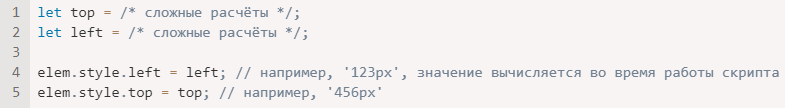
Как правило, существует два способа задания стилей для элемента:

1. Создать класс в CSS и использовать его: <div class="...">
2. Писать стили непосредственно в атрибуте style: <div style="...">.

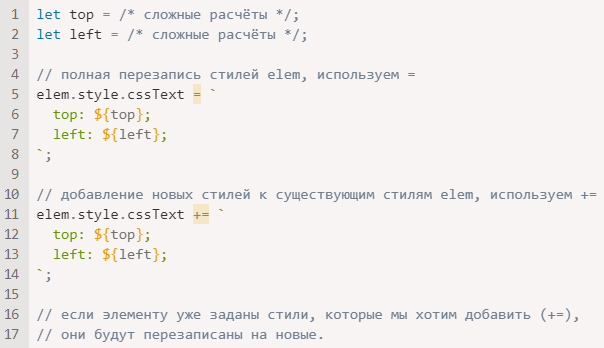
JavaScript может менять и классы, и свойство style.

Классы – всегда предпочтительный вариант по сравнению со style. Мы должны манипулировать свойством style только в том случае, если классы «не могут справиться».

Например, использование style является приемлемым, если мы вычисляем координаты элемента динамически и хотим установить их из JavaScript:



Мы также можем установить сразу несколько стилей для элемента, используя свойство cssText. Но с ним нужно быть осторожнее, т.к оно может как добавить стили к существующим, так и полностью перезаписать их на новые:



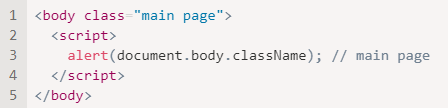
В других случаях, например, чтобы сделать текст красным, добавить значок фона – описываем это в CSS и добавляем класс (JavaScript может это сделать). Это более гибкое и лёгкое в поддержке решение.

**[className и classList](https://learn.javascript.ru/styles-and-classes" \l "classname-i-classlist)**

Изменение класса является одним из наиболее часто используемых действий в скриптах.

Когда-то давно в JavaScript существовало ограничение: зарезервированное слово типа "class" не могло быть свойством объекта. Это ограничение сейчас отсутствует, но в то время было невозможно иметь свойство elem.class.

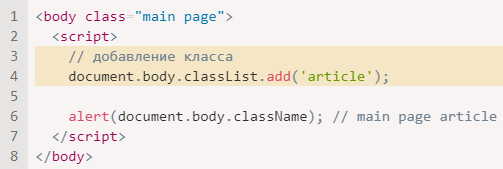
Поэтому для классов было введено схожее свойство "className": elem.className соответствует атрибуту "class".



Если мы присваиваем что-то elem.className, то это заменяет всю строку с классами. Иногда это то, что нам нужно, но часто мы хотим добавить/удалить один класс.

Для этого есть другое свойство: elem.classList.

elem.classList – это специальный объект с методами для добавления/удаления одного класса.

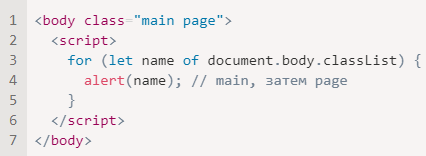


Так что мы можем работать как со строкой полного класса, используя className, так и с отдельными классами, используя classList. Выбираем тот вариант, который нам удобнее.

Методы classList:

* elem.classList.add/remove("class") – добавить/удалить класс.
* elem.classList.toggle("class") – добавить класс, если его нет, иначе удалить.
* elem.classList.contains("class") – проверка наличия класса, возвращает true/false.

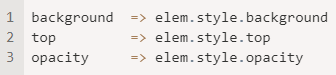
Кроме того, classList является перебираемым, поэтому можно перечислить все классы при помощи for..of:



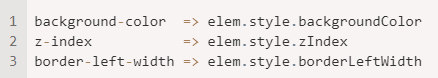
## [Свойство style](https://learn.javascript.ru/styles-and-classes" \l "svoystvo-style)

Свойство elem.style – это объект, который соответствует тому, что написано в атрибуте "style".

Свойства из одного слова записываются так же, с маленькой буквы:



Для свойств из нескольких слов используется [camelCase](https://ru.wikipedia.org/wiki/CamelCase):



**Свойства с префиксом**

Стили с браузерным префиксом, например, -moz-border-radius, -webkit-border-radius преобразуются по тому же принципу: дефис означает заглавную букву.

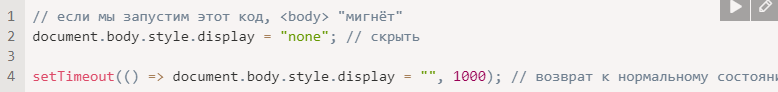


## [Сброс стилей](https://learn.javascript.ru/styles-and-classes" \l "sbros-stiley)

Иногда нам нужно добавить свойство стиля, а потом, позже, убрать его.

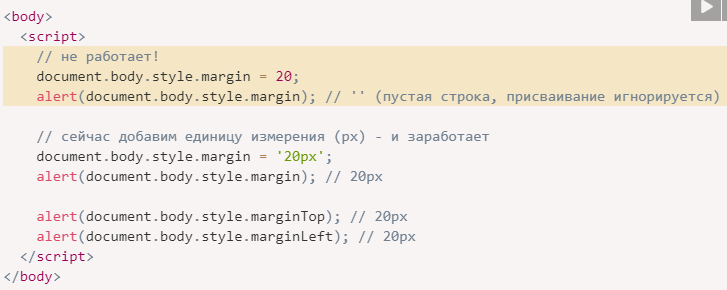
Например, чтобы скрыть элемент, мы можем задать elem.style.display = "none".

Затем мы можем удалить свойство style.display, чтобы вернуться к первоначальному состоянию. Вместо delete elem.style.display мы должны присвоить ему пустую строку: elem.style.display = "".



Если мы установим в style.display пустую строку, то браузер применит CSS-классы и встроенные стили, как если бы такого свойства style.display вообще не было.

## [Следите за единицами измерения](https://learn.javascript.ru/styles-and-classes" \l "sledite-za-edinitsami-izmereniya)



Пожалуйста, обратите внимание, браузер «распаковывает» свойство style.margin в последних строках и выводит style.marginLeft и style.marginTop из него.

## [Вычисленные стили: getComputedStyle](https://learn.javascript.ru/styles-and-classes" \l "vychislennye-stili-getcomputedstyle)

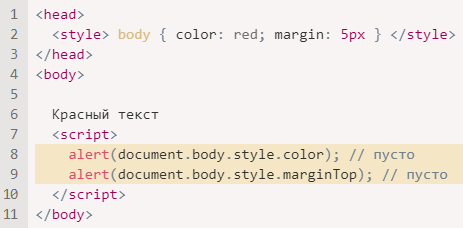
Итак, изменить стиль очень просто. Но как его прочитать?

Например, мы хотим знать размер, отступы, цвет элемента. Как это сделать?

**Свойство style оперирует только значением атрибута "style", без учёта CSS-каскада.**

Поэтому, используя elem.style, мы не можем прочитать ничего, что приходит из классов CSS.

Например, здесь style не может видеть отступы:



…Но что, если нам нужно, скажем, увеличить отступ на 20px? Для начала нужно его текущее значение получить.

Для этого есть метод: getComputedStyle.

Синтаксис:



**element**

Элемент, значения для которого нужно получить

**pseudo**

Указывается, если нужен стиль псевдоэлемента, например ::before. Пустая строка или отсутствие аргумента означают сам элемент.

Результат вызова – объект со стилями, похожий на elem.style, но с учётом всех CSS-классов.

Например:



**Вычисленное (computed) и окончательное (resolved) значения**

Есть две концепции в [CSS](https://drafts.csswg.org/cssom/#resolved-values):

1. *Вычисленное* (computed) значение – это то, которое получено после применения всех CSS-правил и CSS-наследования. Например, height:1em или font-size:125%.
2. *Окончательное* ([resolved](https://drafts.csswg.org/cssom/" \l "resolved-values)) значение – непосредственно применяемое к элементу. Значения 1em или 125% являются относительными. Браузер берёт вычисленное значение и делает все единицы измерения фиксированными и абсолютными, например, height:20px или font-size:16px. Для геометрических свойств разрешённые значения могут иметь плавающую точку, например, width:50.5px.

Давным-давно getComputedStyle был создан для получения вычисленных значений, но оказалось, что окончательные значения гораздо удобнее, и стандарт изменился.

Так что, в настоящее время getComputedStyle фактически возвращает окончательное значение свойства, для геометрии оно обычно в пикселях.

**getComputedStyle требует полное свойство!**

Для правильного получения значения нужно указать точное свойство. Например: paddingLeft, marginTop, borderTopWidth. При обращении к сокращённому: padding, margin, border – правильный результат не гарантируется.

Например, если есть свойства paddingLeft/paddingTop, то что мы получим вызывая getComputedStyle(elem).padding? Ничего, или, может быть, «сгенерированное» значение из известных внутренних отступов? Стандарта для этого нет.

**Стили, применяемые к посещённым :visited ссылкам, скрываются!**

Посещённые ссылки могут быть окрашены с помощью псевдокласса :visited.

Но getComputedStyle не даёт доступ к этой информации, чтобы произвольная страница не могла определить, посещал ли пользователь ту или иную ссылку, проверив стили.

JavaScript не видит стили, применяемые с помощью :visited. Кроме того, в CSS есть ограничение, которое запрещает в целях безопасности применять к :visited CSS-стили, изменяющие геометрию элемента. Это гарантирует, что нет обходного пути для «злой» страницы проверить, была ли ссылка посещена и, следовательно, нарушить конфиденциальность.

## [Итого](https://learn.javascript.ru/styles-and-classes" \l "itogo)

Для управления классами существуют два DOM-свойства:

* className – строковое значение, удобно для управления всем набором классов.
* classList – объект с методами add/remove/toggle/contains, удобно для управления отдельными классами.

Чтобы изменить стили:

* Свойство style является объектом со стилями в формате camelCase. Чтение и запись в него работают так же, как изменение соответствующих свойств в атрибуте "style". Чтобы узнать, как добавить в него important и делать некоторые другие редкие вещи – смотрите [документацию](https://developer.mozilla.org/ru/docs/Web/API/CSSStyleDeclaration).
* Свойство style.cssText соответствует всему атрибуту "style", полной строке стилей.

Для чтения окончательных стилей (с учётом всех классов, после применения CSS и вычисления окончательных значений) используется:

* Метод getComputedStyle(elem, [pseudo]) возвращает объект, похожий по формату на style. Только для чтения.