Стили и классы

До того, как начнёте изучать способы работы со стилями и классами в JavaScript, есть одно важное правило. Надеемся, это достаточно очевидно, но мы все равно должны об этом упомянуть.

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:

```
let top = /* сложные расчёты */;
let left = /* сложные расчёты */;

elem.style.left = left; // например, '123px', значение вычисляется во время работы скрипта
elem.style.top = top; // например, '456px'
```

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

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"`.

Например:

```
<body class="main page">
  <script>
    alert(document.body.className); // main page
  </script>
</body>
```

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.

Например:

```
<body class="main page">
  <script>
    // добавление класса
    document.body.classList.add('article');

    alert(document.body.className); // main page article
  </script>
</body>
```

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`:

```
<body class="main page">
  <script>
    for (let name of document.body.classList) {
      alert(name); // main, затем page
    }
  </script>
</body>
```

Element style

Свойство `elem.style` – это объект, который соответствует тому, что написано в атрибуте `"style"`. Установка стиля `elem.style.width="100px"` работает так же, как наличие в атрибуте `style` строки `width:100px`.

Для свойства из нескольких слов используется camelCase:

```
background-color  => elem.style.backgroundColor
z-index           => elem.style.zIndex
border-left-width => elem.style.borderLeftWidth
```

Например:

```
document.body.style.backgroundColor = prompt('background color?', 'green');
```

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

Например:

```
button.style.MozBorderRadius = '5px';
button.style.WebkitBorderRadius = '5px';
```

Сброс стилей

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 запустим этот код, <body> "мигнёт"
document.body.style.display = "none"; // скрыть

setTimeout(() => document.body.style.display = "", 1000); // возврат к
нормальному состоянию
```

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.

Полная перезапись `style.cssText`
Обычно мы используем `style.*` для присвоения индивидуальных свойств стиля. Нельзя установить список стилей как, например, `div.style="color: red; width: 100px"`, потому что `div.style` – это объект, и он доступен только для чтения.

Для задания нескольких стилей в одной строке используется специальное свойство `style.cssText`:

```
<div id="div">Button</div>

<script>
  // можем даже устанавливать специальные флаги для стилей, например,
"important"
  div.style.cssText=`color: red !important;
    background-color: yellow;
    width: 100px;
    text-align: center;
  `;

  alert(div.style.cssText);
</script>
```

Это свойство редко используется, потому что такое присваивание удаляет все существующие стили: оно не добавляет, а заменяет их. Можно ненароком удалить что-то нужное. Но его можно использовать, к примеру, для новых элементов, когда мы точно знаем, что не удалим существующий стиль.

То же самое можно сделать установкой атрибута: `div.setAttribute('style', 'color: red...')`.

Следите за единицами измерения

Не забудьте добавить к значениям единицы измерения.

Например, мы должны устанавливать `10px`, а не просто `10` в свойство `elem.style.top`. Иначе это не сработает:

```
<body>
  <script>
    // не работает!
    document.body.style.margin = 20;
    alert(document.body.style.margin); // '' (пустая строка, присваивание игнорируется)

    // сейчас добавим единицу измерения (px) - и заработает
    document.body.style.margin = '20px';
    alert(document.body.style.margin); // 20px

    alert(document.body.style.marginTop); // 20px
    alert(document.body.style.marginLeft); // 20px
  </script>
</body>
```

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

Вычисленные стили: 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:

```
<head>
  <style> body { color: red; margin: 5px } </style>
</head>
<body>

  Красный текст
  <script>
    alert(document.body.style.color); // пусто
    alert(document.body.style.marginTop); // пусто
  </script>
</body>
```

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

Для этого есть метод: `getComputedStyle`.

Синтаксис:

```
getComputedStyle(element, [pseudo])
```

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:

```
<head>
  <style> body { color: red; margin: 5px } </style>
</head>
<body>

  <script>
    let computedStyle = getComputedStyle(document.body);

    // сейчас мы можем прочитать отступ и цвет

    alert( computedStyle.marginTop ); // 5px
    alert( computedStyle.color ); // rgb(255, 0, 0)
  </script>

</body>
```

Вычисленное (computed) и окончательное (resolved) значения
Есть две концепции в CSS:

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) 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.

Есть и другие несоответствия. Например, некоторые браузеры (Chrome) отображают `10px` в документе ниже, а некоторые (Firefox) – нет:

```
<style>
  body {
    margin: 10px;
  }
</style>
<script>
  let style = getComputedStyle(document.body);
  alert(style.margin); // пустая строка в Firefox
</script>
```

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

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.
- 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.

Задачи

Создать уведомление

важность: 5

Напишите функцию `showNotification(options)`, которая создаёт уведомление: `<div class="notification">` с заданным содержимым. Уведомление должно автоматически исчезнуть через 1,5 секунды.

Пример объекта `options`:

```
// показывает элемент с текстом "Hello" рядом с правой верхней частью окна.
showNotification({
  top: 10, // 10px от верхней границы окна (по умолчанию 0px)
  right: 10, // 10px от правого края окна (по умолчанию 0px)
  html: "Hello!", // HTML-уведомление
  className: "welcome" // дополнительный класс для div (необязательно)
});
```

Демо в новом окне

Используйте CSS-позиционирование для отображения элемента в заданных координатах. Исходный документ имеет необходимые стили.