Критерии

Подготовка и проверка личных проектов проводится по базовым и дополнительным критериям.

Базовые критерии охватывают наиболее важные требования к проекту и проверяют основные знания и навыки. Для успешной защиты личного проекта должны быть выполнены все базовые критерии.

Дополнительные критерии проверяют то, насколько студент внимателен к деталям, и оценивают проект с точки зрения шлифовки его качества и оптимизации. Выполнение этих критериев необходимо для защиты на 100%.

Во время финальной защиты баллы за выполнение дополнительных критериев добавляются только при выполнении всех базовых

Базовые

Задача Свернуть все

- Б1. Код соответствует техническому заданию

проекта.
Все обязательные пункты технического задания выполнены.

- Б2. При выполнении кода не возникает необработанных ошибок.
При открытии диалогов, загрузке данных и работе с сайтом не возникает ошибок, программа не ломается и не зависает.

Именование Свернуть все

- Б3. Название переменных, параметров, свойств и методов начинается со строчной буквы и записываются в нотации lowerCamelCase.
Исключение составляют перечисления, они записываются в нотации CamelCase.
Критерий касается как переменных, объявленных пользователем, так и полученных извне.
Такие данные нужно адаптировать. Например, руками:

```
fetch(`https://my-site.fake/api/user`)
  .then((response) => response.json())
  .then(({user_name: userName, user_age: userAge}) => ({
    userName,
    userAge
  }));
```

- Но лучше написать для этого отдельную функцию.

- Б4. Для названия значений используются английские существительные.
  Сокращения в словах запрещены. Сокращённые названия переменных можно использовать, только если такое название широко распространено.
  Допустимые сокращения:
  - `evt` для объектов `Event` и его производных (`MouseEvent`, `KeyboardEvent` и подобные)
  - `i`, `j`, `k`, `l`, `t` для счётчика в цикле, `j` для счётчика во вложенном цикле и так далее по алфавиту
  - `cb` для единственного колбэка в параметрах функции
- Допустимо именовать переменные-предикаты — флаги или функции, которые возвращают булево значение — по схеме «`is` + признак».
  Например
  ```
  const isLoading = true;
  ```
-
- ```
  const isChecked = (checkboxes) =>
  checkboxes.some((checkbox) =>
  ```
  ```

`checkbox.checked);`

Использовать `data` для аргумента с данными запрещено. Название должно быть осмысленным.

Неправильно

```
const getFilterMarkup = (data) => {
  return data.map(
      (i) => <label><input type="radio" name="filter-${i.id}" />${i.name}</label>
  ).join(``);
};
```

Правильно

```
const getFilterMarkup = (filters) => {
  return filters.map(
      (filter) => <label><input type="radio" name="filter-${filter.id}" />${filter.name}</label>
  ).join(``);
};
```

Б5. Названия констант (постоянных значений) написаны прописными (заглавными) буквами. Слова разделяются подчёркиваниями

(UPPER_SNAKE_CASE), например:

```
const MAX_HEIGHT = 400;
const EARTH_RADIUS = 6370;
```

Б6. Классы названы английскими существительными. Название класса начинается с заглавной буквы.

Неправильно:

```
class wizard {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
}

class Run {
  constructor() {
    console.log(`О, я бегу!`);
  }
}
```

Правильно:

```
class Wizard {
  constructor(name, age) {
```

```
    this.name = name;
    this.age = age;
  }
}

class Runner {
  constructor() {
    console.log(`О, я бегун!`);
  }
}
```

Б7. Перечисления (Enum) названы английскими существительными и начинаются с прописной (заглавной) буквы.

Перечисления начинаются с прописной (заглавной) буквы. Перечисления названы существительными в единственном числе. Значения перечислений объявлены как константы.

Неправильно:

```
const view = {
  artist: Artist,
  genre: Genre,
};

const EndGameType = {
```

```
  lives: `lives`,
  quests: `quests`,
};
```

Правильно:

```
const View = {
  ARTIST: Artist,
  GENRE: Genre,
};

const EndGameType = {
  LIVES: `lives`,
  QUESTS: `quests`,
};
```

Б8. Массивы названы существительными во множественном числе.

Неправильно:

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

const wizard = {
```

```
  name: `Гендальф`,
  friend: [`Саурон`, `Фродо`, `Бильбо`]
};
```

Правильно:

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

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

Б9. В названии переменных не используется тип данных.

Неправильно:

```
const filtersArray = [`All`, `Past`, `Feature`];

const wizardObject = {
  name: `Гендальф`,
  age: 386
```

```
};
```

Правильно:

```
const filters = [`All`, `Past`, `Feature`];

const wizard = {
  name: `Гендальф`,
  age: 386
};
```

Б10. Название функции или метода содержит глагол.

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

Исключение функции-обработчики (см. критерий `Из названия обработчика события и функции-колбэка следует, что это обработчик`). Исключение справедливо только для выполнения дополнительного критерия.

Неправильно:

```
const function1 = (names) => {
  names.forEach((name) => {
```

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

const wizard = {
  name: `Гендальф`,
  action() {
    console.log(`Стреляю файрболлом!`);
  }
};

const randomNumber = () => {
  return Math.random();
};
```

Правильно:

```
const printNames = (names) => {
  names.forEach((name) => {
    console.log(name);
  });
};

const wizard = {
  name: `Гендальф`,
  fire() {
    console.log(`Стреляю файрболлом!`);
```

```
  }
};

const getRandomNumber = () => {
  return Math.random();
};
```

Б11. Названия файлов модулей записаны строчными (маленькими) буквами. Слова разделены дефисами.

Для того, чтобы избежать конфликтов имён в разных операционных системах, лучше применять наименее конфликтный способ именования файлов — строчными (маленькими) буквами через дефис.

Форматирование и внешний видСвернуть все

Б12. Неизменяемые значения объявлены через `const`.

При объявлении новых значений предпочтение стоит отдавать использованию ключевого слова `const`. Использовать `let` нужно только в том случае, если значение будет перезаписано.

Неправильно:

```
let a = 1;
let b = 2;
let sum = a + b;
```

Правильно:

```
const a = 1;
const b = 2;
const sum = a + b;

for (let i = 0; i < 42; i++) {
  console.log(i);
}
```

Неправильно:

```
let level = getLevel(this.state.level,
this.quest);
let answerNames = Object.keys(level.answers);
let answers = answerNames.map((key) => ({key,
value: level.answers[key]}));
```

Правильно:

```
const level = getLevel(this.state.level,
this.quest);
```

```
const answerNames =
Object.keys(level.answers);
const answers = answerNames.map((key) =>
({key, value: level.answers[key]}));
```

Б13. 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 строчки.

Неправильно:

```
(() => {
  if (x % 2 === 1) return;
})();
```

Правильно:

```
(() => {
  if (x % 2 === 1) {
    return;
  }
})();
```

Исключения составляют однострочные стрелочные функции, которые можно использовать без обязательных блоков кода:

```
const checkedCheckBoxes =
checkboxes.filter((checkbox) =>
checkbox.checked);
```

Б14. Код всех JS-файлов соответствует рекомендованной структуре.

Рекомендованная структура:

```
// 1. Импорты
import intersection from 'lodash/
intersection';

// 2. Объявление констант
const DEFAULT_COLORS = [`red`, `green`,
`blue`];

// 3. Объявление переменных, значение которых
известно до начала работы программы
const colorPicker =
document.querySelector(`.color-picker`);

// 4. Объявление функций
```

```
const getColorsIntersection = (userColors, defaultColors) => {
  return intersection(userColors, defaultColors);
};

// 5. Код программы. Вызов функций, использование ранее объявленных переменных, объявление класса. Объявление вычисляемых переменных
const rightColors = getColorsIntersection(colorPicker.value, DEFAULT_COLORS);

// 6. Экспорты
export {rightColors};
```

Некоторые блоки могут отсутствовать, но оставшиеся всё равно должны придерживаться порядка.

Не допускается использование экспорта в момент объявления переменной. Исключения:

- прямой экспорт значений по умолчанию;
- модули с утилитарными функциями и константами. В них все значения должны быть

экспортированы с помощью именованного экспорта.

-
-

Б15. Код соответствует гайдлайнам.

- 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.

- **Указания к проверке**

Не возникает ошибок при проверке проекта ESLint: `npm i && npm test`.

-

Б16. Сложные составные константы собираются в перечисления `Enum`.

Множества **однотипных констант** собираются в перечисления.

Неправильно:

```
const COLOR_SUCCESS = `#00FF00`;
```

-
```
const COLOR_WARNING = `#FF9900`;
```
-
```
const COLOR_DANGER = `#FFFF00`;
```

-

  Правильно:

  ```
  const Color = {
    SUCCESS: `#00FF00`,
    WARNING: `#FF9900`,
    DANGER: `#FFFF00`
  };
  ```

  Не стоит путать перечисления с обычными объектами или объектами-неймспейсами, например:

  ```
  const helpers = {
    getRandom() { /* ... */ },
    getSubArray() { /* ... */ },
  };

  const Wizard = {
    width: 10,
    beard: true,
    eyesColor: `blue`
  };
  ```

- Б17. Приватные поля в классах помечены и не используются снаружи.
  Названия методов, которые есть в классе, но не предназначены для внешнего использования, начинаются с нижнего подчёркивания `_`. Доступ к таким полям извне класса запрещён.

Мусор Свернуть все

- Б18. В коде проекта нет файлов и частей кода, которые не используются, включая закомментированные участки кода.

- Б19. Версии используемых зависимостей зафиксированы в `package.json`.
  В списках зависимостей в файле `package.json` указаны точные версии используемых пакетов. Версия обязательно должна быть указана. Не допускается использование `^`, `*` и `~`.

- Б20. В коде нет заранее недостижимых участков кода.
  - Невыполнимые условия:

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

- `}`
-
    - Операции после выхода из функции:
- `(() => {`
- `  return;`
- `  console.log(`This will not happen!`);`
- `})();`
-

Корректность Свернуть все

- Б21. Константы и перечисления нигде в коде не переопределяются.
  Константы и перечисления (`enum`) используются только для чтения и никогда не переопределяются на всём промежутке жизни программы.

- Б22. Используются строгие сравнения вместо нестрогих.
  Вместо операторов нестрогого сравнения `==` и `!=` используются операторы строгого сравнения `===`, `!==`. Таблицы истинности для JavaScript.
  Неправильно:
  `const foo = ``;`

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

Правильно:

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

Б23. 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.

Б24. Отсутствуют потенциально некорректные операции
Например, некорректное сложение двух операндов

как строк. Проблема приоритета конкатенации над сложением.
Неправильно:

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

-

Правильно:

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

-

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

```
const double = (value) => {
  if (!value) {
    return NaN;
  }

  return value * 2;
};

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

-

Потенциально некорректная операция взятия целой части числа.

Неправильно:

```
const minutesNumber = ~~(seconds / 60);
```

Правильно:

```
const minutesNumber = Math.trunc(seconds / 60);
```

Модульность Свернуть все

Б25. Все файлы JS представляют собой отдельные модули ES2015.

Экспорт и импорт значений производится при помощи ключевых слов `export` и `import`. Сохранение в глобальную область видимости значений не допускается.

Пример правильного модуля:

```
import {changeView} from '../util';
import WelcomeView from './welcome-view';
import App from '../main';


export default class Welcome {
  constructor() {
    this.view = new WelcomeView();
  }
```

```
  init() {
    changeView(this.view);

    this.view.onStart = () => {
      App.showGame();
    };
  }
}
```

Б26. Модули не экспортируют изменяющиеся переменные.

Модуль не должен экспортировать переменную, значение которой может измениться в будущем.

Неправильно:

```
export let latestResult;
```

Правильно:

```
export const latestResult =
loadLatestResult();
```

- Б27. Название модуля соответствует его содержимому.
  Разные логические части кода вынесены в отдельные файлы модулей. Имя модуля должно соответствовать его содержимому. Например, если в модуле лежит класс `GameView`, то и имя модуля должно быть `game-view.js`.

- Б28. Из одного модуля экспортируется не больше одного класса. Класс всегда экспортируется как `default`.

Универсальность Свернуть все

- Б29. 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 в браузерах Chrome, Firefox, Safari, Microsoft Edge.
  Указания к проверке
  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, которые предоставляют поставщики, а не в тех, которые установлены в данный момент на компьютере проверяющего.
Важно: для пользователей Windows последняя версия браузера Safari — 5, а у всех остальных — 9, поэтому проводить тестирование на Windows не надо. IE не поддерживается, только Edge.

Магия Свернуть все

- Б30. Нельзя пользоваться глобальной переменной `event`.
Приводит к неосознанному коду:
```
const elem = document.querySelector(`.test`);

const onElemClick = () => {
  event.target.innerText = `you really need event`;
};

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

- Б31. В коде не используются «магические значения», под каждое из них заведена отдельная переменная, названная как константа.

Оптимальность Свернуть все

- Б32. Своевременный выход из цикла: цикл не работает дольше, чем нужно.

Неправильно:

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

Правильно:

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

- Б33. Внутри шаблонов-строк (template literals) не используется конкатенация строк.

Конкатенация строк в шаблонных строках является антипаттерном, так как ухудшает читаемость шаблонной строки.

Неправильно:

```
const page = `${header + `\n` + main + `\n` + footer}`;
```

Правильно:

```
const page = `${header}\n${main}\n${footer}`;
```

Не забывайте, что внутри шаблонных строк можно использовать обратные апострофы:

```
const genreMarkup = `${genres.length > 1 ? `Genres` : `Genre`}`;
```

Б34. Количество вызовов циклов минимизировано.

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

Неправильно:

```
const wizardNames = source.
    map((it) => it.wizard).
    map((it) => it.name);
```

- Правильно:
```
const wizardNames = source.map((it) =>
it.wizard.name);
```
- Допускается использовать цепочку разных методов.
Например:
```
[].map(() => {}).filter(() => {});
```
- Б35. Множественные DOM-операции производятся на элементах, которые не добавлены в DOM.
Например, наполнение скопированного из шаблона элемента данными.

Безопасность Свернуть все

- Б36. 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 так далее. Поэтому поведение должно быть **явным** — если в этот момент времени обработчики не нужны, их нужно удалить. Явное и предсказуемое поведение.

- Б37. Запрещено вставлять в `innerHTML` и подобные ему свойства и методы строки, полученные снаружи (пользовательский ввод, данные сервера), без применения экранирования.

Защита от XSS-атак, а также изменения исходных данных, запутывание пользователя и прочее. Перед вставкой необходимо провести экранирование, например, с помощью DOMPurify.

Неправильно: через `innerHTML` вставляются данные, которые невозможно полностью контролировать, без предварительного экранирования. Это может быть пользовательский ввод, который может

содержать XSS.

```
const listItem =
listItemTemplate.cloneNode(true);
```

- `listItem.querySelector(`.title`).innerHTML = user.fullName;`
-

Правильно: проводить экранирование при вставке внешних данных.

```
const listItem =
listItemTemplate.cloneNode(true);
```

- `listItem.querySelector(`.title`).innerHTML = DOMPurify.sanitize(user.fullName);`
-

Правильно: вставлять данные, которые полностью созданы программистом.

```
const listItemTemplate = `<li
class="amenity><i></i><a href="#"></a></li>`;
```

- `list.innerHTML = listItemTemplate;`

ДОПОЛНИТЕЛЬНЫЕ

Задача Свернуть все

- Д1. Техническое задание реализовано в полном объёме.
  Все обязательные и необязательные пункты технического задания выполнены.

Именование Свернуть все

- Д2. Переменные носят абстрактные названия и не содержат имён собственных.
  Неправильно:

```
const keks = {
  name: `Кекс`
};
```

Правильно:

```
const cat = {
  name: `Кекс`
};
```

- Д3. Название методов и свойств объектов не содержит название объектов.

Неправильно:

```
const popup = {
  openPopup() {
    console.log(`I will open popup`);
  }
};

class Wizard {
  constructor(name = `Пендальф`) {
    this.wizardName = name;
  }
}
```

Правильно

```
const popup = {
  open() {
    console.log(`I will open popup`);
  }
};

class Wizard {
  constructor(name = `Пендальф`) {
    this.name = name;
  }
}
```

-

- Д4. Из названия обработчика события и функции-колбэка следует, что это за 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:
- ```
  const onSidebarClick = () => {};
  ```
- ```
  const onContentLoad = () => {};
  ```
-
- ```
  const onResize = () => {};
  ```
-

  - (на каком элементе) + что случилось + `Handler`:
- ```
  const sidebarClickHandler = () => {};
  ```
- ```
  const contentLoadHandler = () => {};
  ```
-
- ```
  const resizeHandler = () => {};
  ```
-

Единообразие Свернуть все

- Д5. Все функции объявлены единообразно.
  При объявлении функций используются только стрелочные функции. Для объявления методов объектов используется специальный синтаксис для методов. Для объявления классов используется ключевое слово `class`. Смешение стилей в рамках проекта не допускается.
  Функция:

```
const getTheMeaningOfLive = () => {
  return 42;
};
```

  или

```
const getTheMeaningOfLive = function () {
  return 42;
};
```

  Метод:

```
const GOD = {
  createWorld() {
    return `Your world is ready!`;
  }
};
```

Конструктор:

```
class Planet {
  constructor(weight, mass) {
    this.weight = weight;
    this.mass = mass;
  }
}
```

Указания к проверке

Использование объявления функций через `function` допускается, но не рекомендуется, так как все возможные случаи использования контекстных функций решаются при помощи синтаксиса для методов и классов.

Д6. Используется единый стиль именования переменных.

Стиль именования переменных сохраняется во всех модулях, например:

- не следует мешать обработчики содержащие `Handler` и `on`;

- если есть переменные, которые хранят DOM-элемент и содержат слово `Element`, то это правило работает везде.

- Неправильно:

```
const popupMainElement =
document.querySelector(`.popup`);
```

- ```
  const sidebarNode =
  document.querySelector(`.sidebar`);
  ```
- ```
  const similarContainer =
  popupMainElement.querySelector(`ul.similar`);
  ```
-

Правильно:

```
const popupMainElement =
document.querySelector(`.popup`);
```

- ```
  const sidebarElement =
  document.querySelector(`.sidebar`);
  ```
- ```
  const similarContainerElement =
  popupMainElement.querySelector(`ul.similar`);
  ```
-

-

Д7. 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:

```
const popupMainElement = document.querySelector(`#popup`);
const sidebarElement = document.getElementById(`sidebar`);

const popupClassName = popupMainElement.getAttribute(`class`);
const sidebarClassName = sidebarElement.className;
```

Правильно:

```
const popupMainElement = document.querySelector(`#popup`);
const sidebarElement = document.querySelector(`#sidebar`);

const popupClassName = popupMainElement.getAttribute(`class`);
const sidebarClassName = sidebarElement.getAttribute(`class`);
```

-

  или

  ```
  const popupMainElement =
  document.getElementById(`popup`);
  ```
- ```
  const sidebarElement =
  document.getElementById(`sidebar`);
  ```
-

  ```
  const popupClassName =
  popupMainElement.className;
  ```
- ```
  const sidebarClassName =
  sidebarElement.className;
  ```
-
-
-

Д8. Методы внутри классов упорядочены.

Во всех классах методы упорядочены следующим образом:

- Конструктор.
- Геттеры и сеттеры свойств класса.
- Основные методы класса:
- перегруженные методы родительского класса.

- методы класса;
- приватные методы;
- 4. Обработчики событий.
- 5. Статические методы.

- Сортировка основных методов объекта свободная, подразумевается что методы будут расположены оптимально для конкретного класса. Нет смысла ограничивать порядок, потому что он может меняться в зависимости от особенностей объекта.

Модульность Свернуть все

- Д9. 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 работы с DOM:

```
export const createElement = (template) => {
  const outer = document.createElement(`div`);
  outer.innerHTML = template;
  return outer;
};
```

```
const main = document.getElementById(`main`);

export const changeView = (view) => {
  main.innerHTML = ``;
  main.appendChild(view.element);
};
```

Не стоит выносить в отдельный модуль одну повторяющуюся инструкцию:

```
export const createElement = (template) => {
  const outer = document.createElement(`div`);
  outer.innerHTML = template;
  return outer;
};
```

Избыточность Свернуть все

Д10. 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:

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

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

Правильно:

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

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

Д11. Отсутствует дублирование кода: повторяющиеся части кода переписаны как функции или вынесены из условий.

При написании кода следует придерживаться принципа DRY.

Неправильно:

```
if (this.level >= 10) {
  this.timer.stopTimer();
  this.timer.stopTimeout();
  this.setResult();
  removeTimer();
} else if (this.lives <= 0) {
  this.timer.stopTimer();
  this.timer.stopTimeout();
  app.showResultFail();
  removeTimer();
}
```

Правильно:

```
this.timer.stopTimer();
this.timer.stopTimeout();

if (this.level >= 10) {
  this.setResult();
} else if (this.lives <= 0) {
  app.showResultFail();
}

removeTimer();
```

Д12. Если при использовании условного оператора в любом случае возвращается значение, альтернативная ветка опускается.

Неправильно:

```
const getValue = (val, anotherVal) => {
  if (2 > 1) {
    return val;
  } else {
    return anotherVal;
  }
};
```

Правильно:

```
const getValue = (val, anotherVal) => {
  if (2 > 1) {
    return val;
  }

  return anotherVal;
};
```

- Д13. Отсутствуют лишние приведения и проверки типов.
  Если заранее известно, что в переменной число, то нет смысла превращать переменную в число `parseInt(myNumber)`. То же касается и избыточной проверки булевой переменной.
  Неправильно:

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

  Правильно:

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

- Д14. Там, где возможно, в присвоении значения вместо if используется тернарный оператор.
  Неправильно:

```
let sex;
if (male) {
  sex = `Мужчина`;
```

```
} else {
  sex = `Женщина`;
}
```

Правильно:

```
const sex = male ? `Мужчина` : `Женщина`;
```

Д15. Условия упрощены.

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

Неправильно:

```
((firstValue, secondValue) => {
  if (firstValue === secondValue) {
    return true;
  } else {
    return false;
  }
});
```

Правильно:

```
((firstValue, secondValue) => {
  return firstValue === secondValue;
```

- });
-

Оптимальность Свернуть все

-

Д16. Значения не конвертируются в строку и обратно без необходимости.

Если состояние переменной можно сохранить в переменную или во внутреннее свойство, то лучше использовать внутреннее состояние объекта, вместо сериализации из значения в строку и наоборот.

Неправильно:

```
class Timer {
  setTime({minutes, seconds}) {
    document.querySelector(`.timer-value-mins`).textContent = minutes;
    document.querySelector(`.timer-value-secs`).textContent = seconds;
  }
  getTime() {
    const minutes = parseInt(document.querySelector(`.timer-value-mins`).textContent, 10);
    const seconds = parseInt(document.querySelector(`.timer-value-secs`).textContent, 10);
```

```
    return {minutes, seconds};
  }
}
```

Правильно:

```
class Timer {
  constructor(time) {
    this.minutesEl = document.querySelector(`.timer-value-mins`);
    this.secondsEl = document.querySelector(`.timer-value-secs`);
    this.time = time;
  }

  update() {
    this.minutesEl.textContent = this.time.minutes;
    this.secondsEl.textContent = this.time.seconds;
  }

  get time() {
    return this.myTime;
  }
```

```
  set time(time) {
    this.myTime = time;
    this.update();
  }
}
```

Д17. Константы, используемые внутри функций, создаются вне функций и используются повторно через замыкания.

Д18. Поиск элементов по селекторам делается минимальное количество раз, после этого ссылки на элементы сохраняются.

Неправильно:

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

Правильно:

```
const dialog =
document.querySelector(`.dialog`);
```

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

Д19. Массивы и объекты, содержимое которых вычисляется, собираются один раз, а после этого только переиспользуются.

Д20. Для итерирования по массивам и структурам данных, по которому можно итерироваться, (**Iterable**) используется конструкция `for .. of`.
Там, где не требуется индекс элемента массива или нужно обойти все элементы итерируемой структуры данных, используется цикл `for .. of` вместо цикла `for`.
Неправильно:

```
for (let i = 0; i < levels.length; i++) {
  const level = levels[i];
  renderLevel(level);
}
```

Правильно:

```
for (const level of levels) {
  renderLevel(level);
}
```

Д21. Изменения применяются точечно

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:

```
const imageContainer = document.querySelector(`.image-container`);

const changeFilter = (filterName) => {
  imageContainer.classList.remove(`filter-chrome`, `filter-sepia`, `filter-marvin`, `filter-phobos`, `filter-heat`);
  imageContainer.classList.add(filterName);
};
```

Правильно:

```
const imageContainer =
document.querySelector(`.image-container`);
```

```
let currentFilter;
const changeFilter = (filterName) => {
  if (currentFilter) {
imageContainer.classList.remove(currentFilter)
;
  }
  imageContainer.classList.add(filterName);
  currentFilter = filterName;
};
```

Сложность и читаемость Свернуть все

Д22. Для каждого события используется отдельный обработчик.

Одна функция не является обработчиком нескольких разных событий.

- Д23. Длинные функции и методы разбиты на несколько небольших.

- Д24. 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((el) => {
  el.onclick = () => {
    console.log(el);
  };
});
```

- Д25. Оператор присваивания не используется как часть выражения.
Неправильно:

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

Правильно:

```
picArray = JSON.parse(images);
```

- `imgGenerate(picArray);`
-
-

Д26. Операции над DOM-элементами инкапсулированы.

Все операции над элементами DOM-дерева происходят только там, где эти элементы были созданы, и не используются снаружи. Например, всё, что связано с отрисовкой данных, должно находиться внутри класса `View` и управляться только внутри этого класса. Любой доступ к закрытым данным снаружи запрещён.

Неправильно:

```
class PlayerController {
  constructor(view) {
    this.view = view;
  }

  init() {
    const checkboxes = Array.from(this.view.element.querySelectorAll(`input`));

    const answers = [];

    this.view.makeDecision = () => {
```

```
      answers.push(checkboxes.filter((it) => it.checked));

      if (answers.length > 0) {
        goToNextScreen();
      }
    };
  }
}
```

Правильно:

```
class PlayerController {
  constructor(view) {
    this.view = view;
  }

  init() {
    this.view.onAnswer = (answers) => {
      if (answers.length > 0) {
        goToNextScreen();
      }
    };
  }
}
```