

Базовые критерии

Задача

Код соответствует техническому заданию проекта

Все обязательные пункты техническому заданию выполнены

При выполнении кода не возникает необработанных ошибок

При открытии диалогов, загрузки данных и работе с сайтом не возникает ошибок, программа не ломается и не зависает

Именованние

Название переменных, параметров, свойств и методов начинается со строчной буквы и записываются в нотации CamelCase

Для названия значений используются английские существительные

Сокращения в словах запрещены. Сокращённые названия переменных можно использовать только, если такое название широко распространено. Допустимые сокращения:

- xhr, для объектов XMLHttpRequest
- evt для объектов Event и его производных (MouseEvent, KeyboardEvent и подобные)
- ctx для контекста канваса
- i, j, k, l, t для счётчика в цикле, j для счётчика во вложенном цикле и так далее по алфавиту
- если циклов два и более, то можно не переиспользовать переменную i
- cb для единственного коллбэка в параметрах функции

Массивы названы существительными во множественном числе

Неправильно:

```
var age = [12, 40, 22, 7];
var name = ['Иван', 'Петр', 'Мария', 'Алексей'];

var wizard = {
  name: 'Гендальф',
  friend: ['Саурон', 'Фродо', 'Бильбо']
};
```

Правильно:

```
var ages = [12, 40, 22, 7];
var names = ['Иван', 'Петр', 'Мария', 'Алексей'];

var wizard = {
  name: 'Гендальф',
  friends: ['Саурон', 'Фродо', 'Бильбо']
};
```

Название функции или метода содержит глагол

Название функции/метода должно быть глаголом и соответствовать действию, которое выполняет функция/метод. Например, можно использовать глагол get для функций/методов, которые что-то возвращают

Исключения:

1. Функции конструкторы (см. критерий Конструкторы названы английскими существительными)
2. Функции обработчики/коллбэки (см. критерий Из названия обработчика события и функции-коллбэка следует, что это обработчик)

Неправильно:

```
var function1 = function (names) {
  names.forEach(function (name) {
    console.log(name);
  });
};

var wizard = {
```

```

    name: 'Гендальф',
    action: function () {
        console.log('Стреляю файрболлом!');
    }
};

var randomNumber = function () {
    return Math.random();
};

```

Правильно:

```

var printNames = function (names) {
    names.forEach(function (name) {
        console.log(name);
    });
};

var wizard = {
    name: 'Гендальф',
    fire: function () {
        console.log('Стреляю файрболлом!');
    }
};

var getRandomNumber = function () {
    return Math.random();
};

```

Названия констант (постоянных значений) написаны прописными (заглавными) буквами

Слова разделяются подчёркиваниями (UPPER_SNAKE_CASE), например:

```

var MAX_HEIGHT = 400;
var EARTH_RADIUS = 6370;

```

Название конструкторов и перечислений начинается с заглавной буквы. В названии используются английские существительные.

Неправильно:

```

var wizard = function (name, age) {
    this.name = name;
    this.age = age;
};

var Fly = function (coordinate) {
    console.log('Смотрите я лечу!');
};

```

Правильно:

```

var Wizard = function (name, age) {
    this.name = name;
    this.age = age;
};

var fly = function (coordinate) {
    console.log('Смотрите я лечу!');
};

```

Названия функций не являющихся конструкторами должны начинаться со строчной буквы

Форматирование и внешний вид

Используются обязательные блоки кода

В любых конструкциях, где подразумевается использование блока кода (фигурных скобок), таких как `for`, `while`, `if`, `switch`, `function` — блок кода используется обязательно, даже если инструкция состоит из одной строки

Неправильно:

```

var isEven = true;
if (x % 2 === 1) isEven = false;

```

Правильно:

```

var isEven = true;
if (x % 2 === 1) {

```

```
    isEven = false;
}
```

Список констант идёт перед основным кодом

Все константы выносятся в начало модуля

Код соответствует гайдлайнам (ESLint)

Для проверки можно использовать любой доступный конфиг ESLint

- Отступы между операторами и ключевым словами соответствуют стайлгайду.
- Для отступов используются одинаковые символы, вложенность кода обозначается отступами.
- Однообразно расставлены пробелы перед, после и внутри скобок, операторов и ключевых слов

Мусор

В итоговом коде проекта находятся только те файлы, которые были на момент создания репозитория, которые были получены в патчах и файлы, созданные по заданию

В коде проекта нет файлов, модулей и частей кода, которые не используются, включая, закомментированные участки кода

Нет файлов скриптов, которые не подключены в файле `index.html`

В коде нет заранее недостижимых участков кода

Например:

- Невыполнимые условия:

```
var happen = false;
if (happen) {
  console.log('This will not happen anyway!');
}
```

- Операции после выхода из функции:

```
(function () {
  return;
  console.log('This will not happen!');
})();
```

Корректность

Константы нигде в коде не переопределяются

Константы используются только для чтения, и никогда не переопределяются на всем промежутке жизни программы

Включён строгий режим (ESLint)

В коде запрещены небезопасные конструкции. Код работает в [строгом режиме](#). В начале js-файлов установлена директива `'use strict';`

Используются строгие сравнения вместо нестрогих (ESLint)

Вместо операторов нестрогого сравнения `==` и `!=`, используются операторы строгого сравнения `===`, `!==`. [Таблицы истинности](#) для JavaScript

Неправильно:

```
var foo = '';
var bar = [];
if (foo == bar) {
  destroy(world);
}
```

Правильно:

```
var foo = '';
var bar = [];
```

```
if (foo === bar) {  
    destroy(world);  
}
```

В коде не используются зарезервированные слова в качестве имён переменных и свойств

В названия переменных и свойств не включаются операторы и ключевые слова зарезервированные для будущих версий языка (например, `class`, `extends`). Список всех зарезервированных слов можно найти [тут](#)

Модульность

Все скрипты подключаются через файл `index.html`

Файлы скриптов подключаются перед закрывающимся тегом `</body>`, атрибуты `async` и `defer` не используются

Все файлы JS представляют собой отдельные модули в [IIFE](#)

Экспорт значений производится через глобальную область видимости. Код вне модуля запрещён. Вне модуля могут располагаться комментарии и утилитные инструкции, такие как `'use strict'`;

Пример правильного модуля:

```
'use strict';  
  
(function () {  
    window.load = function (url, onLoad) {  
        var xhr = new XMLHttpRequest();  
        xhr.addEventListener('load', onLoad);  
  
        xhr.responseType = 'json';  
        xhr.open('GET', url);  
        xhr.send();  
    };  
})();
```

Все значения, используемые только внутри модулей ограничены по видимости

Из модуля ничего не должно попадать случайными образом в глобальную область видимости

Неправильно:

```
'use strict';  
  
var ENTER_KEYCODE = 13;  
  
(function () {  
    var userIcon = document.querySelector('.user');  
  
    userIcon.addEventListener('keydown', function (evt) {  
        if (evt.keyCode === ENTER_KEYCODE) {  
            popup.classList.remove('hidden');  
        }  
    });  
})();
```

Правильно:

```
'use strict';  
  
(function () {  
    var ENTER_KEYCODE = 13;  
  
    var userIcon = document.querySelector('.user');  
  
    userIcon.addEventListener('keydown', function (evt) {  
        if (evt.keyCode === ENTER_KEYCODE) {  
            popup.classList.remove('hidden');  
        }  
    });  
})();
```

Универсальность

Код является кроссбраузерным и не вызывает ошибок в разных браузерах и разных операционных системах

При проверке этого критерия, необходимо удостовериться в правильной работе и отсутствии сообщений об ошибках в выполняемых скриптах только в последних версиях браузеров: Chrome, Firefox, Safari, Microsoft Edge.

Допустимое исключение в кроссбраузерности кода: валидация форм в Safari. Safari плохо поддерживает работу с валидацией, например, не показывает ошибку, если при отправке формы не введены данные в поле с атрибутом required, поэтому небольшие ошибки, связанные с валидацией в Safari можно проигнорировать. Тестирование необходимо проводить именно в последних версиях браузеров, а не в тех, которые установлены в данный момент на компьютере проверяющего. Важно: в Windows старая версия Safari, поэтому тестировать в нём не нужно. IE не поддерживается, только Edge.

Магия

Нельзя пользоваться глобальной переменной event

Приводит к неосознанному коду:

```
var elem = document.querySelector('.test');

var onElemClick = function () {
  event.target.innerText = 'you really need event';
};

elem.addEventListener('click', oneElemClick);
```

Оптимальность

Своевременный выход из цикла: цикл не работает дольше чем нужно

Неправильно:

```
apartments.forEach(function (it, index) {
  if (index < 3) {
    render(it);
  }
});
```

Правильно:

```
for (var i = 0; i < Math.min(apartments.length, 3); i++) {
  render(apartments[i]);
}
```

Количество вызовов циклов минимизировано

Если задачу можно решить за один проход по циклу, вместо нескольких она должна быть решена за один

Неправильно:

```
var wizardNames = source.map(function (it) {
  return it.wizard;
}).map(function (it) {
  return it.name;
});
```

Правильно:

```
var wizardNames = source.map(function (it) {
  return it.wizard.name;
});
```

Множественные DOM-операции производятся на элементах, которые не добавлены в DOM

Например, наполнение скопированного из шаблона элемента данными

Безопасность

Обработчики события добавляются и удаляются своевременно

Обработчики событий для виджетов добавляются только в момент появления виджета на странице и удаляются в момент их исчезновения.

Защита от memory-leak

Кол-во обработчиков подвешенных на глобальную область видимости не должно возрастать. Например, если подвешивается

обработчик, который следит за перемещением курсора по экрану, то он должен подвешиваться и отвешиваться в нужный момент. В случае если обработчик на `document` только подвешивается это может свидетельствовать о проблеме бесконечного создания обработчиков и потенциальной утечке памяти.

Защита от неправильного поведения интерфейса Например, на странице может существовать попап, который скрывается по ESC. Лучше для него гасить обработчик, если он не показан, потому что он может каким-то образом ломать поведение сайта — останавливать распространение, отменять поведение по умолчанию и т.д. Поэтому поведение должно быть **явным** — если в этот момент времени обработчики не нужны, их нужно удалить. Явное и предсказуемое поведение.

Для вставки пользовательских строк (имён, фамилий и т.д.) использован `textContent`

Защита от XSS-атак, а также изменения исходных данных, запутывание пользователя и прочее

Дополнительные критерии

Задача

Техническое задание реализовано в полном объёме

Все обязательные и необязательные пункты технического задания выполнены

Именованние

Переменные носят абстрактные названия и не содержат имён собственных

Неправильно:

```
var keks = {
  name: 'Кекс'
};
```

Правильно:

```
var cat = {
  name: 'Кекс'
};
```

Название методов и свойств объектов не содержит название объектов

Неправильно:

```
popup.openPopup = function () {
  console.log('I will open popup');
};
wizard.wizardName = 'Пендальф';
```

Правильно

```
popup.open = function () {
  console.log('I will open popup');
};
wizard.name = 'Пендальф';
```

Из названия обработчика события и функции-коллбэка следует, что это обработчик

Для единственного обработчика или функции можно использовать `callback` или `cb`. Для именования нескольких обработчиков внутри одного модуля используется `on` или `handler` и описание события. Название обработчика строится следующим образом:

- `on` + (на каком элементе) + что случилось:

```
var onSidebarClick;
var onContentLoad;

var onResize;
```

- (на каком элементе) + что случилось + `Handler`:

```
var sidebarClickHandler;
var contentLoadHandler;

var resizeHandler;
```

Единообразие

Все функции объявлены единообразно

Все функции создаются в едином стиле: используется либо [«функциональное выражение»](#), либо как [«функциональное объявление»](#). Смешение стилей в рамках проекта не допускается

Неправильно:

```
var doSomethingElse = function () {  
  // function body  
};  
  
function doSomething() {  
  // function body  
}
```

Правильно:

```
var doSomething = function () {  
  // function body  
};  
  
var doSomethingElse = function () {  
  // function body  
};
```

или

```
function doSomething() {  
  // function body  
}  
  
function doSomethingElse() {  
  // function body  
}
```

Используется единый стиль именования переменных

Стиль именования переменных сохраняется во всех модулях, например:

- не следует мешать обработчики содержащие Handler и on
- если переменные, которые хранят DOM-элемент и содержат слово Element, то это правило работает везде.

Неправильно:

```
var popupMainElement = document.querySelector('.popup');  
var sidebarNode = document.querySelector('.sidebar');  
var similarContainer = popupMainElement.querySelector('ul.similar');
```

Правильно:

```
var popupMainElement = document.querySelector('.popup');  
var sidebarElement = document.querySelector('.sidebar');  
var similarContainerElement = popupMainElement.querySelector('ul.similar');
```

При использовании встроенного API, который поддерживает несколько вариантов использования, используется один способ

Если существуют несколько разных **API** позволяющих решить одну и ту же задачу, например поиск элемента по **id** в DOM-дереве, то в проекте используется только один из этих **API**.

Неправильно:

```
var popupMainElement = document.querySelector('#popup');  
var sidebarElement = document.getElementById('sidebar');  
  
var popupClassName = popupMainElement.getAttribute('class');  
var sidebarClassName = sidebarElement.className;
```

Правильно:

```
var popupMainElement = document.querySelector('#popup');  
var sidebarElement = document.querySelector('#sidebar');  
  
var popupClassName = popupMainElement.getAttribute('class');  
var sidebarClassName = sidebarElement.getAttribute('class');
```

или

```
var popupMainElement = document.getElementById('popup');
var sidebarElement = document.getElementById('sidebar');

var popupClassName = popupMainElement.className;
var sidebarClassName = sidebarElement.className;
```

Корректность

API встроенных функций и объектов используется правильно

Передаются корректные значения, которые ожидаются по спецификации

Неправильно:

```
var isPressed = element.getAttribute('aria-pressed', false);
```

Правильно:

```
var isPressed = element.getAttribute('aria-pressed');
```

Встроенные методы массивов используются по назначению.

Неправильно:

```
var greet = 'Привет ';

wizards.map(function (it) {
  greet += ', ' + it.name;
});

console.log(greet + '!!');
```

Правильно:

```
var greet = 'Привет ';

var names = wizards.map(function (it) {
  return it.name;
});

console.log(greet + names.join(', ') + '!!');
```

Отсутствуют потенциально некорректные операции

Например некорректное сложение двух операндов как строк. Проблема приоритета конкатенации над сложением.

Неправильно:

```
new Date() + 1000;
```

Правильно:

```
+new Date() + 1000;
```

Некорректные проверки на существование с числами. Пример некорректной проверки на то, что переменная является числом:

```
var double = function (value) {
  if (!value) {
    return NaN;
  }

  return value * 2;
};

double(0);
double();
double(5);
```

Модульность

В случае, если одинаковый код повторяется в нескольких модулях, повторяющаяся часть вынесена в отдельный модуль

Критерий касается структурных единиц кода — повторяющийся блок кода, либо функции с одним и теми же конструкциями, например, утилитные методы для проверки клавиш:

```
// Файл keyboard.js
'use strict';

(function () {
  var ENTER_KEYCODE = 13;
  var ESC_KEYCODE = 27;

  window.keyboard = {
    isKeyBoardEvent: function (evt) {
      return evt instanceof KeyboardEvent;
    },
    isEnterPressed: function (evt) {
      return evt.keyCode === ENTER_KEYCODE;
    },
    isEscPressed: function (evt) {
      return evt.keyCode === ESC_KEYCODE;
    }
  };
})();
```

Не стоит выносить в отдельный модуль одну повторяющуюся инструкцию:

```
// Файл hide-gallery.js
'use strict';

(function () {
  window.hideGallery = function (gallery) {
    return gallery.classList.add('invisible');
  };
})();
```

При экспорте из одного модуля нескольких значений используется пространство имён

Множественные значения записываются в один объект. Имя объекта совпадает с именем файла без учёта кейса. Неправильно:

```
// Файл dialog-util.js
'use strict';

(function () {
  var ENTER_KEYCODE = 13;
  var ESC_KEYCODE = 27;

  window.isEnterPressed = function (evt) {
    return evt.keyCode === ENTER_KEYCODE;
  };

  window.isEscPressed = function (evt) {
    return evt.keyCode === ESC_KEYCODE;
  };
})();
```

Правильно:

```
// Файл dialog-util.js
'use strict';

(function () {
  var ENTER_KEYCODE = 13;
  var ESC_KEYCODE = 27;

  window.dialogUtil = {
    isEnterPressed: function (evt) {
      return evt.keyCode === ENTER_KEYCODE;
    },
    isEscPressed: function (evt) {
      return evt.keyCode === ESC_KEYCODE;
    }
  };
})();
```

Во всех модулях для ограничения области видимости используются ПФЕ и только они

Неправильно:

```
'use strict';

window.load = function (url, onLoad) {
  var xhr = new XMLHttpRequest();
  xhr.addEventListener('load', onLoad);
```

```
xhr.responseType = 'json';
xhr.open('GET', url);
xhr.send();
};
```

Правильно:

```
'use strict';

(function () {
  window.load = function (url, onLoad) {
    var xhr = new XMLHttpRequest();
    xhr.addEventListener('load', onLoad);

    xhr.responseType = 'json';
    xhr.open('GET', url);
    xhr.send();
  };
})();
```

Избыточность

В проекте не должно быть избыточных проверок

Например, если заранее известно, что функция всегда принимает числовой параметр, то не следует проверять его на существование

Неправильно:

```
var isPositiveNumber = function (myNumber) {
  if (typeof myNumber === 'undefined') {
    throw new Error('Parameter is not defined');
  }
  return myNumber > 0;
};

isPositiveNumber(15);
isPositiveNumber(-30);
```

Правильно:

```
var isPositiveNumber = function (myNumber) {
  return myNumber > 0;
};

isPositiveNumber(15);
isPositiveNumber(-30);
```

Отсутствует дублирование кода: повторяющиеся части кода переписаны как функции

При написании кода следует придерживаться принципа [DRY](#)

Если при использовании условного оператора в любом случае возвращается значение, альтернативная ветка опускается

Неправильно:

```
var decide = function (val, anotherVal) {
  if (2 > 1) {
    return val;
  } else {
    return anotherVal;
  }
};
```

Правильно:

```
var decide = function (val, anotherVal) {
  if (2 > 1) {
    return val;
  }

  return anotherVal;
};
```

Отсутствуют лишние приведения и проверки типов

Если заранее известно что в переменной число, то нет смысла превращать переменную в число `parseInt(myNumber)`. Тоже касается и избыточной проверки булевой переменной

Неправильно:

```
if (booleanValue === true) {
  console.log('It\'s true!');
}
```

Правильно:

```
if (booleanValue) {
  console.log('It\'s true!');
}
```

Там где возможно, в присвоении значения вместо if используется тернарный оператор

Неправильно:

```
var sex;
if (male) {
  sex = 'Мужчина';
} else {
  sex = 'Женщина';
}
```

Правильно:

```
var sex = male ? 'Мужчина' : 'Женщина';
```

Условия упрощены

Если функция возвращает булево значение, не используется `if..else` с лишними `return`

Неправильно:

```
var equals = function (firstValue, secondValue) {
  if (firstValue === secondValue) {
    return true;
  } else {
    return false;
  }
};
```

Правильно:

```
var equals = function (firstValue, secondValue) {
  return firstValue === secondValue;
};
```

Магия

В коде не используются «магические значения», под каждое из них заведена отдельная переменная, названная как константа

Оптимальность

Константы, используемые внутри функций создаются вне функций и используются повторно через замыкания

Поиск элементов по селекторам делается минимальное количество раз, после этого ссылки на элементы сохраняются

Неправильно:

```
for (var i = 0; i < Math.min(apartments.length, 3); i++) {
  var dialog = document.querySelector('.dialog');
  render(dialog, apartments[i]);
}
```

Правильно:

```
var dialog = document.querySelector('.dialog');
```

```
for (var i = 0; i < Math.min(apartments.length, 3); i++) {  
    render(dialog, apartments[i]);  
}
```

Массивы и объекты, содержимое которых вычисляется, собираются один раз, а после этого только переиспользуются

Изменения применяются точно

Например, при удалении классов с DOM-элемента, не производится попытка удалить все возможные классы, если можно убрать лишь тот, который действительно установлен на DOM-элементе в данный момент

Неправильно:

```
var imageContainer = document.querySelector('.image-container');  
  
var changeFilter = function (filterName) {  
    imageContainer.classList.remove('filter-chrome', 'filter-sepia', 'filter-marvin', 'filter-phobos', 'filter-heat');  
    imageContainer.classList.add(filterName);  
};
```

Правильно:

```
var imageContainer = document.querySelector('.image-container');  
  
var currentFilter;  
var changeFilter = function (filterName) {  
    if (currentFilter) {  
        imageContainer.classList.remove(currentFilter);  
    }  
    imageContainer.classList.add(filterName);  
    currentFilter = filterName;  
};
```

Сложность. Читаемость.

Для каждого события используется отдельный обработчик

Одна функция не является обработчиком нескольких разных событий

Неправильно:

```
var activationHandler = function (evt) {  
    if (evt.keyCode === 13 || evt.type === 'click') {  
        showPopup();  
    }  
};  
  
button.addEventListener('click', activationHandler);  
button.addEventListener('keydown', activationHandler);
```

Правильно:

```
var buttonClickHandler = function () {  
    showPopup();  
};  
  
var buttonKeydownHandler = function (evt) {  
    if (evt.keyCode === 13) {  
        showPopup();  
    }  
};  
  
button.addEventListener('click', buttonClickHandler);  
button.addEventListener('keydown', buttonKeydownHandler);
```

Длинные функции и методы разбиты на несколько небольших

Для работы с JS-коллекциями используются итераторы для массивов

Итераторы используются для трансформаций массивов — map, filter, sort и прочие. А также для обхода проблемы потери окружения в циклах — forEach

Например:

```
elements.forEach(function (el) {  
    el.onclick = function () {
```

```
    console.log(e1);  
  };  
});
```

Оператор присваивания не используется как часть выражения

Неправильно:

```
imgGenerate(picArray = JSON.parse(data));
```

Правильно:

```
picArray = JSON.parse(data);  
imgGenerate(picArray);
```