

Раздел

[Качество кода](#)

Навигация по уроку

Синтаксис

Размещение функций

Руководства по стилю кода


Автоматизированные
средства проверки
(линтеры)

Итого

Задачи (1)

Комментарии

Поделиться

[Редактировать на GitHub](#)[→ Язык программирования JavaScript](#)
[→ Качество кода](#) 9-го сентября 2020

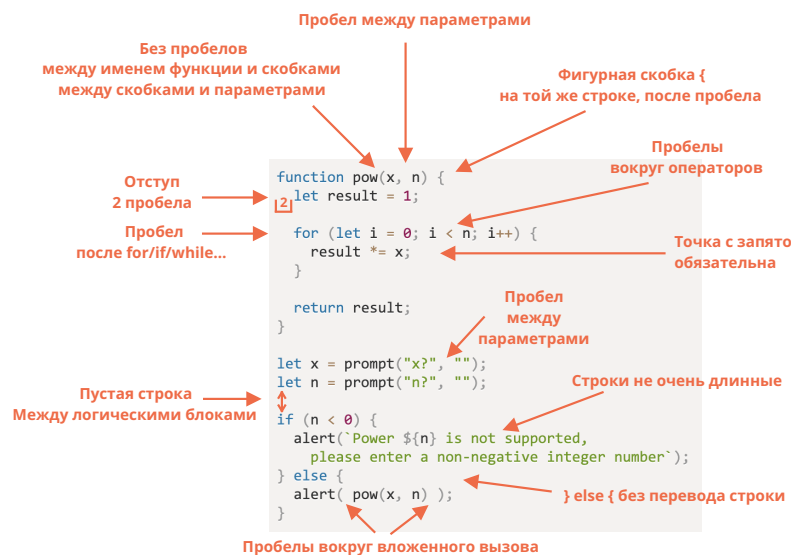
Советы по стилю кода

Код должен быть максимально читаемым и понятным.

Это и есть искусство программирования – взять сложную задачу и написать такой код для её решения, который и правильно работает, и легко читается, понятен для людей. Для этого нужен *хороший стиль* написания кода. В этой главе мы рассмотрим составляющие такого стиля.

Синтаксис

Шпаргалка с правилами синтаксиса (подробнее смотрите ниже по тексту):



Не всё здесь однозначно, так что разберём эти правила подробнее.



Ни одно правило не является жёстко обязательным

Здесь нет железных правил. Это стилевые предпочтения, а не религиозные догмы.

Фигурные скобки

В большинстве JavaScript проектов фигурные скобки пишутся в так называемом «египетском» стиле с открывающей скобкой на той же строке, что и соответствующее ключевое слово – не на новой строке. Перед открывающей скобкой должен быть пробел, как здесь:

```
1 if (condition) {
2     // делай это
3     // ...и это
4     // ...и потом это
5 }
```

А что если у нас однострочная запись, типа `if (condition) doSomething()`, должны ли мы использовать фигурные скобки?

Вот различные варианты расстановки скобок с комментариями, посмотрите сами, какой вам кажется самым читаемым:

Раздел

Качество кода

Навигация по уроку

Синтаксис

Размещение функций

Руководства по стилю кода

Автоматизированные
средства проверки
(линтеры)

Итого

Задачи (1)

Комментарии

Поделиться



Редактировать на GitHub



1. 😞 Такое иногда бывает в коде начинающих. Плохо, фигурные скобки не нужны:

```
1 if (n < 0) {alert(`Степень ${n} не поддерживается`);}
```

2. 😞 Никогда не разделяйте строки без фигурных скобок, можно ненароком сделать ошибку при добавлении строк:

```
1 if (n < 0)
2   alert(`Степень ${n} не поддерживается`);
```

3. 😞 В одну строку без скобок – приемлемо, если эта строка короткая:

```
1 if (n < 0) alert(`Степень ${n} не поддерживается`);
```

4. 😊 Самый лучший вариант:

```
1 if (n < 0) {
2   alert(`Степень ${n} не поддерживается`);
3 }
```

Для очень короткого кода допустима одна строка. Например: `if (cond) return null`. Но блок кода (последний вариант) обычно всё равно читается лучше.

Длина строки

Никто не любит читать длинные горизонтальные строки кода. Лучше всего разбивать их, например:

```
1 // обратные кавычки ` позволяют разделять строку на части
2 let str = `
3   Рабочая группа TC39 организации Ecma International -
4   это группа JavaScript-разработчиков, теоретиков и авт
5   которые вместе с сообществом занимаются поддержкой и
6 `;
```

Или для `if`:

```
1 if (
2   id === 123 &&
3   moonPhase === 'Waning Gibbous' &&
4   zodiacSign === 'Libra'
5 ) {
6   letTheSorceryBegin();
7 }
```

Максимальную длину строки согласовывают в команде. Обычно это 80 или 120 символов.

Отступы

Существует два типа отступов:

- **Горизонтальные отступы: два или четыре пробела.**

Горизонтальный отступ выполняется с помощью 2 или 4 пробелов, или символа табуляции (клавиша `Tab`). Какой из них выбрать – это уже на ваше усмотрение. Пробелы больше распространены.

Одно из преимуществ пробелов над табуляцией заключается в том, что пробелы допускают более гибкие конфигурации отступов, чем символ

Раздел

Качество кода

Навигация по уроку

Синтаксис

Размещение функций

Руководства по стилю кода

Автоматизированные
средства проверки
(линтеры)

Итого

Задачи (1)

Комментарии

Поделиться



Редактировать на GitHub



табуляции.

Например, мы можем выравнивать аргументы относительно открывающей скобки:

```
1 show(parameters,
2     aligned, // 5 пробелов слева
3     one,
4     after,
5     another
6 ) {
7     // ...
8 }
```

- **Вертикальные отступы: пустые строки для разбивки кода на «логические блоки».**

Даже одну функцию часто можно разделить на логические блоки. В примере ниже разделены инициализация переменных, основной цикл и возвращаемый результат:

```
1 function pow(x, n) {
2     let result = 1;
3     //      <--
4     for (let i = 0; i < n; i++) {
5         result *= x;
6     }
7     //      <--
8     return result;
9 }
```

Вставляйте дополнительный перевод строки туда, где это сделает код более читаемым. Не должно быть более 9 строк кода подряд без вертикального отступа.



Точка с запятой

Точки с запятой должны присутствовать после каждого выражения, даже если их, казалось бы, можно пропустить.

Есть языки, в которых точка с запятой необязательна и редко используется. Однако в JavaScript бывают случаи, когда перенос строки не интерпретируется, как точка с запятой, что может привести к ошибкам. Подробнее об этом – в главе [о структуре кода](#).

Если вы – опытный разработчик на JavaScript, то можно выбрать стиль кода без точек с запятой, например [StandardJS](#). В ином случае, лучше будет использовать точки с запятой, чтобы избежать подводных камней. Большинство разработчиков их ставят.

Уровни вложенности

Уровней вложенности должно быть немного.

Например, в цикле бывает полезно использовать директиву `continue`, чтобы избежать лишней вложенности.

Например, вместо добавления вложенного условия `if`, как здесь:

```
1 for (let i = 0; i < 10; i++) {
2     if (cond) {
3         ... // <- ещё один уровень вложенности
4     }
5 }
```

Мы можем написать:

Раздел

Качество кода

Навигация по уроку

Синтаксис

Размещение функций

Руководства по стилю кода

Автоматизированные
средства проверки
(линтеры)

Итого

Задачи (1)

Комментарии

Поделиться



Редактировать на GitHub



```
1 for (let i = 0; i < 10; i++) {  
2   if (!cond) continue;  
3   ... // <- нет лишнего уровня вложенности  
4 }
```

Аналогичная ситуация – с if/else и return.

Например, две нижеследующие конструкции идентичны.

Первая:

```
1 function pow(x, n) {  
2   if (n < 0) {  
3     alert("Отрицательные значения 'n' не поддерживаются");  
4   } else {  
5     let result = 1;  
6  
7     for (let i = 0; i < n; i++) {  
8       result *= x;  
9     }  
10  
11    return result;  
12  }  
13 }
```

Вторая:

```
1 function pow(x, n) {  
2   if (n < 0) {  
3     alert("Отрицательные значения 'n' не поддерживаются");  
4     return;  
5   }  
6  
7   let result = 1;  
8  
9   for (let i = 0; i < n; i++) {  
10    result *= x;  
11  }  
12  
13  return result;  
14 }
```

Второй вариант является более читабельным, потому что «особый случай» `n < 0` обрабатывается на ранней стадии. После проверки можно переходить к «основному» потоку кода без необходимости увеличения вложенности.

Размещение функций

Если вы пишете несколько вспомогательных функций, а затем используемый ими код, то существует три способа организации функций.

1. Объявить функции *перед* кодом, который их вызовет:

```
1 // объявление функций  
2 function createElement() {  
3   ...  
4 }  
5  
6 function setHandler(elem) {  
7   ...  
8 }  
9  
10 function walkAround() {  
11   ...  
12 }  
13
```

Раздел

Качество кода

Навигация по уроку

Синтаксис

Размещение функций

Руководства по стилю кода

Автоматизированные
средства проверки
(линтеры)

Итого

Задачи (1)

Комментарии

Поделиться



Редактировать на GitHub



```
14 // код, который использует их
15 let elem = createElement();
16 setHandler(elem);
17 walkAround();
```

2. Сначала код, затем функции

```
1 // код, использующий функции
2 let elem = createElement();
3 setHandler(elem);
4 walkAround();
5
6 // --- вспомогательные функции ---
7 function createElement() {
8   ...
9 }
10
11 function setHandler(elem) {
12   ...
13 }
14
15 function walkAround() {
16   ...
17 }
```

3. Смешанный стиль: функция объявляется там, где она используется впервые.

В большинстве случаев второй вариант является предпочтительным.

Это потому, что при чтении кода мы сначала хотим знать, что *он делает*. Если сначала идёт код, то это тут же становится понятно. И тогда, может быть, нам вообще не нужно будет читать функции, особенно если их имена хорошо подобраны.

Руководства по стилю кода

Руководство по стилю содержит общие правила о том, как писать код, например: какие кавычки использовать, сколько пробелов отступать, максимальную длину строки и так далее – в общем, множество мелочей.

Когда все участники команды используют одно и то же руководство по стилю, код выглядит одинаково, независимо от того, кто из команды его написал.

Конечно, команда всегда может написать собственное руководство по стилю, но обычно в этом нет необходимости. Существует множество уже готовых.

Некоторые популярные руководства:

- [Google JavaScript Style Guide](#)
- [Airbnb JavaScript Style Guide](#) (есть [перевод](#))
- [Idiomatic.JS](#) (есть [перевод](#))
- [StandardJS](#)
- (и ещё множество других)

Если вы – начинающий разработчик, то начните со шпаргалки в начале этой главы. Как только вы освоитесь, просмотрите другие руководства, чтобы выбрать общие принципы и решить, какое вам больше подходит.

Автоматизированные средства проверки (линтеры)

Автоматизированные средства проверки, так называемые «линтеры» – это инструменты, которые могут автоматически проверять стиль вашего кода и вносить предложения по его улучшению.

Самое замечательное в них то, что проверка стиля может также найти программные ошибки, такие как опечатки в именах переменных или

Раздел

Качество кода

Навигация по уроку

Синтаксис

Размещение функций

Руководства по стилю кода

Автоматизированные
средства проверки
(линтеры)

Итого

Задачи (1)

Комментарии

Поделиться



Редактировать на GitHub



функций. Из-за этой особенности использовать линтер рекомендуется, даже если вы не хотите придерживаться какого-то конкретного «стиля кода».

Вот некоторые известные инструменты для проверки:

- **JSLint** – проверяет код на соответствие [стилю JSLint](#), в онлайн-интерфейсе вверху можно ввести код, а внизу – различные настройки проверки, чтобы попробовать её в действии.
- **JSHint** – больше проверок, чем в JSLint.
- **ESLint** – пожалуй, самый современный линтер.

Все они, в общем-то, работают. Автор пользуется [ESLint](#).

Большинство линтеров интегрированы со многими популярными редакторами: просто включите плагин в редакторе и настройте стиль.

Например, для ESLint вы должны выполнить следующее:

1. Установите [Node.JS](#).
2. Установите ESLint с помощью команды `npm install -g eslint` (npm – установщик пакетов JavaScript).
3. Создайте файл конфигурации с именем `.eslintrc` в корне вашего JavaScript-проекта (в папке, содержащей все ваши файлы).
4. Установите/включите плагин для вашего редактора, который интегрируется с ESLint. У большинства редакторов он есть.

Вот пример файла `.eslintrc`:

```
1 {
2   "extends": "eslint:recommended",
3   "env": {
4     "browser": true,
5     "node": true,
6     "es6": true
7   },
8   "rules": {
9     "no-console": 0,
10    "indent": ["warning", 2]
11  }
12 }
```

Здесь директива `"extends"` означает, что конфигурация основана на наборе настроек `«eslint:recommended»`. После этого мы уточняем наши собственные.

Кроме того, возможно загрузить наборы правил стиля из сети и расширить их. Смотрите <https://eslint.org/docs/user-guide/getting-started> для получения более подробной информации об установке.

Также некоторые среды разработки имеют встроенные линтеры, возможно, удобные, но не такие гибкие в настройке, как ESLint.

Итого

Все правила синтаксиса, описанные в этой главе (и в ссылках на руководства по стилю), направлены на повышение читаемости вашего кода. О любых можно поспорить.

Когда мы думаем о написании «лучшего» кода, мы должны задать себе вопросы: «Что сделает код более читаемым и лёгким для понимания?» и «Что может помочь избежать ошибок?». Это – основные моменты, о которых следует помнить при выборе и обсуждении стилей кода.

Чтение популярных руководств по стилю позволит вам быть в курсе лучших практик и последних идей и тенденций в стилях написания кода.

✓ Задачи

Плохой стиль

важность: 4

Какие недостатки вы видите в стиле написания кода этого примера?

Раздел

[Качество кода](#)

Навигация по уроку

Синтаксис

Размещение функций

Руководства по стилю кода

Автоматизированные
средства проверки
(линтеры)

Итого

Задачи (1)

Комментарии

Поделиться



[Редактировать на GitHub](#)



```
1 function pow(x,n)
2 {
3   let result=1;
4   for(let i=0;i<n;i++) {result*=x;}
5   return result;
6 }
7
8 let x=prompt("x?",""), n=prompt("n?","")
9 if (n<=0)
10 {
11   alert(`Степень ${n} не поддерживается, введите целую
12 }
13 else
14 {
15   alert(pow(x,n))
16 }
```

решение

Проводим [курсы по JavaScript и фреймворкам](#).



Комментарии

перед тем как писать...