются с использованием обозначения пользовательских свойств (например, `--main-color: black;`) и доступны с помощью функции `var()` (например, `color: var(--main-color);`).

Сложные веб-сайты содержат очень большое количество CSS, часто с большим количеством повторяющихся значений. Например, один и тот же цвет может использоваться в сотнях разных мест, что потребует глобального поиска и замены, если этот цвет необходимо изменить. Пользовательские свойства позволяют хранить значение в одном месте, а затем использовать его в нескольких других местах. Дополнительным преимуществом являются семантические идентификаторы. Например, `--main-text-color` легче понять, чем `#00ff00`, особенно если этот же цвет используется и в других контекстах.

Пользовательские свойства подлежат каскадированию и наследуют свое значение от своего родителя.

Синтаксис

Объявление настраиваемого свойства выполняется с использованием имени настраиваемого свойства, которое начинается с двойного дефиса (`--`), и значения свойства, которое может быть любым допустимым значением CSS. Как и любое другое свойство, оно записывается внутри набора правил, например (листинг 14.1):

Листинг 14.1 – Объявление переменной

```
body {  
  --font-s: 20px;  
}
```

Теперь эту переменную можно использовать вместо значений в свойствах стилей. Для этого пишем `var()`, а в скобках — название переменной (листинг 14.2). Когда браузер встретит такую конструкцию, он подставит вместо неё значение переменной.

Листинг 14.2 – Использование переменной

```
body {  
  --font-s: 20px;  
  font-size: var(--font-s);  
}
```

Базовый синтаксис функции `var()`

Функция `var()` принимает 2 аргумента:

1. Имя кастомного свойства.
2. Резервное значение (необязательный).

Резервное значение кастомного свойства. Используется в случае, если кастомное свойство не определено или не может быть использовано в текущем контексте. Резервное значение может содержать любой символ, кроме некоторых спецсимволов, вроде символа новой строки, непарных закрывающих скобок (т.е. `)`, `]`, или `}`), точку с запятой или восклицательный знак.

Функции в качестве первого аргумента обязательно нужно передать имя кастомного свойства, значение которого нужно получить. **Необязательный второй аргумент функции используется в качестве резервного значения**. Если кастомное свойство, указанное в первом аргументе, оказалось недоступным, функция вернёт второе значение (листинг 14.3):

Листинг 14.3 – Пример базового синтаксиса

```
.card {  
  padding: var(--card-padding, 10px);  
}
```

Дело в том, что всё, что находится после первой запятой, воспринимается как единое значение. То есть в качестве резервного значения функция вернёт значение от первой запятой до закрывающей скобки.

Резервное значение (листинг 14.4) будет: `20px, 20px`:

Листинг 14.4 – Пример с резервными значениями

```
.navigation {  
  --translate: var(--my-translate, 20px, 20px);  
  transform: translate(var(--translate));  
}
```

Имена пользовательских свойств чувствительны к регистру — `--my-color` будет рассматриваться как отдельное пользовательское свойство `--My-color`.

Область видимости

Переменные видны в том блоке, в котором их объявляют.

Обратите внимание, что **селектор, заданный набору правил, определяет область, в которой может использоваться пользовательское свойство**. Ещё переменные действуют во всех вложенных элементах внутри этого блока: если мы объявим переменную внутри блока `body {}` или `html {}`, то она будет действовать для всех элементов на странице, потому что они находятся внутри этих блоков.

Обычно рекомендуется определять пользовательские свойства в псевдоклассе `:root`, чтобы их можно было применять **глобально во всем HTML-документе** (листинг 14.2):

Листинг 14.2 – Глобальное объявление переменной

```
:root {  
  --main-color: aqua;  
}
```

Также переменные действуют внутри дочерних элементов: если мы объявим переменную для описания CSS-свойства класса, то и, например, абзацы с этим классом будут поддерживать эту переменную (листинг 14.12):

Листинг 14.12 – Дочерний элемент с переменной

```
.alert {  
  --alert-color: aqua;  
}  
.alert p {  
  color: var(--alert-color);  
  border: 2px solid var(--alert-color);  
}
```

Возможности

Функцию `var()` можно подставить как часть значения свойства (листинг 14.6):

Листинг 14.6 – Переменная как часть значения свойства

```
.card {  
  --border-color: green;  
  border: 2px solid var(--border-color);  
}
```

Если нам понадобится поменять везде размер шрифта и активный цвет, то придётся перебирать вручную все стили и смотреть, не забыли ли мы чего сделать. Вместо этого можно использовать переменные — так мы можем поменять значение в одном месте, и оно сразу будет использоваться во всех местах (листинг 14.13):

Листинг 14.13 – Пример использования переменных с `:root`

```
:root{  
  --font-s: 20px;  
  --color-f: #32590a;  
}  
  
p {  
  font-size: var(--font-s);  
  color: var(--color-f);  
}  
  
a {  
  font-size: var(--font-s);  
  color: var(--color-f);  
}  
  
.tags {  
  font-style: italic;  
  color: var(--color-f);  
}
```

Функция `var()` также работает с сокращёнными свойствами: `margin`, `padding`, `border`, `background`, `transform`, `transition` и т.д.

Можно использовать для подставки как одного из значений (листинг 14.7):

Листинг 14.7 – Переменная как одно из значений

```
.element {  
  --margin-top: 10px;  
  margin: var(--margin-top) 10px 20px 30px;  
}
```

Так и для нескольких (листинг 14.8):

Листинг 14.8 – Переменная как несколько значений свойства

```
.element {
  --margin-top-right: 10px 10px;
  margin: var(--margin-top-right) 10px 50px;
}
```

15 Свойство `filter`

Свойство `filter` даёт возможность прямо в CSS применять к элементам различные графические эффекты. Например, перекрасить элемент в сепию, добавить ему необычную тень или использовать SVG в качестве фильтра.

Фильтр — очень мощный инструмент, он позволяет реализовать многие дизайнерские идеи при помощи одной строчки кода. Фильтры можно применить не только к картинкам, но и к любым непустым элементам.

В использовании фильтров удобно то, что исходная картинка или элемент никак не изменяются сами по себе. Меняется только их визуальное отображение в браузере. А это значит, что вы сможете использовать их повторно в других ситуациях с другими фильтрами или совсем без них.

Можно не только задавать статичные визуальные эффекты, но и анимировать их.

Функции фильтров

Список функций, с помощью которых можно задать фильтры картинкам.

Функции используются в качестве значений для свойств `filter` и `backdrop-filter`.

1. Функция `blur()`

Примеряет размытие Гаусса к изображению. Значение в скобках указывает сколько пикселей сливаются друг с другом. Чем больше значение, тем больше размытие. Можно указать положительное значение в любых единицах измерения, кроме процентов (листинг 15.1 и рисунок 15.1).

Листинг 15.1 – Функция `blur()`

```
img {
  filter: blur(5px);
}
```



Рисунок 15.1 – Применение функции `blur()` к изображению

2. Функция `brightness()`

Меняет яркость изображения. В скобках можно указать любое значение от 0% и выше. Значение 0% сделает изображение полностью чёрным. Значение 100% вернёт изображению исходную яркость. Значение больше 100% усилит яркость картинки. Значением может быть целое или дробное число без единиц измерения (листинг 15.2 и рисунок 15.2).

Листинг 15.2 – Функция `brightness()`

```
img {  
  filter: brightness(30%);  
}
```

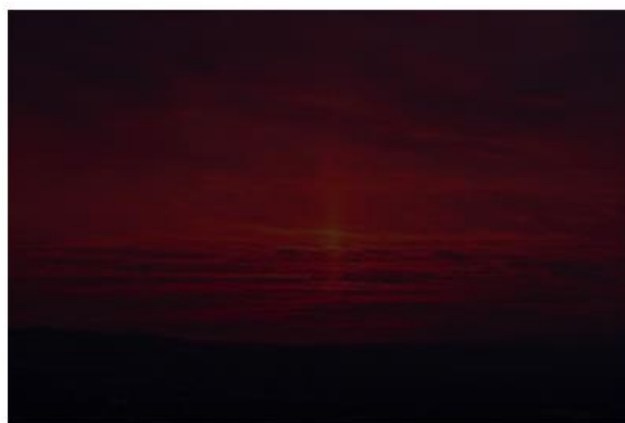


Рисунок 15.2 – Применение функции `brightness()` к изображению

3. Функция `contrast()`

Меняет контраст изображения. В скобках можно указать любое значение от 0% и выше. Значение 0% сделает изображение полностью чёрным. Значение 100% вернёт изображению исходный контраст. Значение больше 100% усилит исходный контраст.

Значением может быть целое или дробное число без единиц измерения (листинг 15.3 и рисунок 15.3).

Листинг 15.3 – Функция `contrast()`

```
img {  
  filter: contrast(250%);  
}
```



Рисунок 15.3 – Применение функции `contrast()` к изображению

4. Функция `drop-shadow()`

Задаёт тень для картинки. Тень располагается снаружи элемента (листинг 15.4 и рисунок 15.4).

Эта функция очень похожа на `box-shadow` по допустимым значениям и результату. Разница лишь в том, что нельзя указывать ключевое слово `inset`.

Листинг 15.4 – Функция `drop-shadow()`

```
img {  
  filter: drop-shadow(7px 7px red);  
}
```

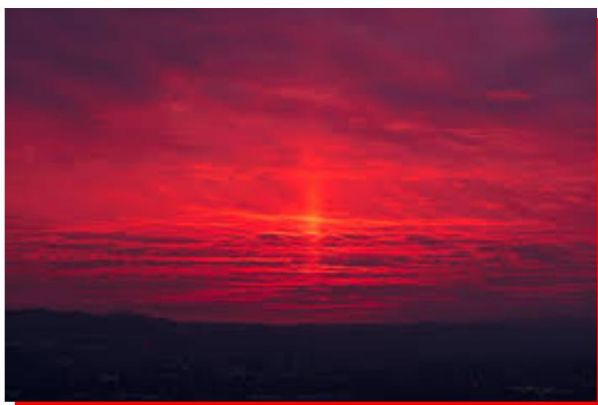


Рисунок 15.4 – Применение функции `drop-shadow()` к прямоугольному изображению

С помощью `drop-shadow()` можно создать тень по форме самого изображения (рисунок 15.5). Функция учитывает альфа-канал картинки и способна отбрасывать не только прямоугольную тень, как в случае с `box-shadow`.



Рисунок 15.5 – Применение функции `drop-shadow()` к изображению .png

5. Функция `grayscale()`

Делает изображение чёрно-белым. В скобках можно указать значение от 0% до 100%. Значение 100% сделает изображение полностью чёрно-белым. Значение 0% вернёт изображению исходные цвета. Значением может быть целое или дробное число без единиц измерения (листинг 15.5 и рисунок 15.6).

Листинг 15.5 – Функция `grayscale()`

```
img {  
  filter: grayscale(80%);  
}
```

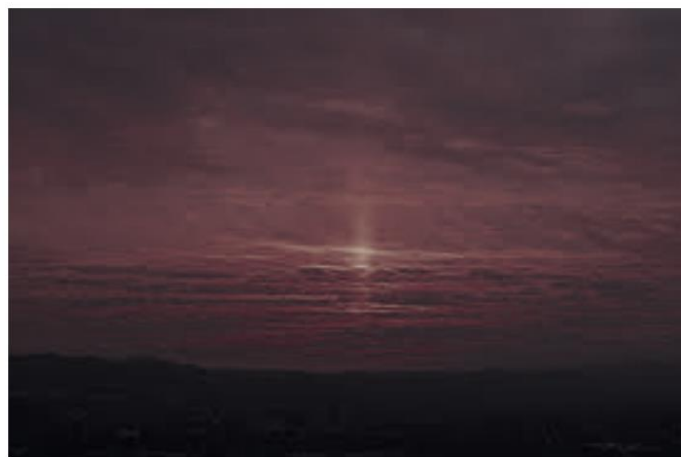


Рисунок 15.6 – Применение функции `grayscale()` к изображению

6. Функция `hue-rotate()`

Меняет цвета изображения за счёт поворота цветового круга. Угол поворота указывается в скобках функции. Можно указывать угол в градусах `deg` или в поворотах `turn` (листинг 15.6 и рисунок 15.7).

Листинг 15.6 – Функция `hue-rotate()`

```
img {  
  filter: hue-rotate(0.5turn);  
}
```



Рисунок 15.7 – Применение функции `hue-rotate()` к изображению

7. Функция `invert()`

Инвертирует цвета изображения, как бы выворачивает их, превращая в противоположные. В результате получается что-то вроде негатива. Можно указать процент инверсии от 0% до 100%. При 100% цвета на картинке полностью инвертированы. Отрицательные значения или значения больше 100% не допускаются (листинг 15.7 и рисунок 15.8).

Листинг 15.7 – Функция `invert()`

```
img {  
  filter: invert(100%);  
}
```



Рисунок 15.8 – Применение функции `invert()` к изображению

8. Функция `opacity()`

Меняет прозрачность изображения. Можно указать процент прозрачности от 0% до 100%. 0% делает картинку полностью прозрачной. 100% не меняет прозрачность изображения. Отрицательные значения или значения больше 100% не допускаются (листинг 15.8 и рисунок 15.9).

Очень похоже на работу свойства `opacity` с той разницей, что для фильтра браузер, как правило, применяет аппаратное ускорение для улучшения производительности.

Листинг 15.8 – Функция `opacity()`

```
img {  
  filter: opacity(40%);  
}
```

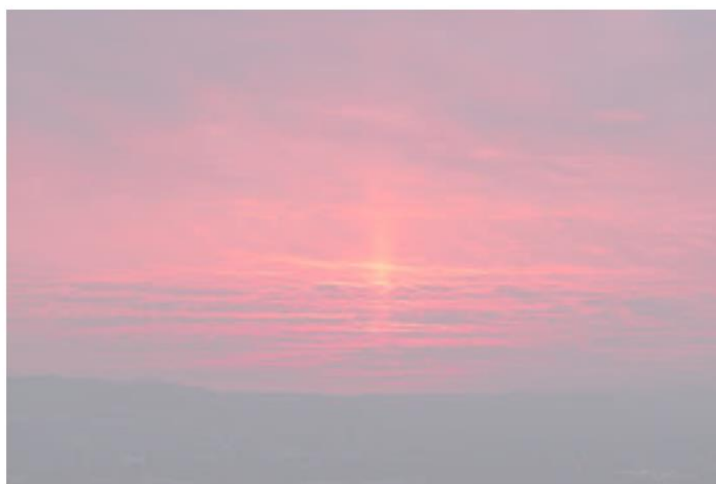


Рисунок 15.9 – Применение функции `opacity()` к изображению

9. Функция `saturate()`

Меняет насыщенность цветов изображения. Значение 0% полностью убирает насыщенность цветов. Значение 100% не изменяет исходное изображение. Допускаются значения больше 100% что приводит к перенасыщенности. Нельзя указать отрицательное значение (листинг 15.9 и рисунок 15.10).

Листинг 15.9 – Функция `saturate()`

```
img {  
  filter: saturate(390%);  
}
```



Рисунок 15.10 – Применение функции `saturate()` к изображению

10. Функция `sepia()`

Меняет цвета изображения на сепию — коричневые оттенки. Значение 100% полностью преобразует изображение в сепию. Значение 0% не изменяет исходное изображение. Отрицательные значения или значения больше 100% не допускаются. Можно использовать целое или дробное число без единиц измерения в качестве значения (листинг 15.10 и рисунок 15.11).

Листинг 15.10 – Функция `sepia()`

```
img {  
  filter: sepia(0.6);  
}
```



Рисунок 15.11 – Применение функции `sepia()` к изображению

16 Свойство `backdrop-filter`

Свойство `backdrop-filter` позволяет применить фильтры типа размытия, изменения контраста или обесцвечивания к фоновому изображению, находящемуся за элементом. Поскольку оно действует по отношению ко всему, что находится позади элемента, его необходимо сделать, по крайней мере, частично прозрачным.

Синтаксис (листинги 16.1 и 16.2):

Листинг 16.1 – Синтаксис свойства `backdrop-filter`

```
.element {  
  backdrop-filter: filter-function;  
}
```

Листинг 16.2 – Синтаксис свойства `backdrop-filter`

```
.element {  
  backdrop-filter: none;  
}
```

Можно использовать любую (или сразу несколько функций через пробел) из нижепредставленных:

- `blur()`
- `brightness()`
- `contrast()`
- `drop-shadow()`
- `grayscale()`
- `hue-rotate()`
- `invert()`
- `opacity()`
- `saturate()`
- `sepia()`
- `url()` – для применения SVG-фильтров

Эффект glassmorphism

Glassmorphism — новая тенденция, которая активно набирает популярность на таких сервисах, как Dribbble и Behance.

Его характерными чертами являются:

- прозрачность (эффект матового стекла);
- яркие или пастельные цвета;
- светлые границы элементов.

Для создания эффекта стекломорфизма следует использовать `backdrop-filter: blur()` (листинги 16.3, 16.4 и рисунок 16.1).

Листинг 16.3 – HTML-код для создания эффекта glassmorphism

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Glassmorphism</title>
  <link href="https://fonts.googleapis.com/css2?family=Inter:wght@300;400;500;600;700&display=swap" rel="stylesheet">
</head>
<body>
  
  <div class="card">
    <h3 class="card-title">Glassmorphism is awesome</h3>
    <p>Glassmorphism — это стиль дизайна для соединения и объединения всех видов использования эффекта «матового стекла» в пользовательском интерфейсе.</p>
    <a href="#">Read more</a>
  </div>
</body>
</html>
<style>
```

Листинг 16.4 – CSS-код для создания эффекта glassmorphism

```
<style>

body {
  padding: 4.5rem;
  margin: 0;
  background: linear-gradient(90deg, #60efff 0, #00ff87 62% );
  font-family: 'Inter', sans-serif;
}

.card {
  width: 400px;
  height: auto;
  padding: 2rem;
  border-radius: 1rem;
  background: rgba(255, 255, 255, .7);
  -webkit-backdrop-filter: blur(10px);
  backdrop-filter: blur(10px);
}

.card-title {
  margin-top: 0;
  margin-bottom: .5rem;
  font-size: 1.2rem;
}

p, a {
  font-size: 1rem;
}

a {
  color: #4d4ae8;
  text-decoration: none;
}

.image {
  position: absolute;
  width: 150px;
  top: .5rem;
  left: .5rem;
}

</style>
```

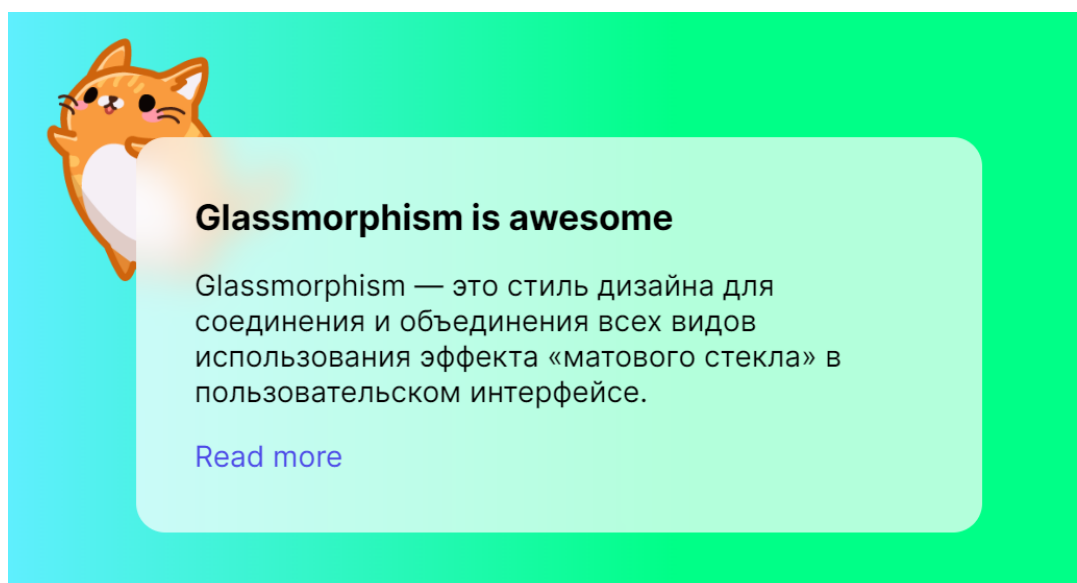


Рисунок 16.1 – Эффект glassmorphism

17 Свойство `cursor`

С помощью этого свойства можно указать, какой внешний вид будет у курсора, когда пользователь наведёт мышку на элемент.

Основные значения (более популярные) свойства `cursor`:

- `auto` – браузер сам решит какой курсор показывать в зависимости от того, на какой элемент он наведён. Например, для ссылки это будет рука с пальчиком, для поля ввода чёточка, а для обычного блока — стандартная стрелка.
- `default` – обычная, привычная стрелочка.
- `none` – курсора совсем нет.
- `pointer` – курсор, который обычно появляется над нажимаемыми элементами типа ссылок.
- `text` – курсор в виде чёточки с засечками сверху и снизу. Обычно показывается там, где текст может быть выбран, выделен.

Остальные, менее популярные значения свойства `cursor`:

- `context-menu` – курсор контекстного меню (обычно вызывается правой кнопкой мыши).
- `help` – доступен вспомогательный информационный контент.
- `progress` – программа в фоне выполняет какие-то действия, но пользователь всё ещё может с ней взаимодействовать.
- `wait` – программа не отвечает, занята обработкой какой-то операции.
- `cell` – можно выбрать одну или несколько ячеек таблицы.
- `crosshair` – курсор-крестик, обычно используется, чтобы показать, что на изображении можно выбрать какую-то область.
- `vertical-text` – практически как `text`, но вертикально.
- `alias` – загнутая стрелочка, так обозначают ссылки, клик по которым уведёт с текущего сайта.
- `copy` – содержимое можно скопировать.
- `move` – содержимое можно подвигать.
- `no-drop` – в эту область нельзя перетащить файл.
- `not-allowed` – действие не будет выполнено.
- `grab` – содержимое можно схватить, чтобы перетащить.

- `grabbing` – содержимое было схвачено для перетаскивания.
- `all-scroll` – содержимое может быть проскроллено в любом направлении.
- `col-resize` – колонку / ячейку таблицы можно изменить в размерах по горизонтали.
- `row-resize` – строку в таблице или другой элемент можно изменить в размерах по вертикали.
- `zoom-in` – содержимое можно приблизить, увеличить.
- `zoom-out` – содержимое можно отдалить, уменьшить.

Кроме значений, заданных при помощи ключевых слов, можно указывать ссылку на картинку, которая будет показана вместо курсора (листинг 17.1).

Если вы решили сделать кастомный курсор, то обязательно укажите через запятую одно из стандартных значений. Это нужно на случай, если браузер не смог загрузить или отобразить картинку. Тогда будет показан тот курсор, который вы указали в конце значения.

Можно указывать несколько картинок подряд через запятую (`url(cat.svg), url(cat.png), ...`), тогда будет показана первая из доступных.

Листинг 17.1 – Установка курсора в виде картинки

```
article {
  cursor: url(cat.png), auto;
}
```

Наглядное представление всех значений свойства `cursor` можно посмотреть по ссылке [Cursor Property Cheatsheet]: <https://codepen.io/GuillaumeChabot/pen/zrJbvM>

Также ниже представлен рисунок с некоторыми значениями свойства `cursor` (рисунок 17):

CSS СВОЙСТВО **CURSOR**

определяет тип
отображаемого курсора

















default 	help 	pointer 	move 
wait 	progress 	not-allowed 	crosshair 
ew-resize 	ns-resize 	nw-resize 	sw-resize 
text 	vertical-text 	cell 	copy 

Рисунок 17 – Некоторые значения свойства `cursor`

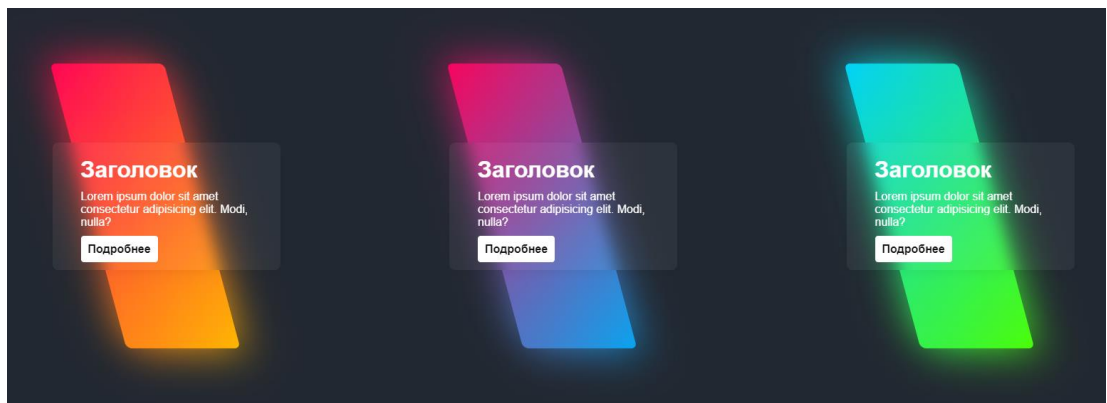
Практическое задание

Примечание. В каждом элементе должны быть использованы CSS-переменные и адаптивность (медиа-запросы, свойство `flex-wrap` или относительные единицы измерения: `%`, `rem`, `em`, `vw`, `vh` и т.д.). Контент страницы всегда должен пропорционально уместиться на экране устройств с шириной от 320px до 2560px. Применить функцию `var()` в одном или нескольких заданиях (не нагружая сильно страницу).

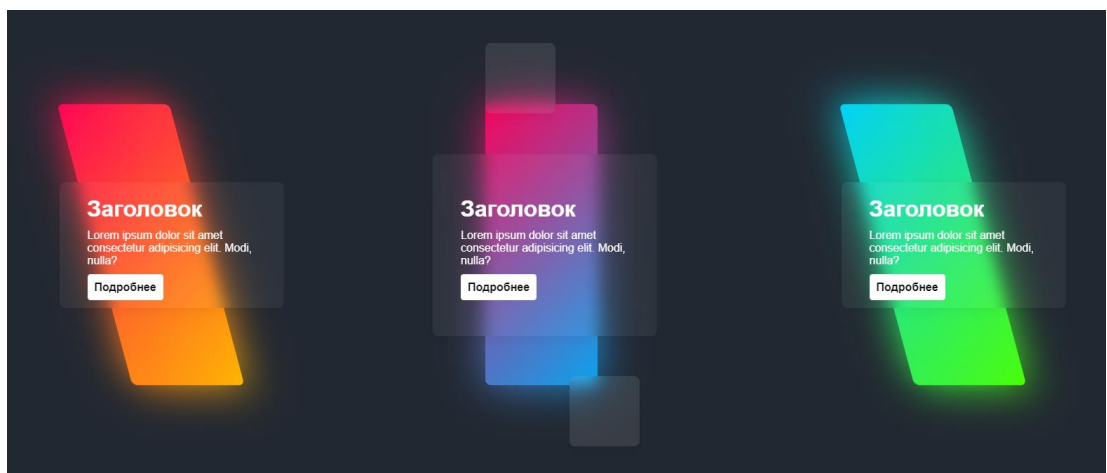
1. Создать три карточки по тематике своего сайта в стиле glassmorphism с градиентным элементом сзади. Добавить плавную покадровую анимацию (`@keyframes`) при наведении, как на фото ниже. Меньшие по размеру квадраты должны перемещаться.

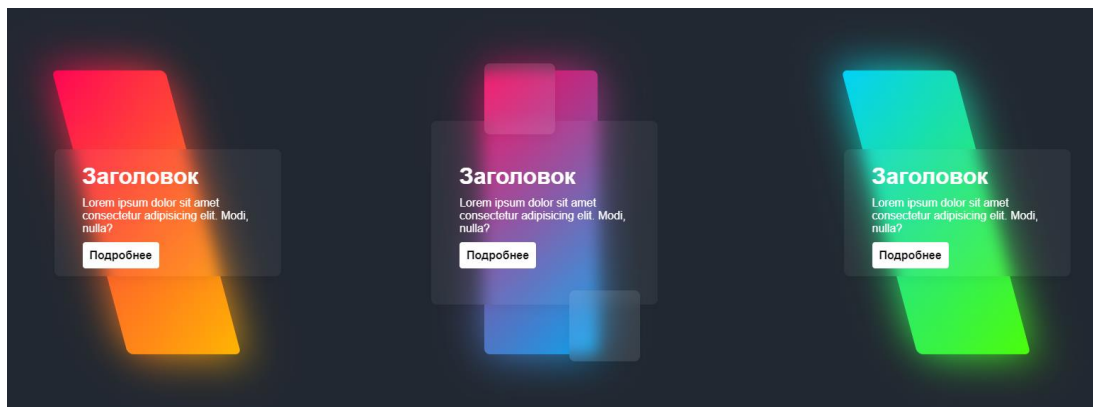
Обязательно: использовать псевдоэлементы `::before` и `::after`.

До наведения:



После наведения (включается покадровая анимация `@keyframes`):

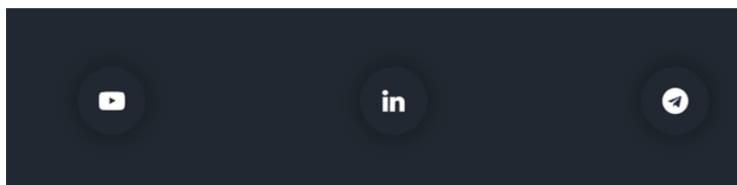




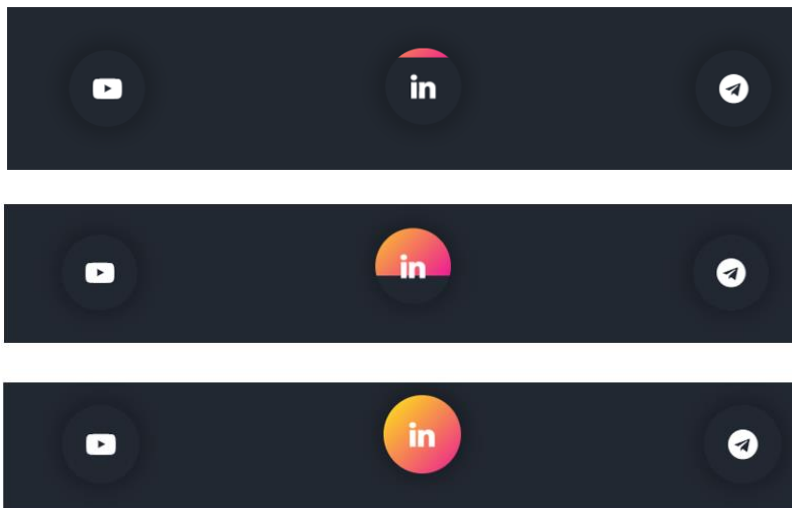
2. Создать 3 кнопки социальных сетей (например, в футере) с плавной анимацией: при наведении сверху должен появляться градиент и закрывать весь предыдущий задний фон. **Обязательно:** использовать псевдоэлемент(-ы) и применить свойство `cursor` при наведении на кнопки.

Пример:

До наведения:



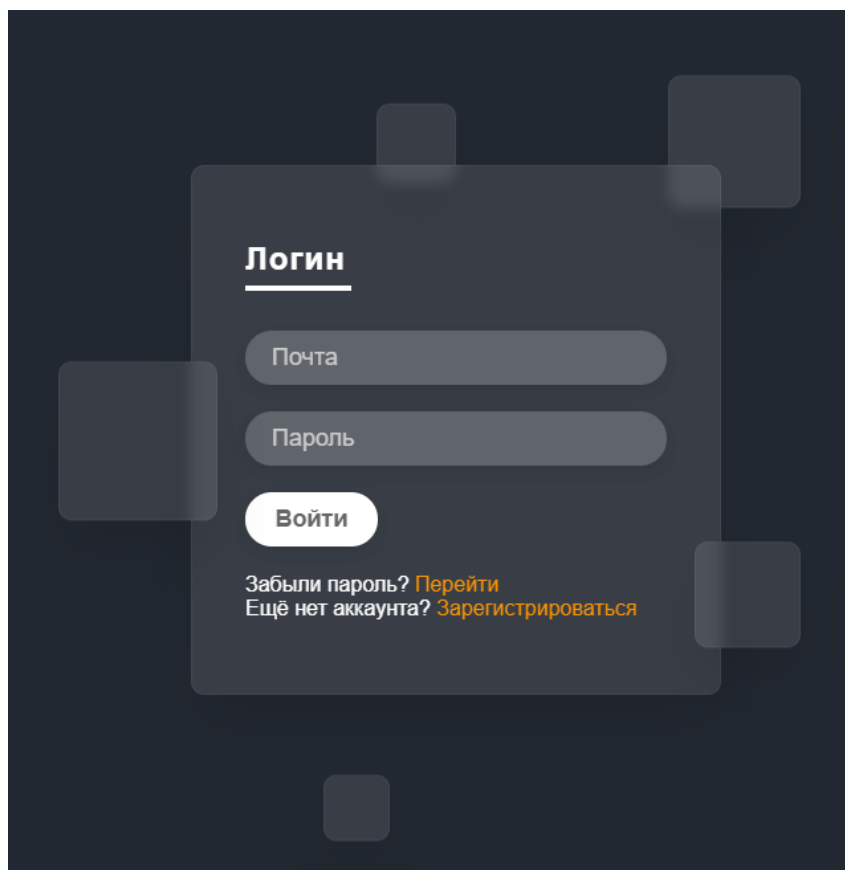
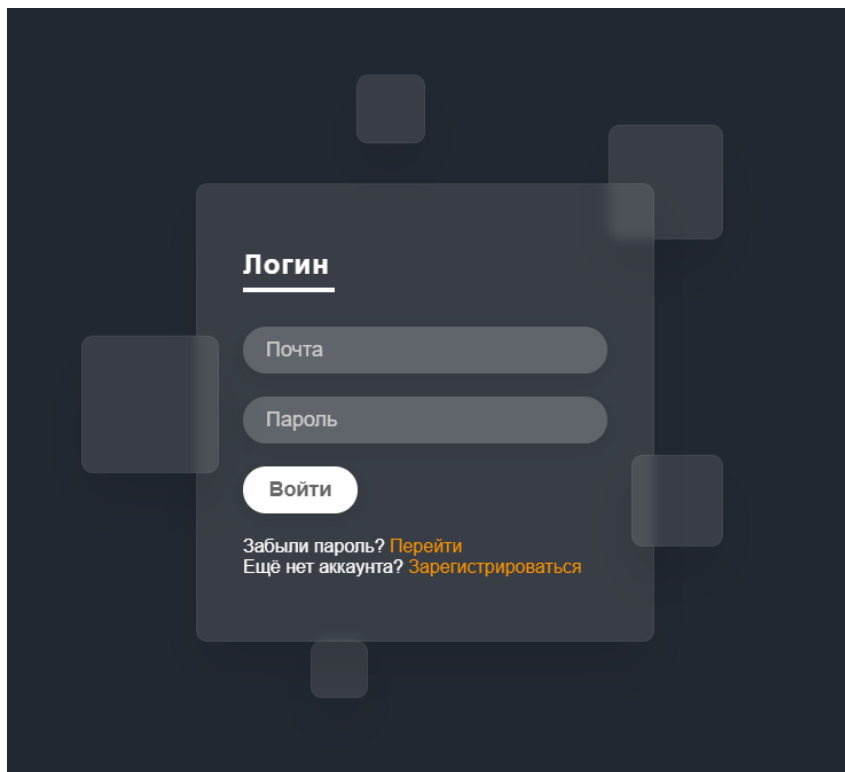
После наведения:



3. Создать форму регистрации (или доработать те формы, которые вы создавали на одной из предыдущих практических работ) в стиле glassmorphism с плавной покадровой анимацией. Меньшие по размеру квадраты должны быть анимированы и перемещаться в случайном порядке (используйте переменные и функцию `calc()`).

Обязательно: использовать псевдоэлемент(-ы), функции свойства `transform`, функцию `calc()`, переменные.

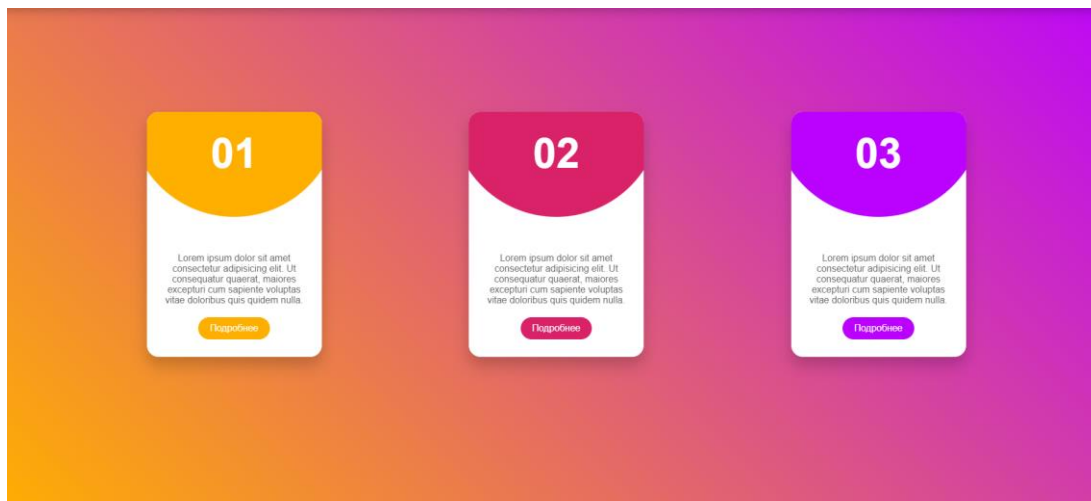
Пример:



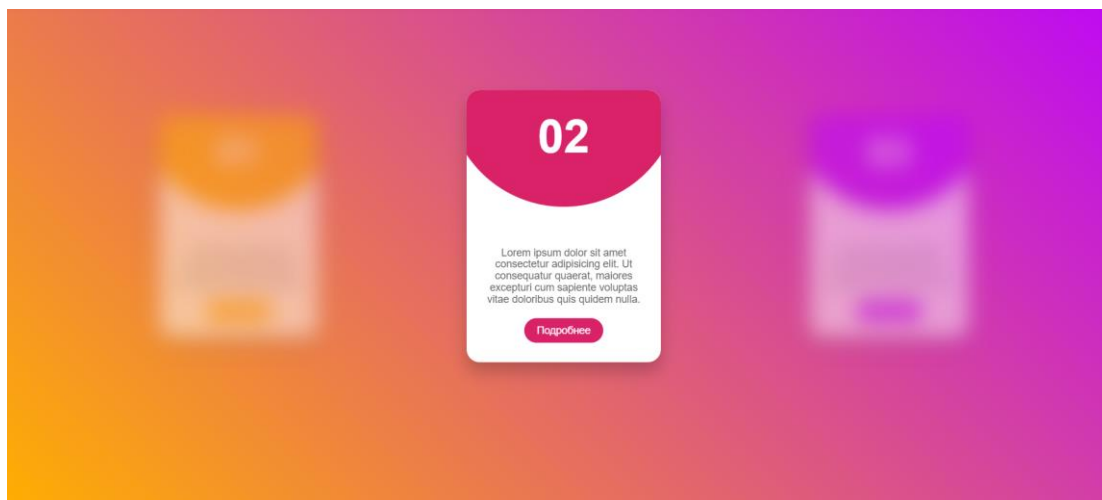
4. Создать три карточки по тематике своего сайта. Использовать свойства модуля flexbox, свойства `transition`, `transform` и `filter`. Добавить плавную анимацию при наведении.

Пример:

До наведения:



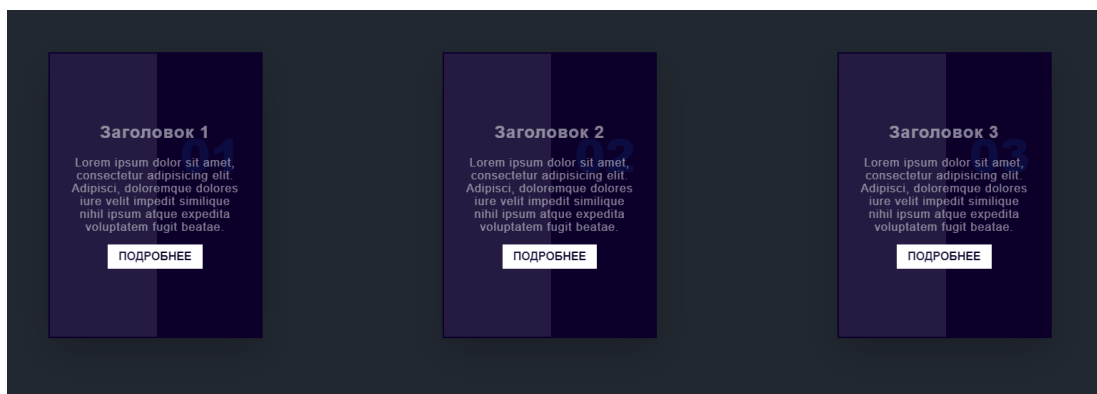
После наведения:



5. Создать три карточки по тематике своего сайта. Использовать свойства `transition`, `transform` и псевдоэлемент `::before`. Добавить плавную покадровую анимацию при наведении, как на фото ниже: выбранная карточка становится более яркой и вдоль её границ "протекают" лучи.

Пример:

До наведения:



После наведения:

