Торшин Роман – Старший FE разработчик

КАЧЕСТВО КОДА

ESLINT

Качество кода – что это?

Основные принципы

- KISS. KISS расшифровывается как «чем проще — тем лучше»
- DRY. DRY означает «не повторяйся»
- Для самостоятельного изучения: SOLID

Способы контроля качества

- Style Guide (стайл гайд), стандарт (ES6 / ECMAScript 2015)
- Автоматическая проверка – ESLINT
- Code Review

Style Guide (стайл гайд)

- Описание принципов оформления и написания кода, синтаксиса. В компании есть такие для разных языков программирования

- HTML / Twig
- SCSS
- JS
- ...

HTML/Twig

Не используйте атрибут style для описания стилей элемента - это не подходит для повторного использования, а также добавляет ряд ограничений, связанных со специфичностью css-правил. Используйте классы.

```
<!-- Плохо -->
<span style="font-size: 11px;">Текст мелким шрифтом</span>

<!-- Хорошо -->
<span class="text-desc">Текст мелким шрифтом</span>
```

Если для элемента ввода указывается сопроводительный текст, то он должн быть оформлен с использованием тега <label> - в этом случае при нажатии на текст будет происходить фокус на элементе ввода.

```
<!-- Плохо -->
<input type="radio" name="male" id="male1"> <span>М</span>
<input type="radio" name="male" id="male2"> <span>Ж</span>

<!-- Хорошо -->
<input type="radio" name="male" id="male1"> <label for="male1">М</label>
<input type="radio" name="male" id="male2"> <label for="male2">Ж</label>

<!-- Хорошо -->
<label><input type="radio" name="male" id="male1"> М</label>
<label><input type="radio" name="male" id="male2"> Ж</label>
```

Правила форматирования

Названия тегов и атрибутов требуется писать в нижнем регистре.

```
<!-- Плохо -->
<A HREF="http://www.livemaster.ru">Ярмарка Мастеров</A>

<!-- Хорошо -->
<a href="http://www.livemaster.ru">Ярмарка Мастеров</a>
```

Заключайте значения атрибутов только в двойные кавычки.

```
<!-- Плохо -->
<input type='submit' value='Отправить'>

<!-- Хорошо -->
<input type="submit" value="Отправить">
```

Идентификаторы (id) элементов требуется писать в нижнем регистре, для разделения слов использовать дефис.

```
<!-- Плохо -->
<ul id="mainMenu"></ul>

<!-- Хорошо -->
<ul id="main-menu"></ul>
```

Используйте для отступов табуляцию размером в 4 символа, а не пробелы.

```
<!-- Плохо -->
<ul>
  <li></li>
  <li></li>
</ul>

<!-- Хорошо -->
<ul>
    <li></li>
    <li></li>
</il>
```

Вложенные элементы следует переносить на новую строку с отступом относительно родительского элемента.

```
<!-- Плохо -->
<ul class="menu"><li class="active"><a href="//www.livemaster.ru">Главная</a></li><li><a href="//www

<!-- Хорошо -->
<ul class="menu">
    <li class="active">
        <a href="http://www.livemaster.ru">Главная</a>
    </li>
    <li>
        <a href="http://www.livemaster.ru/sectionlist.php">Каталог</a>
    </li>
</ul>
```

ⓘ Допускается не переносить строчные элементы, если они вложены друг в друга (например, <i> внутри <a>).

Для тега <img> нужен alt="", даже если alt пустой

```
<!-- Плохо -->
<img src="/img/cart.svg">


<!-- Хорошо -->
<img src="/img/cart.svg" alt="Корзина">
```

Для комментариев нужно использовать конструкцию {# text #}. Комментарии вида <!-- --> недопустимы

```
<!-- Плохой комментарий, т.к. такие комментарии не вырезаются и попадают в код для клиента. -->

{# Правильный комментарий, т.к. такие комментарии вырезаются и не попадают в код для клиента. #}
```

Для всех ссылок необходимо использовать относительные пути.

```
<!-- Плохо -->
<div class="block">
    <a class="js-click-p2p-item" href="https://www.livemaster.ru/myshop"> </a>
    <img src="https://www.livemaster.ru/image/responsive/pages/feedback/feedbacks-bg.jpg"/>
</div>

<!-- Хорошо -->
<div class="block">
    <a class="js-click-p2p-item" href="myshop"> </a>
    <img src="/image/responsive/pages/feedback/feedbacks-bg.jpg"/>
</div>
```

TWIG v.1.X

Используйте нативную конкатенацию Twig

```
<!-- Плохо -->
<div class="{{ adaptiveItemPrefix }}__cover">
    <a class="js-click-p2p-item" href="/item/{{ item.urlname }}" title="">
        <div class="{{ adaptiveItemPrefix }}__responsive-box">
            <div class="{{ adaptiveItemPrefix }} __content" ></div>
        </div>
    </a>
</div>

<!-- правильно -->
<div class="{{ adaptiveItemPrefix ~ "__cover" }}">
    <a class="js-click-p2p-item" href="{{ '/item/' ~ item.urlname }}" title="">
        <div class="{{ adaptiveItemPrefix ~ "__responsive-box"}}">
            <div class="{{ adaptiveItemPrefix ~ "__content" }}"></div>
        </div>
    </a>
</div>
```

Если в цикле twig вам необходим счётчик цикла, нужно использовать встроенный счётчик иттераций цикла loop.index (начинается с 1) или loop.index0 (начинается с 0)

```twig
<!-- Плохо -->
{% set  iterationCount = 1 %}
{% for user in contest %}
    {% if iterationCount == 1 %}
        <div class="banner"></div>
    {% endif %}

    <span class="user-name" data-id="{{ iterationCount }}">
        {{ user.name }}
    </span>

    {% set  iterationCount = iterationCount + 1 %}
{% endfor %}


<!-- Хорошо -->
{% for user in contest %}
    {% if loop.index == 1 %}
        <div class="banner"></div>
    {% endif %}

    <span class="user-name" data-id="{{ loop.index }}">
        {{ user.name }}
    </span>
{% endfor %}
```

Если вам нужно создать цикл с количеством иттераций равный n, можно воспользоваться конструкцией {% for i in 1..n %}

```
{% for i in 1..24 %}
    <div class="js-item-{{ i }}"></div>
{% endfor %}


{% for i in 1..users|length %}
    ...
{% endfor %}
```

При использовании вложенных конструкций Twig также должны делаться отступы - итоговый вариант должен не красиво смотреться в исходном коде браузера, а быть хорошо читаемым при разработке.

```
<!-- Плохо -->
{% if showPrice %}
<span class="price">1 999 руб</span>
{% endif %}

<!-- Хорошо -->
{% if showPrice %}
    <span class="price">1 999 руб</span>
{% endif %}
```

При комментировании текста должна использоваться конструкция вида {# ... #}

```
{# Хлебные крошки #}
<div class="row">
    <div class="col-xs-12 hidden-xs">
        {% include 'responsive/_elements/breadcrumbs/index.tpl' with { breadcrumbs: adapter.content.e
    </div>
</div>
```

Мультиязычность

Весь русскоязычный текст должен быть обернут в конструкцию {% trans %}русскоязычный текст{% endtrans %}

```
<!-- При использовании одиночной формы используется конструкция trans -->
{% trans %}Каталог{% endtrans %}

{% trans %}
    Моя Ярмарка
{% endtrans %}

<!-- При использовании множественной формы используется конструкция trans с конструкцией plural внутр
<!-- До конструкции plural указывается текст в 1й множественной форме -->
<!-- После конструкции plural указывается текст в 3й множественной форме -->
<!-- В plural указывается число, на основании которого выбирается форма, в тексте это число доступно
{% trans %}
    По запросу найдена {{ count }} работа.
{% plural itemsFound %}
    По запросу найдено {{ count }} работ.
{% endtrans %}
```

Внутри конструкции trans могут использоваться только простые переменные, массивы и объекты использовать нельзя.

Фильтры

Если необходимо обрезать массив данных или строку, можно использовать фильтр **slice**

```
{% for i in [1, 2, 3, 4, 5]|slice(1, 2) %}
    {# will iterate over 2 and 3 #}
{% endfor %}

{{ '12345'|slice(1, 2) }}

{# outputs 23 #}
```

Если необходимо, чтобы первый символ будет прописным, все остальные строчными, можно воспользоваться фильтром **capitalize**

```
{{ 'my first car'|capitalize }}

{# outputs 'My first car' #}
```

Фильтр **date** форматирует дату в заданный формат (Фильтр даты принимает строки (он должен быть в формате, поддерживаемом функцией strtotime), экземпляры DateTime или DateInterval. Например, чтобы отобразить текущую дату, отфильтруйте слово «now»)

```
{{ post.published_dt|date('d.m.Y H:i:s') }}


{{ "now"|date("m/d/Y") }}
```

Фильтр **default** возвращает переданное значение по умолчанию, если значение не определено или пусто, в противном случае значение переменной

```
{{ var|default('var is not defined') }}
```

Фильтр **length** возвращает количество элементов последовательности или длину строки.

```
{% if users|length > 10 %}
    ...
{% endif %}
```

Фильтр **url_encode** кодирует данную строку как сегмент URL или массив как строку запроса

```
{{ "path-seg*ment"|url_encode }}
{# outputs "path-seg%2Ament" #}

{{ "string with spaces"|url_encode }}
{# outputs "string%20with%20spaces" #}

{{ {'param': 'value', 'foo': 'bar'}|url_encode }}
{# outputs "param=value&foo=bar" #}
```

Фильтр **number_format** форматирует числа. Это оболочка PHP-функции number_format. Вы можете контролировать количество десятичных разрядов, десятичную точку и разделитель тысяч, используя дополнительные аргументы:

1-ый аргумент - десятичное число, количество отображаемых десятичных знаков
2-ой аргумент - символ(ы) для отделения дробной части числа
3-ий аргумент - символ(ы) для разделения тысяч

```
{{ 9800.333|number_format(2, '.', ',') }}
{# outputs "9,800.33" #}
```

SCSS

Переменные.

Переменные используются для хранения и повторного использования различной информации. Для создания переменной в Sass используется символ "$".

Основные переменные находятся в файле 'css/responsive/mixins/_variables.scss', который подключается к основному файлу со стилями строкой @import 'css/responsive/mixins/_variables';

```
$fontSizeBase: 14px;
$fontSizeSmall: 12px;

.modal__subtitle {
    font-size: $fontSizeSmall;
}
```

Вложенность.

Вложенность. Sass предоставляет удобную возможность по организации вложенности, которая удачно сочетается с используемой методологией BEM. Для ссылки на родителя используется символ "&".

```
/* Так как в CSS нет вложенности, то принадлежность одного элемента к другому можно определить только
.modal {}
.modal--open {}
.modal__window {}
.modal__mask {}

/* С помощью Sass можно организовать полноценную вложенность */
.modal {
    &--open {}
    &__window {}
    &__mask {}
}
```

Глубина вложености селекторов ограничена 4 степенями. Рекомендуется максимум 3.

```scss
// Плохо
.landing {

    &__banner {
        display: flex;
        justify-content: center;
        align-items: center;

        &-content {
            padding: 20px;

            &-title {
                width: 100%;
                font-size: 36px;
                line-height: 42px;
                margin-bottom: 12px;

            }
        }
    }
}
```

```scss

// Хорошо
.landing {

    &__banner {
        display: flex;
        justify-content: center;
        align-items: center;
    }

    &__banner-content {
        padding: 20px;
    }

    &__banner-content-title {
        width: 100%;
        font-size: 36px;
        line-height: 42px;
        margin-bottom: 12px;
    }
}
```

"Лидирующий 0" должен удаляться

```css
// Плохо
.foo {
  font-size: 0.5em;
}


// Хорошо
.foo {
  font-size: .5em;
}
```

После двоеточия(":") и запятой(",") должен ставиться пробел.

```css
// Правильно
.foo {
    content: 'bar';
    color: rgb(255, 144, 32);
}

// Не правильно
.foo {
    content:'bar';
    color: rgb(255,144,32);
}
```

Позиционирование следует первым потому, что оно влияет на положение блоков в потоке документа. Блочная модель идёт следующей, так как она определяет размеры и расположение блоков. Все остальные объявления, которые изменяют вид внутренних частей блоков и не оказывают влияния на другие блоки, идут в последнюю очередь.

```
.declaration-order {
  // Позиционирование
  position: absolute;
  top: 0;
  right: 0;
  bottom: 0;
  left: 0;
  z-index: 100;

  // Блочная модель
  display: block;
  float: right;
  width: 100px;
  height: 100px;
  margin: 10px;
  padding: 10px;

  // Типографика
  font: normal 13px/1.5 "Arial", sans-serif;
  font-style: normal;
  font-size: 13px;
  line-height: 1.5;
  font-family: "Arial", sans-serif;
  text-align: center;
  color: #333333;

  // Оформление
  background-color: #f5f5f5;
  border: 1px solid #e5e5e5;
  border-radius: 3px;
  opacity: 1;

  // Анимация
  transition: color 1s;

  // Разное
  will-change: auto;
}
```

MIXIN

Миксины.

Миксины - (часто используется название примеси) позволяют определить стили, которые могут быть использованы повторно в любом месте документа без необходимости прибегать к классам.

- Миксины также могут содержать целые CSS правила или что-либо другое, разрешённое в Sass файле.
- Они даже могут принимать аргументы, что позволяет создавать большое разнообразие стилей при помощи небольшого количества миксинов.
- Реализованные миксины можно посмотреть в файле /css/responsive/_mixins.scss.
- Для того, чтобы использовать миксины в своём файле, необходимо их импортировать @import'css/responsive/_mixins';

Миксин **mq** - используется для создания медиа-запроса, принимает два параметра:

1. Первый параметр это ширина экрана в px или один из трёх алиасов phone, tablet, desktop, которые будут заменены на 768px, 992px, 1280px соответственно.
2. Второй параметр необязательный, по умолчанию значение max, отвечает будет ли медиа запрос вида для максимальной или минимальной ширины (@media (**max**-width: $width) or @media (**min**-width: $width))

```
.header-vacancy {
    width: 100%;
    padding: 0 10px;

    @include mq('phone') {
        padding: 0;
    }
}
```

Миксин **breakpoint** - используется для медиа-запроса вида от ... до ... Принимает два параметра, строки вида "ширина в px" (например 600px) оба по умолчанию 0;

```
// При такой записи элемент будет скрыт при ширине экрана от 768px до 820px (включительно)
&__input {
    display: inline-block;
    vertical-align: top;
    width: 54px;
    text-align: center;
    padding: 5px 0;


    @include breakpoint(768px, 820px) {
        display: none;
    }
}
```

Миксин **z** - устанавливает свойство z-index для элемента, принимает обязательный параметр строку, одну из списка:

```
'loader',
'outdated-browser',
'modal',
'modal-overlay',
'site-header-2',
'site-header-1',
'site-header', // etc.
'page-wrapper', // z-index: 2
'site-footer' // z-index: 1
```

Чем ниже строка в списке тем меньше будет установлен z-index.

```scss
// Значение z-index будет 3
.header-container {
    position: relative;
    z-index: z('site-header');
    min-width: 1200px;
    background-color: $headerBackground;
    box-shadow: $headerBoxshadow;


    &--fixed {
        position: fixed;
        top: 0;
        right: 0;
        left: 0;
    }
}
```

Методология SCSS

- https://ru.bem.info/methodology/css/
- Также в компании есть сниппеты для переменны
  цветов, размеров и т.п. для разных редакторов

JS

Объявление переменных

Используйте `const` для объявления переменных; избегайте `var`.

> Почему? Это гарантирует, что вы не сможете переопределять значения, т.к. это может привести к ошибкам и к усложнению понимания кода.

```
// плохо
var a = 1;
var b = 2;

// хорошо
const a = 1;
const b = 2;
```

Если вам необходимо переопределять значения, то используйте `let` вместо `var`

> Почему? Область видимости `let` — блок, у `var` — функция.

```
// плохо
var count = 1;
if (true) {
  count += 1;
}

// хорошо, используйте let.
let count = 1;
if (true) {
  count += 1;
}
```

Строки

Используйте одинарные кавычки '' для строк.

```
// плохо
const name = "Capt. Janeway";

// плохо - литерал шаблонной строки должен содержать интерполяцию или переводы строк
const name = `Capt. Janeway`;

// хорошо
const name = 'Capt. Janeway';
```

Строки, у которых в строчке содержится более 100 символов, не пишутся на нескольких строчках с использованием конкатенации.

> Почему? Работать с разбитыми строками неудобно и это затрудняет поиск по коду.

```
// плохо
const errorMessage = 'This is a super long error that was thrown because \
of Batman. When you stop to think about how Batman had anything to do \
with this, you would get nowhere \
fast.';

// плохо
const errorMessage = 'This is a super long error that was thrown because ' +
  'of Batman. When you stop to think about how Batman had anything to do ' +
  'with this, you would get nowhere fast.';

// хорошо
const errorMessage = 'This is a super long error that was thrown because of Batman. When you stop to
```

Если строка содержит html разметку, она должна быть отформатированна следующим образом (любой html в js должен начинаться и заканчиваться на новой пустой строке):

```
// адаптивная часть сайта
const btns = `
    <div class="btns">
        <button class="btn btn--primary" type="button">${i18n._('В корзину')}</button>
        <button class="btn" type="button">${i18n._('Отмена')}</button>
    </div>
`;


// не адаптивная часть сайта
const btns = (
    '<div class="btns">' +
        '<button class="btn btn--primary" type="button">' + i18n._('В корзину') + '</button>' +
        '<button class="btn" type="button">' + i18n._('Отмена') + '</button>' +
    '</div>'
);
```

Есть не адаптивная часть сайта, например, админ-панель. Она находится вне видимости import общей сборки и придерживается старых стандартов.

При создании строки программным путём используйте шаблонные строки вместо конкатенации.

> Почему? Шаблонные строки дают вам читабельность, лаконичный синтаксис с правильными символами перевода строк и функции интерполяции строки.

```
// плохо
function sayHi(name) {
  return 'How are you, ' + name + '?';
}

// плохо
function sayHi(name) {
  return ['How are you, ', name, '?'].join();
}

// плохо
function sayHi(name) {
  return `How are you, ${ name }?`;
}

// хорошо
function sayHi(name) {
  return `How are you, ${name}?`;
}
```

Объекты

Для создания объекта используйте литеральную нотацию.

```
// плохо
const item = new Object();

// хорошо
const item = {};
```

Используйте сокращённую запись метода объекта.

```
// плохо
const atom = {
  value: 1,

  addValue: function (value) {
    return atom.value + value;
  },
};

// хорошо
const atom = {
  value: 1,

  addValue(value) {
    return atom.value + value;
  },
};
```

Используйте сокращённую запись свойств объекта.

> Почему? Это короче и понятнее.

```
const lukeSkywalker = 'Luke Skywalker';

// плохо
const obj = {
  lukeSkywalker: lukeSkywalker,
};

// хорошо
const obj = {
  lukeSkywalker,
};
```

Используйте оператор расширения вместо `Object.assign` для поверхностного копирования объектов. Используйте синтаксис оставшихся свойств, чтобы получить новый объект с некоторыми опущенными свойствами.

```
// очень плохо
const original = { a: 1, b: 2 };
const copy = Object.assign(original, { c: 3 }); // эта переменная изменяет `original` ಠ_ಠ
delete copy.a; // если сделать так

// плохо
const original = { a: 1, b: 2 };
const copy = Object.assign({}, original, { c: 3 }); // copy => { a: 1, b: 2, c: 3 }

// хорошо
const original = { a: 1, b: 2 };
const copy = { ...original, c: 3 }; // copy => { a: 1, b: 2, c: 3 }

const { a, ...noA } = copy; // noA => { b: 2, c: 3 }
```

Массивы

Для создания массива используйте литеральную нотацию.

```
// плохо
const items = new Array();

// хорошо
const items = [];
```

Для добавления элемента в массив следует использовать метод *Array::push* вместо прямого присваивания.

```
const someStack = [];

// Плохо
someStack[someStack.length] = 'abracadabra';

// Хорошо
someStack.push('abracadabra');
```

Если вам необходимо скопировать массив, используйте *Array::slice*.

> ⓘ Имейте ввиду, что конструкция "const array1 = array2;" не копирует массив, а создает на него ссылку!

```
const len = items.length;
const itemsCopy = [];
let i;

// Плохо
for (i = 0; i < len; i++) {
    itemsCopy[i] = items[i];
}

// Хорошо
itemsCopy = items.slice();


// Очень хорошо
const itemsCopy = [...items];
```

Свойства

Используйте точечную нотацию для доступа к свойствам.

```
const luke = {
  jedi: true,
  age: 28,
};

// плохо
const isJedi = luke['jedi'];

// хорошо
const isJedi = luke.jedi;
```

Запятые

Добавляйте висячие запятые.

> Почему? Такой подход даёт понятную разницу при просмотре изменений. Кроме того, транспиляторы типа Babel удалят висячие запятые из собранного кода, поэтому вы можете не беспокоиться о проблемах в старых браузерах.

```
// плохо - git diff без висячей запятой
const hero = {
     firstName: 'Florence',
-    lastName: 'Nightingale'
+    lastName: 'Nightingale',
+    inventorOf: ['coxcomb chart', 'modern nursing']
};

// хорошо - git diff с висячей запятой
const hero = {
     firstName: 'Florence',
     lastName: 'Nightingale',
+    inventorOf: ['coxcomb chart', 'modern nursing'],
};
```

Приведение типов

Строки

```
// плохо
const totalScore = new String(this.reviewScore); // тип totalScore будет "object", а не "string"

// плохо
const totalScore = this.reviewScore.toString(); // нет гарантии что вернется строка

// хорошо
const totalScore = String(this.reviewScore);
```

Числа: Используйте `Number` и `parseInt` с основанием системы счисления.

```
const inputValue = '4';

// плохо
const val = new Number(inputValue);

// плохо
const val = +inputValue;

// плохо
const val = inputValue >> 0;

// плохо
const val = parseInt(inputValue);

// хорошо
const val = Number(inputValue);

// хорошо
const val = parseInt(inputValue, 10);
```

Условные выражения и равенства

Используйте сокращения для булевских типов, а для строк и чисел применяйте явное сравнение.

```
// плохо
if (isValid === true) {
  // ...
}

// хорошо
if (isValid) {
  // ...
}

// плохо
if (name) {
  // ...
}

// хорошо
if (name !== '') {
  // ...
}

// плохо
if (collection.length) {
  // ...
}

// хорошо
if (collection.length > 0) {
  // ...
}
```

При смешивании операторов, помещайте их в круглые скобки. Единственное исключение — это стандартные арифметические операторы (+, −, * и /), так как их приоритет широко известен.

> Почему? Это улучшает читаемость и уточняет намерения разработчика.

```
// плохо
const foo = a && b < 0 || c > 0 || d + 1 === 0;

// плохо
const bar = a ** b - 5 % d;

// плохо
// можно ошибиться, думая что это (a || b) && c
if (a || b && c) {
  return d;
}

// хорошо
const foo = (a && b < 0) || c > 0 || (d + 1 === 0);

// хорошо
const bar = (a ** b) - (5 % d);

// хорошо
if (a || (b && c)) {
  return d;
}
```

Деструктуризация

При обращении к нескольким свойствам объекта используйте деструктуризацию объекта.

```
// плохо
function getFullName(user) {
  const firstName = user.firstName;
  const lastName = user.lastName;

  return `${firstName} ${lastName}`;
}

// хорошо
function getFullName(user) {
  const { firstName, lastName } = user;
  return `${firstName} ${lastName}`;
}

// отлично
function getFullName({ firstName, lastName }) {
  return `${firstName} ${lastName}`;
}
```

Никогда не объявляйте аргумент функции *arguments*, он будет более приоритетным над объектом *arguments*, который доступен без объявления для каждой функции.

```
// Плохо
function nope(name, options, arguments) {
    // ...код...
}

// Хорошо
function yup(name, options, args) {
    // ...код...
}
```

Используйте синтаксис записи аргументов по умолчанию, а не изменяйте аргументы функции.

```
// плохо
function handleThings(opts) {
  opts = opts || {};
  // ...
}

// хорошо
function handleThings(opts = {}) {
  // ...
}
```

ПОЛНЫЕ ГАЙДЫ БУДУТ ВЫДАНЫ ДЛЯ ИЗУЧЕНИЯ ПРИ ТРУДОУСТРОЙСТВЕ

Стандарт (ES6 / ECMAScript 2015)

- Современные синтаксические возможности для упрощения решений и повышения читаемости кода
  JS ES6: https://habr.com/ru/post/460741/

- Другие используемые стандарты:
  HTML5, CSS3, Twig 1.X

ESLINT – Автоматическая проверка

- Инструмент статического анализа кода для выявления проблемных шаблонов, обнаруженных в коде JavaScript

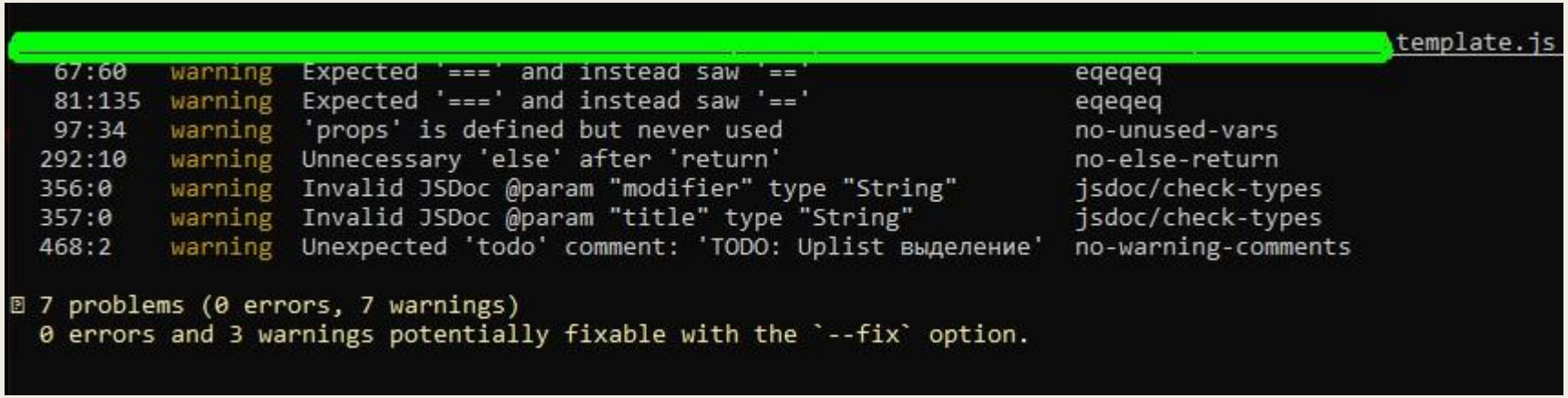
- Строгое равенство (===, а не ==)
- Неиспользуемые переменные
- Магические числа
- Константы в условиях
- Сложные функции

И множество других правил:

https://eslint.org/docs/latest/rules/

ESLINT – Запуск проверки

- Для работы нужен Node JS (открываем консоль)
- Команда:
  node [путь до ESLINT][пробел][путь до JS файла]
- Пример:
  node ./node_modules/eslint ./js/template.js



Code Review (Кодревью)

- Проверка кода выполнения задачи перед тестированием, которую проводит другой разработчик уровня «старший» и выше.

- Комментарии – вопросы и замечания по коду или решению
- Approve (Апрув) – подтверждение качества кода, которое позволяет перевести задачу в тестирование

ПРАКТИКА
ЧАСТЬ 1

1 Что здесь не так?

Плохой код

```
<button role="link" title="Name of website" tabindex="0">
  <img alt="Name of website" src="logo.jpg" title="Name of website">
</button>
```

1 Ответ

Ошибки и что следует исправить

- Пример неправильного использования элемента `<button>`. Для ссылок на другую страницу или сайт следует использовать элемент `<a>`. Не пренебрегайте семантикой HTML-тегов, если только в этом нет явной потребности
- Благодаря элементам `<a>`, на страницы можно ссылаться и без использования JavaScript
- Атрибут `title` описывает содержимое элемента в виде всплывающей подсказки и для элементов `<button>` указывать его излишне
- В атрибуте `tabindex` также нет необходимости, ведь при переключении между элементами с помощью клавиатуры кнопки получают фокус по умолчанию

Хороший код

```
<a href="https://">
  <img alt="Name of website" src="logo.jpg">
</a>
```

2 Что не так?

Плохой код

```
<img src="/images/edit.gif" onclick="openEditDialog(123)">
<img src="/images/delete.gif" onclick="openDeleteDialog(123)">
```

2 Ответ

Ошибки и что следует исправить

- Элемент `<img>` предназначен отнюдь не для выполнения JavaScript, а для показа изображений
- Как и на упомянутом ранее `<div>`, на элементе `img>` событие клика вызывается только непосредственно кликом мыши. Если бы вместо него использовался элемент `<button>`, это происходило бы ещё и при нажатии кнопок `Enter` или `Space` на клавиатуре
- Для самого изображения не задана текстовая альтернатива (атрибут `alt`). Из-за этого скринридеры могут озвучивать название самого файла изображения, что далеко не всегда информативно

Хороший код

Решение №1: Использовать кнопки, а к помещённым внутрь кнопок изображениям добавить атрибут `alt`

```
<button onclick="openEditDialog(123)">
  <img src="/images/edit.gif" alt="Edit product XY">
</button>
<button onclick="openDeleteDialog(123)">
  <img src="/images/delete.gif" alt="Delete product XY">
</button>
```

3

Плохой код

Контекст: это ссылка, стилизованная под кнопку. Ведёт она на форму, расположенную на этой же странице

```
<a href="#form" role="button" aria-haspopup="true">   Register   </a>
```

3

Ошибки и что следует исправить

- Добавляя к ссылке `role="button"` , вы сообщаете, что это кнопка, хотя она ведёт себя как ссылка. Не меняйте семантику элементов, только если в этом нет серьезной необходимости
- Атрибут `aria-haspopup="true"` призван сообщать вспомогательным устройствам, что данный элемент вызывает попап, но в нашем случае этого не происходит
- Внутренний отступ `padding` следует добавлять к элементам через CSS, а не с помощью ` `

Хороший код

```
.button {
  /* с помощью CSS задайте ссылке вид кнопки  */
}
```

```
<a class="button" href="#form"> Register </a>
```

4

Плохой код

```
<table>
  <tr id="body">
    <td id="body">
      <table id="body">
        <tr id="body_row">
          <td id="body_left">…</td>
          <td id="body_middle">…</td>
          <td id="body_right">…</td>
        </tr>
      </table>
    </td>
  </tr>
</table>
```

4

Ошибки и что следует исправить

- Значения атрибутов `id` должны быть уникальными, независимо от того, к какому тегу они добавляются
- Данный код использует раскладку, основанную на таблицах (это при том, что дизайн сайта обновлялся в 2016 году). Избегайте разметки страниц с помощью таблиц, потому что эти элементы имеют вполне конкретное семантическое значение и не предназначены для этих целей
- Текущую разметку следует заменить на семантические HTML5-теги. Это существенно сократит количество тегов и сделает код более понятным
- При стилизации следует использовать новые технологии Flexbox и CSS Grid, но никак не элементы таблиц
- Для значений атрибута `ID` должны быть использованы более семантические термины

Хороший код

```
<main id="body">
   <aside id="secondary_content"> </aside>
   <article id="primary_content"> </article>
   <aside id="tertiary_content"> </aside>
 </main>
```

5

Плохой код

```
<input type="checkbox" id="accept" required>
<label for="accept">
  <a href="/legal"> I accept the confidentiality policy and data… </a>
</label>
```

5

Ошибки и что следует исправить

- Вкладывать элементы с запускаемым поведением (таким как клик) считается плохим решением
- Возможность выбора чекбокса путём нажатия на его название улучшает удобство и доступность (путём увеличения области клика)
- Но в данном случае пользователи не ожидают, что при нажатии на название чекбокса откроется новая страница
- Размещайте ссылки за пределами элемента `<label>`

Хороший код

```
<input type="checkbox" id="accept" required>
<label for="accept"> I accept the confidentiality policy and data… </label>
(read <a href="/legal">Terms and conditions</a>)
```

6

Контекст: список карточек со ссылками, каждая из которых имеет заголовок, изображение и краткое описание

Плохой код

```
<section>
  <section>
    <h2>Overview</h2>
    <figure class="card" data-url="image1.html" style="background: url(image1.jpg)">
      <figcaption>
        <h4>My heading</h4>
        <article>Teasertext...</article>
      </figcaption>
    </figure>
    <figure class="card" data-url="image2.html" style="background: url(image2.jpg)"> … </fi
  </section>
</section>
```

6

Ошибки и что следует исправить

- Вероятнее всего, в подобных ситуациях необходимости в таком количестве элементов `<section>` нет. Чтобы лучше понять почему, рекомендую прочитать статью "Why You Should Choose HTML5 <article> Over <section>" автора Bruce Lawson
- Не следует недооценивать важность соблюдения иерархии заголовков. Например, пользователи скринридеров при навигации по сайту нередко ориентируются в структуре документа именно по его заголовкам
- Согласно спецификации, HTML5-элемент `<figure>` представляет собой самодостаточный элемент, который под основным содержимым опционально может содержать подпись. Но в этом примере нет содержимого, есть только подпись
- Изображнеие карточки не является декоративным, оно несёт какую-то информацию и должно быть частью HTML-кода документа, а не добавляться через CSS-свойство `background` . Фоновые изображения доступны пользователям не всех устройств
- В приведённом примере обработка клика на карточку происходит только через JavaScript. Если нет элемента ссылки с указанием пути (`<a href="path/to/page">`), для пользователей скринридеров переход на страницу карточки становится недоступным. Также элемент карточки не получает фокус при навигации с помощью клавиатуры
- Элементы `<h1>` - `<h6>` представляют собой вводный заголовок для родительского элемента `<section>` . `<h4>` является потоковым содержимым и, как следствие, технически может быть потомком `<figcaption>` , но лучше сделать его заголовком всей карточки
- Элемент `<article>` представляет собой самодостаточную композицию на странице. Это может быть газетная статья , эссе или отчёт, публикация в блоге или социальной сети. Для обычного абзаца текста лучше использовать элемент `<p>`
- Сделать доступной карточку, вся область которой является кликабельной, непросто. Дополнительную информацию можно найти в разделе "источники" ниже

6

Хороший код

```
<div>
  <section>
    <h2>Overview</h2>
    <article class="card">
      <h3>
        <a href="image1.html"> My heading </a>
      </h3>
      <img src="image1.jpg" alt="Description of image1" />
      <p>Teasertext...</p>
    </article>
    <article class="card"> … </article>
  </section>
</div>
```

7 Завершающее задание

Плохой код

```
<div>About us</div>
```

```
<div onClick="location.href='about.html'">
  About us
</div>
```

```
<div data-page="aboutus" data-url="index.php">
  About us
</div>
```

7

Ошибки и что следует исправить

- `<div>` - это элемент для крайнего случая, когда никакие другие элементы не подходят. Использование `<div>` вместо более подходящих по семантике элементов ухудшает доступность
- На `<div>` событие клика вызывается только непосредственно кликом мыши. На элементах `<a>` это происходит ещё и при нажатии на кнопку `Enter` на клавиатуре.
- `<div>` не получает фокус при переключении между элементами с помощью клавиатуры
- При нажатии правой кнопкой мыши в контекстном меню не будет пунктов "Открыть в новой вкладке/окне" или "Добавить ссылку в закладки"
- По умолчанию скринридеры просто озвучивают текст внутри `<div>` (например, "О нас"). В случае использования ссылки `<a>` скринридеры текст и роль элемента (например, "О нас, ссылка")
- Атрибуты наподобие `aria-label` у элементов `<div>` могут работать неправильно
- Пользователи скринридеров могут использовать раздел со списком ссылок страницы. `<div>` -ссылок в этом разделе не будет, если только к элементу не будет добавлен атрибут `role="link"`

Хороший код

```
<a href="aboutus.html">
  About us
</a>
```

Самостоятельное изучение

- Style Guide JS
  github.com/leonidlebede/javascript-airbnb

ПРАКТИКА
ЧАСТЬ 2

8 Что здесь не так?

```
class Character {
    handleInput(input){
        if(input.keyCode == 87) { // 'w'
            this.y += 5;
        } else if(input.keyCode == 68) { // 'd'
            this.x += 5;
        } else if(input.keyCode == 83) { // 's'
            this.y -= 5;
        } else if(input.keyCode == 65) { // 'a'
            this.x -= 5;
        }
    }
    ...
```

8.1

Каждый раз, когда вы видете нетривиальную строку и рядом с ней комментарий, который ее поясняет - это плохая строка. `if(input.keyCode == 87) { // 'w'` - неужели каждый раз придется писать рядом `// 'w'`? А если мы поменяем ее на `if(input.keyCode == 86) ...` нам придется менять и сам комментарий. Один раз мы это сделаем, другой раз забудем. И получится врущий комментарий `if(input.keyCode == 86) // 'w'`. Это одна из причин, почему таких комментариев стоит избегать.

Кстати, об изменениях. Что, если ниже у нас есть еще пара методов, в которых тоже есть злополучные `if(input.keyCode == 87)`, а мы вдруг захотим вместо `WASD` использовать стрелки клавиатуры? Придется в каждом месте руками изменять 87 на 35, 68 на 39 и так далее. Не хорошо.

```
// Запишем только нужные нам коды
const KEY_CODE = {
    A: 65,
    D: 68,
    S: 83
    W: 87,
};

class Character {
    handleInput(input){
        if(input.keyCode == KEY_CODE.W) {
            this.y += 5;
        } else if(input.keyCode == KEY_CODE.D) {
            this.x += 5;
        } else if(input.keyCode == KEY_CODE.S) {
            this.y -= 5;
        } else if(input.keyCode == KEY_CODE.A) {
            this.x -= 5;
        }
    }
```

8.2

Однако, если мы захотим изменить клавиши управления нашим персонажем, у нас получится что-то вроде

```
const KEY_CODE = {
    A: 37,
    D: 39,
    S: 40,
    W: 38,
};
```

Опять получается, что наш код врет. Мы говорим, что букве А соответсвует число 37, хотя это не так. Конечно, наш код будет работать, и наш персонаж будет двигаться при помощи стрелок, однако в дальнейшем такой код может легко запутать всех, кто будет над ним работать, включая вас, самого автора.

Для исправления этой проблемы достаточно изменить имена переменных

```
const INPUT_KEY = {
    LEFT: 65
    RIGHT: 68,
    DOWN: 83
    UP: 87,
};
```

8.3

Теперь подумаем, что, если мы захотим изменить скорость передвижения нашего персонажа. Нам придется изменять руками число 5 на значение, которое мы хотим, и делать это по всему коду. Нехорошо, учитывая, что число 5 в другом месте может означать совершенно другое значение, и, если у нас много таких магических констант, каждое такое исправление превратиться в ад. И если такое происходит - ваш код точно плох.

Определим методы для движения нашего персонажа, а также вынесем скорость в отдельные переменные, которые легко изменять.

```
const SPEED = 5;
```

9

```
int n = A.length; // Количество сотрудников, которые еще не получили зарплату
```

9

Комментарий пытается объяснить то, для чего нужна переменная *n*. При этом само имя *n* ничего нам не говорит. Это явные сигналы того, что переменной стоит дать более содержательное имя. *employessNumberWithoutSalary* вполне подойдет. Длинно, зато понятно.

С другой стороны, массив, содержащий этих сотрудников, называется *A*. Это тоже не несет в себе никакой информации. Переименуй мы его в *employeesWithoutSalary*, объяснять, что такое *n* скорее всего не пришлось бы.

```
int n = employeesWithoutSalary.length;
```

10

```
function sumAllElementsInArray(arrayOfElements){
    let overallSum = 0;
    for(let indexOfElement; indexOfElement < arrayOfElements.length; indexOfElement++){
        overallSum += arrayOfElements[indexOfElement];
    }
    return overallSum;
}
```

10 Вариант решения

```
function sumAllElementsInArray(arrayOfElements){
    let overallSum = 0;
    for(let indexOfElement; indexOfElement < arrayOfElements.length; indexOfElement++){
        overallSum += arrayOfElements[indexOfElement];
    }
    return overallSum;
}
```

```
function sumArray(array){
    let sum = 0;
    for(let i; i < array.length; i++){
        sum += array[i];
    }
    return sum;
}
```

11

```
class Board {

    constructor(){
        this.mainList = [];
    }

    getFl(){
        let a = [];

        for(let i = 0; i < this.mainList; i++){
            if(this.mainList[i].value === 4){
                a.append(this.mainList[i]);
            }
        }

        return a;
    }
}
```

11 Вариант решения

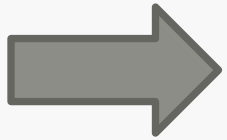
```
class Board {

    constructor(){
        this.mainList = [];
    }

    getFl(){
        let a = [];

        for(let i = 0; i < this.mainList; i++){
            if(this.mainList[i].value === 4){
                a.append(this.mainList[i]);
            }
        }

        return a;
    }
}
```



```
class Board {

    constructor(){
        this.cells = [];
    }

    getFlaggedCells(){
        let flagged = [];

        for(let i = 0; i < this.cells; i++){
            if(this.cells[i].isFlagged()){
                flagged.append(this.cells[i]);
            }
        }

        return flagged;
    }
}
```

12

```
let a = 1;
if(0 == 1){
    a = 01;
} else {
    1 = 01;
}
```

12

```
let a = 1;
if(0 == 1){
    a = 01;
} else {
    1 = 01;
}
```

Конечно, имена здесь как минимум бессмысленные, но в этом примере я хотел показать, почему не стоит называть переменные так, чтобы их трудно было отличить от чисел. Иногда в качестве имени локальной переменной хочется использовать *l* или *O*, однако их бывает сложно отличить от цифр 1 и 0, особенно в некоторых шрифтах. Если уж вы используйте короткие имена для переменных, используйте *i*, или *k*, или *j*. Если у вас уже есть переменные с такими именами, но вам нужна еще одна однобуквенная - подумайте, скорее всего вы что-то делаете не так.

Задание 1

- Написать HTML верстку для статьи
- Добавить стилей CSS в отдельном файле, **не в style**

5 Python Projects to Automate Your Life: From Beginner to Advanced

Brand-new automation projects that you should solve in 2022

Image via Shutterstock under license to Frank Andrade

If you're learning Python, you should try to automate your everyday tasks.

You not only learn more Python by implementing what you already know but, in the end, you can see how all your hard work pays off.

Automate Microsoft Word (Beginner Project)

How many times did you have to use Word to create a cover letter, contract agreement, resume, or report?

Задание 2

- Отсортировать массив чисел так, чтобы нечетные пары по порядку были по возрастанию, а четные – по убыванию.
- Например: [6, 4, 1, 7] -> [1, 6, 4, 7]

1 Пара – 1 и 6

2 Пара, четная – 6 и 4