

[🏠](#) → [Язык JavaScript](#) → [Качество кода](#)

📅 1 января 2023 г.

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):

Пробел между параметрами

Без пробелов между именем функции и скобками между скобками и параметрами

Фигурная скобка { на той же строке, после пробела

Пробелы вокруг операторов

Отступ 2 пробела

Пробел после for/if/while...

Точка с запятой обязательна

Пробел между аргументами

Пустая строка Между логическими блоками

Строки не очень длинные

Пробелы вокруг вложенного вызова

```
function pow(x, n) {  
  let result = 1;  
  for (let i = 0; i < n; i++) {  
    result *= x;  
  }  
  return result;  
}  
  
let x = prompt("x?", "");  
let n = prompt("n?", "");  
if (n < 0) {  
  alert(`Power ${n} is not supported,  
    please enter a non-negative integer number`);  
} else {  
  alert( pow(x, n) );  
}
```

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:

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 авторов движков JavaScript,
5   которые вместе с сообществом занимаются поддержкой и развитием языка JavaScri
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: 2 или 4 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 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  }
```

```
9     return result;
    }
```

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:

```
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
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 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 делает 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?

```
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 степень, большую 0`);
12 }
13 else
14 {
15   alert(pow(x,n))
16 }
```

решение



Вы могли заметить следующие недостатки, сверху вниз:

```
1 function pow(x,n) // <- отсутствует пробел между аргументами
2 { // <- фигурная скобка на отдельной строке
3   let result=1; // <- нет пробелов вокруг знака =
4   for(let i=0;i<n;i++) {result*=x;} // <- нет пробелов
5   // содержимое скобок { ... } лучше вынести на отдельную строку
6   return result;
7 }
8
9 let x=prompt("x?",""), n=prompt("n?","") // <-- технически допустимо,
10 // но лучше написать в 2 строки, также нет пробелов и точки с запятой
11 if (n<=0) // <- нет пробелов, стоит добавить отступ в одну строку вверх
12 { // <- фигурная скобка на отдельной строке
13   // ниже - слишком длинная строка, лучше разбить для улучшения читаемос
14   alert(`Степень ${n} не поддерживается, введите целую степень, большую
15 }
16 else // <- можно на одной строке, вместе: "} else {"
17 {
18   alert(pow(x,n)) // вложенный вызов функции, нет пробелов и точки с за
19 }
```


Исправленный вариант:

```
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  }
10
11 let x = prompt("x?", "");
12 let n = prompt("n?", "");
13
14 if (n <= 0) {
15   alert(`Степень ${n} не поддерживается,
16     введите целую степень, большую 0`);
17 } else {
18   alert( pow(x, n) );
19 }
```



Предыдущий урок

Следующий урок



Поделиться



Карта учебника

Проводим курсы по JavaScript и фреймворкам.



Комментарии

- Если вам кажется, что в статье что-то не так - вместо комментария напишите [на GitHub](#).
- Для одной строки кода используйте тег `<code>`, для нескольких строк кода — тег `<pre>`, если больше 10 строк — ссылку на песочницу ([plnkr](#), [JSBin](#), [codepen...](#))
- Если что-то непонятно в статье — пишите, что именно и с какого места.

Присоединиться к обсуждению...

ВОЙТИ С ПОМОЩЬЮ

ИЛИ ЧЕРЕЗ DISQUS ?

Имя

♡ 54

Поделиться

Лучшие Новые Старые



duott

месяц назад

В этой главе просится упоминание о том, что все эти правила не надо расставлять и соблюдать руками, достаточно настроить Prettier соответствующим образом (или довериться автоматическим настройкам)

0 0 Ответить • Поделиться ›

P

Paul Pumphut

2 месяца назад

Не вижу недостатков, какая-то вкусовщина все это...

0 9 Ответить • Поделиться ›

X

хлорка

2 месяца назад edited

13.12.2022 9:20/очевидно все

0 0 Ответить • Поделиться ›



Alexander Mandrov

3 месяца назад

Обстоятельства и желания сподвигли меня на создание собственного канала. Там я рассказываю о жизни и работе, о том, как стал программистом в 20 лет, коротко и по делу.

ТГ @unsleeping706

0 6 Ответить • Поделиться ›



Mikhail Vinogradov

4 месяца назад edited

⚡ Ну чё, народ, погнажи? ⚡

Реально ли **изучить javascript за 7 месяцев** и трудоустроиться?

Вот я решил проверить и веду свой блог **Джаваскриптизёр** на ютубе, где буду выкладывать видео каждую неделю на протяжении 7 месяцев.

Если тебе тоже интересен джаваскрипт, присоединяйся:

ТГ: @javascriptzerr

Ютуб: Джаваскриптизёр

P

Paul Pumphut

→ Mikhail Vinogradov

2 месяца назад

—



Последнее видео месяц назад, что пошло не так?

0

1

Ответить • Поделиться ›

R

Rowjas

5 месяцев назад

—



Почему фигурная скобка на отдельной строке - плохо? Это ведь повышает читаемость кода и например в других языках так считается правильно.

2

0

Ответить • Поделиться ›

N

nastyaroa

→ Rowjas

месяц назад

—



мне кажется это вкусовщина, мне наоборот очень тяжело воспринимать фигурную скобку на отдельной строке

0

0

Ответить • Поделиться ›

H

Никита Самарин

6 месяцев назад

—



13.08.22/it's cool

0

1

Ответить • Поделиться ›

F

Fufuha Gagunskaya

7 месяцев назад

—



04.08.22

0

1

Ответить • Поделиться ›

V

Viktor

7 месяцев назад

—



22.07.2022 9:42 На урааа

1

3

Ответить • Поделиться ›

C

Слава

→ Viktor

7 месяцев назад

—



тупой

1

2

Ответить • Поделиться ›

V

Viktor

→ Слава

7 месяцев назад

—



0 0 Ответить • Поделиться ›



Uliwe

→ Слава

7 месяцев назад edited

Хахахахаха, до него не дойдет, он блять бревно

1 0 Ответить • Поделиться ›



`wrongju

7 месяцев назад

18.07.2022 ок пон, очевидно

0 0 Ответить • Поделиться ›



Ilya

7 месяцев назад edited

Есть вопрос по существу:

В правильном варианте кода для разбиения слишком длинной строки используется перевод строки внутри неё самой, то есть автор нажал Enter, затем Tab. Я скопировал, проверил и увидел тот же синтаксис, но уже в самом alert'e. Предложу свой вариант кода, где строка в коде разбивается на несколько, а в представлении выглядит целой.

Есть мысли, предложения? Пишите!

Мой вариант:

```
let x = +prompt('Введите x', '');
let n = +prompt('Введите n', '');

if (n <= 0) {
  alert (`Степень ${n} не поддерживается, ` +
    `введите целую степень, большую 0`);
} else {
  alert ( pow(x, n) );
}
```

[показать больше](#)

3 0 Ответить • Поделиться ›



Валерий Языков

→ Ilya

2 месяца назад

Нет, конкатенация здесь лишняя и имхо была бы оправдана в случае вставки переменной.

0 0 Ответить • Поделиться ›



Кирилл

→ Ilya

7 месяцев назад

Тоже вариант, но я бы еще добавил: при переносе строки добавлять tab (2 или 4 пробела)

2 0 Ответить • Поделиться ›



Кирилл



Pavel Zabein

7 месяцев назад



Просто поставьте Prettier....

0 0 Ответить • Поделиться ›



0

Out of the way

7 месяцев назад

07.07.2022 0:21AM

0 0 Ответить • Поделиться ›

Загрузить ещё комментарии

Подписаться

Privacy

Не продавайте мои данные