


 → [Язык JavaScript](#) → [Основы JavaScript](#)

 21 января 2023 г.

# Функции

Зачастую нам надо повторять одно и то же действие во многих частях программы.

Например, необходимо красиво вывести сообщение при приветствии посетителя, при выходе посетителя с сайта, ещё где-нибудь.

Чтобы не повторять один и тот же код во многих местах, придуманы функции. Функции являются основными «строительными блоками» программы.

Примеры встроенных функций вы уже видели – это `alert(message)`, `prompt(message, default)` и `confirm(question)`. Но можно создавать и свои.

## Объявление функции

Для создания функций мы можем использовать *объявление функции*.

Пример объявления функции:

```

1 function showMessage() {
2   alert( 'Всем привет!' );
3 }
```

Вначале идёт ключевое слово `function`, после него *имя функции*, затем список *параметров* в круглых скобках через запятую (в вышеприведённом примере он пустой) и, наконец, код функции, также называемый «телом функции», внутри фигурных скобок.

```

1 function имя(параметры) {
2   ...тело...
3 }
```

Наша новая функция может быть вызвана по своему имени: `showMessage()`.

Например:

```

1 function showMessage() {
2   alert( 'Всем привет!' );
3 }
4
5 showMessage();
6 showMessage();
```



Вызов `showMessage()` выполняет код функции. Здесь мы увидим сообщение дважды.

Этот пример явно демонстрирует одно из главных предназначений функций: избавление от дублирования кода.

Если понадобится поменять сообщение или способ его вывода – достаточно изменить его в одном месте: в функции, которая его выводит.

## Локальные переменные

Переменные, объявленные внутри функции, видны только внутри этой функции.

Например:

```
1 function showMessage() {  
2     let message = "Привет, я JavaScript!"; // локальная переменная  
3  
4     alert( message );  
5 }  
6  
7 showMessage(); // Привет, я JavaScript!  
8  
9 alert( message ); // <-- будет ошибка, т.к. переменная видна только внутри фу-
```

## Внешние переменные

У функции есть доступ к внешним переменным, например:

```
1 let userName = 'Вася';  
2  
3 function showMessage() {  
4     let message = 'Привет, ' + userName;  
5     alert(message);  
6 }  
7  
8 showMessage(); // Привет, Вася
```

Функция обладает полным доступом к внешним переменным и может изменять их значение.

Например:

```
1 let userName = 'Вася';  
2  
3 function showMessage() {  
4     userName = "Петя"; // (1) изменяем значение внешней переменной  
5  
6     let message = 'Привет, ' + userName;  
7     alert(message);  
8 }  
9  
10 alert( userName ); // Вася перед вызовом функции  
11  
12 showMessage();  
13  
14
```

```
alert( userName ); // Петя, значение внешней переменной было изменено функцией
```

Внешняя переменная используется, только если внутри функции нет такой локальной.

Если одноимённая переменная объявляется внутри функции, тогда она перекрывает внешнюю. Например, в коде ниже функция использует локальную переменную `userName`. Внешняя будет проигнорирована:

```
1 let userName = 'Вася';
2
3 function showMessage() {
4     let userName = "Петя"; // объявляем локальную переменную
5
6     let message = 'Привет, ' + userName; // Петя
7     alert(message);
8 }
9
10 // функция создаст и будет использовать свою собственную локальную переменную
11 showMessage();
12
13 alert( userName ); // Вася, не изменилась, функция не трогала внешнюю переменную
```

### **i** Глобальные переменные

Переменные, объявленные снаружи всех функций, такие как внешняя переменная `userName` в вышеприведённом коде – называются *глобальными*.

*Глобальные переменные* видимы для любой функции (если только их не перекрывают одноимённые локальные переменные).

Желательно сводить использование глобальных переменных к минимуму. В современном коде обычно мало или совсем нет глобальных переменных. Хотя они иногда полезны для хранения важнейших «общепроектных» данных.

## Параметры

Мы можем передать внутрь функции любую информацию, используя параметры.

В нижеприведённом примере функции передаются два параметра: `from` и `text`.

```
1 function showMessage(from, text) { // параметры: from, text
2     alert(from + ': ' + text);
3 }
4
5 showMessage('Аня', 'Привет!'); // Аня: Привет! (*)
6 showMessage('Аня', "Как дела?"); // Аня: Как дела? (**)
```

Когда функция вызывается в строках `(*)` и `(**)`, переданные значения копируются в локальные переменные `from` и `text`. Затем они используются в теле функции.

Вот ещё один пример: у нас есть переменная `from`, и мы передаём её функции. Обратите внимание: функция изменяет значение `from`, но это изменение не видно снаружи. Функция всегда получает только копию

значения:



```
1 function showMessage(from, text) {
2
3   from = '*' + from + '*'; // немного украсим "from"
4
5   alert( from + ': ' + text );
6 }
7
8 let from = "Аня";
9
10 showMessage(from, "Привет"); // *Аня*: Привет
11
12 // значение "from" осталось прежним, функция изменила значение локальной переменной
13 alert( from ); // Аня
```

Значение, передаваемое в качестве параметра функции, также называется *аргументом*.

Другими словами:

- Параметр – это переменная, указанная в круглых скобках в объявлении функции.
- Аргумент – это значение, которое передаётся функции при её вызове.

Мы объявляем функции со списком параметров, затем вызываем их, передавая аргументы.

Рассматривая приведённый выше пример, мы могли бы сказать: "функция `showMessage` объявляется с двумя параметрами, затем вызывается с двумя аргументами: `from` и `"Привет"`".

## Значения по умолчанию

Если при вызове функции аргумент не был указан, то его значением становится `undefined`.

Например, вышеупомянутая функция `showMessage(from, text)` может быть вызвана с одним аргументом:

```
1 showMessage("Аня");
```

Это не приведёт к ошибке. Такой вызов выведет `"*Аня*: undefined"`. В вызове не указан параметр `text`, поэтому предполагается, что `text === undefined`.

Если мы хотим задать параметру `text` значение по умолчанию, мы должны указать его после `=`:



```
1 function showMessage(from, text = "текст не добавлен") {
2   alert( from + ": " + text );
3 }
4
5 showMessage("Аня"); // Аня: текст не добавлен
```

Теперь, если параметр `text` не указан, его значением будет `"текст не добавлен"`

В данном случае `"текст не добавлен"` это строка, но на её месте могло бы быть и более сложное выражение, которое бы вычислялось и присваивалось при отсутствии параметра. Например:



```
1 function showMessage(from, text = anotherFunction()) {  
2     // anotherFunction() выполнится только если не передан text  
3     // результатом будет значение text  
4 }
```

### **i** Вычисление параметров по умолчанию

В JavaScript параметры по умолчанию вычисляются каждый раз, когда функция вызывается без соответствующего параметра.

В приведённом выше примере, функция `anotherFunction()` не будет вызвана вообще, если указан параметр `text`.

С другой стороны, функция будет независимо вызываться каждый раз, когда `text` отсутствует.

### **i** Использование параметров по умолчанию в ранних версиях JavaScript

Ранние версии JavaScript не поддерживали параметры по умолчанию. Поэтому существуют альтернативные способы, которые могут встречаться в старых скриптах.

Например, явная проверка на `undefined`:

```
1 function showMessage(from, text) {  
2     if (text === undefined) {  
3         text = 'текст не добавлен';  
4     }  
5  
6     alert( from + ": " + text );  
7 }
```

...Или с помощью оператора `||`:

```
1 function showMessage(from, text) {  
2     // Если значение text ложно, тогда присвоить параметру text значение по !  
3     // заметим, что при этом пустая строка text === "" будет также считаться  
4     text = text || 'текст не добавлен';  
5     ...  
6 }
```

## Альтернативные параметры по умолчанию

Иногда имеет смысл присваивать значения по умолчанию для параметров не в объявлении функции, а на более позднем этапе.

Во время выполнения функции мы можем проверить, передан ли параметр, сравнив его с `undefined`:



```
1 function showMessage(text) {  
2     // ...  
3 }
```

```

4   if (text === undefined) { // если параметр отсутствует
5       text = 'пустое сообщение';
6   }
7   alert(text);
8 }
  showMessage(); // пустое сообщение

```

...Или мы можем использовать оператор `||` :

```

1  function showMessage(text) {
2      // если значение text ложно или равняется undefined, тогда присвоить text з-
3      text = text || 'пусто';
4      ...
5  }

```

Современные движки JavaScript поддерживают **оператор нулевого слияния** `??` . Его использование будет лучшей практикой, в случае, если большинство ложных значений, таких как `0` , следует расценивать как «нормальные».

```

1  function showCount(count) {
2      // если count равен undefined или null, показать "неизвестно"
3      alert(count ?? "неизвестно");
4  }
5  showCount(0); // 0
6  showCount(null); // неизвестно
7  showCount(); // неизвестно

```

## Возврат значения

Функция может вернуть результат, который будет передан в вызвавший её код.

Простейшим примером может служить функция сложения двух чисел:

```

1  function sum(a, b) {
2      return a + b;
3  }
4
5  let result = sum(1, 2);
6  alert( result ); // 3

```

Директива **return** может находиться в любом месте тела функции. Как только выполнение доходит до этого места, функция останавливается, и значение возвращается в вызвавший её код (присваивается переменной `result` выше).

Вызовов **return** может быть несколько, например:

```

1  function checkAge(age) {
2      if (age >= 18) {
3          return true;

```

```

4   } else {
5       return confirm('А родители разрешили?');
6   }
7 }
8
9 let age = prompt('Сколько вам лет?', 18);
10
11 if ( checkAge(age) ) {
12     alert( 'Доступ получен' );
13 } else {
14     alert( 'Доступ закрыт' );
15 }

```

Возможно использовать `return` и без значения. Это приведёт к немедленному выходу из функции.

Например:

```

1 function showMovie(age) {
2     if ( !checkAge(age) ) {
3         return;
4     }
5
6     alert( "Вам показывается кино" ); // (*)
7     // ...
8 }

```

В коде выше, если `checkAge(age)` вернёт `false`, `showMovie` не выполнит `alert`.

### **i** Результат функции с пустым `return` или без него – `undefined`

Если функция не возвращает значения, это всё равно, как если бы она возвращала `undefined`:

```

1 function doNothing() { /* пусто */ }
2
3 alert( doNothing() === undefined ); // true

```

Пустой `return` аналогичен `return undefined`:

```

1 function doNothing() {
2     return;
3 }
4
5 alert( doNothing() === undefined ); // true

```



### Никогда не добавляйте перевод строки между `return` и его значением

Для длинного выражения в `return` может быть заманчиво разместить его на нескольких отдельных строках, например так:

```
1 return
2 (some + long + expression + or + whatever * f(a) + f(b))
```

Код не выполнится, потому что интерпретатор JavaScript подставит точку с запятой после `return`. Для него это будет выглядеть так:

```
1 return;
2 (some + long + expression + or + whatever * f(a) + f(b))
```

Таким образом, это фактически стало пустым `return`.

Если мы хотим, чтобы возвращаемое выражение занимало несколько строк, нужно начать его на той же строке, что и `return`. Или, хотя бы, поставить там открывающую скобку, вот так:

```
1 return (
2   some + long + expression
3   + or +
4   whatever * f(a) + f(b)
5 )
```

И тогда всё сработает, как задумано.

## Выбор имени функции

Функция – это действие. Поэтому имя функции обычно является глаголом. Оно должно быть кратким, точным и описывать действие функции, чтобы программист, который будет читать код, получил верное представление о том, что делает функция.

Как правило, используются глагольные префиксы, обозначающие общий характер действия, после которых следует уточнение. Обычно в командах разработчиков действуют соглашения, касающиеся значений этих префиксов.

Например, функции, начинающиеся с `"show"` обычно что-то показывают.

Функции, начинающиеся с...

- `"get..."` – возвращают значение,
- `"calc..."` – что-то вычисляют,
- `"create..."` – что-то создают,
- `"check..."` – что-то проверяют и возвращают логическое значение, и т.д.

Примеры таких имён:

```
1 showMessage(..) // показывает сообщение
2 getAge(..)      // возвращает возраст (получая его каким-то образом)
```



```
3 calcSum(..)          // вычисляет сумму и возвращает результат
4 createForm(..)       // создаёт форму (и обычно возвращает её)
5 checkPermission(..)  // проверяет доступ, возвращая true/false
```

Благодаря префиксам, при первом взгляде на имя функции становится понятным, что делает её код, и какое значение она может возвращать.

### **i** Одна функция – одно действие

Функция должна делать только то, что явно подразумевается её названием. И это должно быть одним действием.

Два независимых действия обычно подразумевают две функции, даже если предполагается, что они будут вызываться вместе (в этом случае мы можем создать третью функцию, которая будет их вызывать).

Несколько примеров, которые нарушают это правило:

- `getAge` – будет плохим выбором, если функция будет выводить `alert` с возрастом (должна только возвращать его).
- `createForm` – будет плохим выбором, если функция будет изменять документ, добавляя форму в него (должна только создавать форму и возвращать её).
- `checkPermission` – будет плохим выбором, если функция будет отображать сообщение с текстом `доступ разрешён/запрещён` (должна только выполнять проверку и возвращать её результат).

В этих примерах использовались общепринятые смыслы префиксов. Конечно, вы в команде можете договориться о других значениях, но обычно они мало отличаются от общепринятых. В любом случае вы и ваша команда должны чётко понимать, что значит префикс, что функция с ним может делать, а чего не может.

### **i** Сверхкороткие имена функций

Имена функций, которые используются *очень часто*, иногда делают сверхкороткими.

Например, фреймворк `jQuery` определяет функцию с помощью `$`. В библиотеке `Lodash` основная функция представлена именем `_`.

Это исключения. В основном имена функций должны быть в меру краткими и описательными.

## Функции == Комментарии

Функции должны быть короткими и делать только что-то одно. Если это что-то большое, имеет смысл разбить функцию на несколько меньших. Иногда следовать этому правилу непросто, но это определённо хорошее правило.

Небольшие функции не только облегчают тестирование и отладку – само существование таких функций выполняет роль хороших комментариев!

Например, сравним ниже две функции `showPrimes(n)`. Каждая из них выводит **простое число** до `n`.

Первый вариант использует метку `nextPrime`:

```
1 function showPrimes(n) {
2   nextPrime: for (let i = 2; i < n; i++) {
3
4     for (let j = 2; j < i; j++) {
```

```

5         if (i % j == 0) continue nextPrime;
6     }
7
8     alert( i ); // простое
9 }
10 }

```

Второй вариант использует дополнительную функцию `isPrime(n)` для проверки на простое:

```

1 function showPrimes(n) {
2
3     for (let i = 2; i < n; i++) {
4         if (!isPrime(i)) continue;
5
6         alert(i); // простое
7     }
8 }
9
10 function isPrime(n) {
11     for (let i = 2; i < n; i++) {
12         if ( n % i == 0) return false;
13     }
14     return true;
15 }

```

Второй вариант легче для понимания, не правда ли? Вместо куска кода мы видим название действия (`isPrime`). Иногда разработчики называют такой код *самодокументируемым*.

Таким образом, допустимо создавать функции, даже если мы не планируем повторно использовать их. Такие функции структурируют код и делают его более понятным.

## Итого

Объявление функции имеет вид:

```

1 function имя(параметры, через, запятую) {
2     /* тело, код функции */
3 }

```

- Передаваемые значения копируются в параметры функции и становятся локальными переменными.
- Функции имеют доступ к внешним переменным. Но это работает только изнутри наружу. Код вне функции не имеет доступа к её локальным переменным.
- Функция может возвращать значение. Если этого не происходит, тогда результат равен `undefined`.

Для того, чтобы сделать код более чистым и понятным, рекомендуется использовать локальные переменные и параметры функций, не пользоваться внешними переменными.

Функция, которая получает параметры, работает с ними и затем возвращает результат, гораздо понятнее функции, вызываемой без параметров, но изменяющей внешние переменные, что чревато побочными эффектами.

Именование функций:

- Имя функции должно понятно и чётко отражать, что она делает. Увидев её вызов в коде, вы должны тут же понимать, что она делает, и что возвращает.
- Функция – это действие, поэтому её имя обычно является глаголом.
- Есть много общепринятых префиксов, таких как: `create...`, `show...`, `get...`, `check...` и т.д. Пользуйтесь ими как подсказками, поясняющими, что делает функция.

Функции являются основными строительными блоками скриптов. Мы рассмотрели лишь основы функций в JavaScript, но уже сейчас можем создавать и использовать их. Это только начало пути. Мы будем неоднократно возвращаться к функциям и изучать их всё более и более глубоко.

## ✓ Задачи

### Обязателен ли "else"?

важность: 4

Следующая функция возвращает `true`, если параметр `age` больше 18.

В ином случае она запрашивает подтверждение через `confirm` и возвращает его результат:

```
1 function checkAge(age) {
2   if (age > 18) {
3     return true;
4   } else {
5     // ...
6     return confirm('Родители разрешили?');
7   }
8 }
```

Будет ли эта функция работать как-то иначе, если убрать `else`?

```
1 function checkAge(age) {
2   if (age > 18) {
3     return true;
4   }
5   // ...
6   return confirm('Родители разрешили?');
7 }
```

Есть ли хоть одно отличие в поведении этого варианта?

решение



Оба варианта функций работают одинаково, отличий нет.

### Перепишите функцию, используя оператор '?' или '||'

важность: 4

Следующая функция возвращает `true` , если параметр `age` больше 18 .

В ином случае она задаёт вопрос `confirm` и возвращает его результат.

```
1 function checkAge(age) {
2   if (age > 18) {
3     return true;
4   } else {
5     return confirm('Родители разрешили?');
6   }
7 }
```

Перепишите функцию, чтобы она делала то же самое, но без `if` , в одну строку.

Сделайте два варианта функции `checkAge` :

1. Используя оператор `?`
2. Используя оператор `||`

решение

Используя оператор `?` :

```
1 function checkAge(age) {
2   return (age > 18) ? true : confirm('Родители разрешили?');
3 }
```

Используя оператор `||` (самый короткий вариант):

```
1 function checkAge(age) {
2   return (age > 18) || confirm('Родители разрешили?');
3 }
```

Обратите внимание, что круглые скобки вокруг `age > 18` не обязательны. Они здесь для лучшей читаемости кода.

## Функция `min(a, b)`

важность: 1

Напишите функцию `min(a, b)` , которая возвращает меньшее из чисел `a` и `b` .

Пример вызовов:

```
1 min(2, 5) == 2
2 min(3, -1) == -1
3 min(1, 1) == 1
```

решение



Вариант решения с использованием `if`:

```
1 function min(a, b) {  
2   if (a < b) {  
3     return a;  
4   } else {  
5     return b;  
6   }  
7 }
```

Вариант решения с оператором `?`:

```
1 function min(a, b) {  
2   return a < b ? a : b;  
3 }
```

P.S. В случае равенства `a == b` не имеет значения, что возвращать.

## Функция `pow(x,n)`

важность: 4

Напишите функцию `pow(x,n)`, которая возводит `x` в степень `n` и возвращает результат.

```
1 pow(3, 2) = 3 * 3 = 9  
2 pow(3, 3) = 3 * 3 * 3 = 27  
3 pow(1, 100) = 1 * 1 * ... * 1 = 1
```

Создайте страницу, которая запрашивает `x` и `n`, а затем выводит результат `pow(x,n)`.

[Запустить демо](#)

P.S. В этой задаче функция обязана поддерживать только натуральные значения `n`, т.е. целые от 1 и выше.

решение



```
1 function pow(x, n) {  
2   let result = x;  
3  
4   for (let i = 1; 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 < 1) {
15      alert(`Степень ${n} не поддерживается, используйте натуральное число`);
16  } else {
17      alert( pow(x, n) );
18  }
```

[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

125

Share

Лучшие

Новые

Старые

**BB** **bee bee**  
🕒 9 дней назад

— 🗑

Мое решение последней задачи.

```
function pow(x = prompt('vedite chislo', ''), n = prompt('vedite stepen', '')) {  
  if (n > 0) {  
    let result;  
    result = (x ** n);  
    return result;  
  }  
  else {  
    result = ('vedite n > 0')  
    return result;  
  }  
};  
console.log(pow())
```

1 2 Ответить

**S** **Stanislau**  
🕒 9 дней назад

— 🗑

```
function po (a,b) {  
  a = prompt ('введите a')  
  b = prompt('введите b')  
  return ((Math.pow(parseInt(a),parseInt(b))))  
}  
последняя задача
```

0 0 Ответить

**MB** **Mikhail Bagasaryan**  
🕒 10 дней назад

— 🗑

```
Объясните, пожалуйста, для чего в последней задаче использовать for (let i = 1; i < n; i++) {  
  result *= x;  
}
```

Я решал так:

```
function pow(a,b) {  
  a && b > 0 ? alert(a**b) : alert("Число не поддерживается")  
}
```

U U Ответить



Саня

→ Mikhail Bagasaryan

🕒 8 дней назад



Автор сделал решение без использования оператора возведения в степень, поэтому там используется цикл и так далее. То есть, это сугубо спортивный интерес, реализовать самому, потому как в условиях задачи не сказано, что надо обязательно самостоятельно это сделать, а не использовать оператор.

1 0 Ответить



Vova Buryak

🕒 13 дней назад



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

```
function checkNumber() {
  if (n % 1 == 0 && x % 1 == 0 && x > 0 && n > 0) {
    calc(x,n);
  } else {
    alert('Степень не поддерживается, используйте натуральное число');
  }
}
checkNumber()
```

```
function calc(x, n) {
  alert(Math.pow(x, n));
}
```

0 0 Ответить



Denis Butyrskiy

🕒 13 дней назад



Ребят, всем привет! Изучаю JavaScript.

**Главная цель ⚡ - «С нуля до Junior Frontend Developer за 7 месяцев».** На протяжении этих 7 месяцев буду изучать JS и записывать еженедельные **видеоотчёты** об этом.

Всё здесь:

**YouTube** по названию канала **Denis Butyrskiy**

**ТГ @itway\_chat** - обсуждаем всё, что касается обучения

Давайте объединяться, общаться, делиться друг с другом успехами и неудачами. Это сильно помогает пройти сложные моменты, уже проверено. Под каждым видео есть вся нужная информация 🍷

0 0 Ответить



Артём Вишняков

🕒 20 дней назад edited





Помогите гуманитарию (часть 2)

И почему здесь выводится "2" тоже не понимаю, ведь j

0 0 Ответить

**AB** **Артём Вишняков**  
🕒 20 дней назад edited

— 📄

Помогите гуманитарию (часть 1)

Не понимаю, как это работает!

```
function showPrimes(n) {  
  
  for (let i = 2; i < n; i++) {  
    if (!isPrime(i)) continue;  
  
    alert(i); // простое  
  }  
}  
  
function isPrime(n) {  
  for (let i = 2; i < n; i++) {  
    if (n % i == 0) return false;  
  }  
  return true;  
}
```

0 0 Ответить

**BC** **Вадим Сергеев**  
🕒 20 дней назад

— 📄

```
let x = +prompt("Введите число x");  
let n = +prompt("Введите число n");  
alert(pow(x, n));  
  
function pow(x, n){  
  if (n < 1 || n % 1 !== 0){  
    return "Степень n не поддерживается введите число больше единицы и натуральное";  
  }else{  
    return x ** n;  
  }  
}
```

P.S. Вот моё решение со всеми проверками)

0 0 Ответить

**CS** **Curtz STJ**  
🕒 21 день назад

— 📄

Ребят, можно попросить знающих интерпретировать код в последней задаче. Конкретно само решение. Пожалуйста.

Уже час потратила на бесцельные попытки понять. И тему циклов перечитала...

0 0 Ответить

Я тоже вхожу в то число людей, которые не поняли, зачем изворачиваться циклом)

```
let x = +prompt();
let n = +prompt();
let result = pow(x,n);

function pow(x,n) {
  if (n < 1) {
    console.log('Степень меньше единицы');
  }
  return result = x**n;
}
console.log(`Результат возведения: ${result}`);
```

2 0 Ответить

**АД** **Алексей Данилюк**  
🕒 21 день назад

— 🚩

```
function pow(x, n) {
  return x**n;
}
let x = prompt('x:');
let n = prompt('n:');
if (n < 1) {
  alert(`Степень ${n} не поддерживается, используйте натуральное число`);
} else {
  alert( pow(x, n) );
}
```

0 0 Ответить

**АД** **Алексей Данилюк**  
🕒 22 дня назад

— 🚩

```
function min(a, b) {
  if (a < b){
    return a;
  }
  else {
    return b;
  }
}
```

```
alert(min(6,2));
```

0 0 Ответить



**Road to WEB 2023**  
🕒 22 дня назад

— 🚩

✅ функции

давайте вместе ботать, пытаюсь вкатиться в веб-разработку за январь, roadtowed2023

0 0 Ответить

ВК **Влад Колесников**  
🕒 24 дня назад edited

— 📌

Не совсем понимаю зачем все усложнять циклами если и так работает?:

```
x = prompt("x");
n = prompt("n");
alert(pow(x,n));
function pow(x,n){
  if (n < 1){
    result = "Не натуральное число!";
  } else {
    result = (x ** n);
  };
  return result
}
```

Хотя как понял в моем варианте n может иметь и нецелые числа, например 3.5 и т.д

А так работает как надо:

```
x = prompt("x");
n = prompt("n");
alert(pow(x,n));
function pow(x,n){
```

показать больше

4 0 Ответить

НН **Hick Hickert** → Влад Колесников  
🕒 21 день назад

— 📌

в решении из учебника вообще нет никаких проверок за исключением 0 в степени. чтобы оставить только натуральные числа можно провести проверку с методом Number.isInteger (будет дальше) и исключить falsy значения которые к 0 приравниваются. Например так:

```
let x = +prompt('Enter a number!'),
    n = +prompt('Enter degree of (natural number)!');

function pow(x, n) {
  (Number.isInteger(n) && n !== 0) ? alert(x ** n) : alert('wrong value! Enter the natural
}

pow(x, n); // вызвать функцию
```

2 0 Ответить

КК **Katya Klep** → Hick Hickert  
🕒 20 дней назад

— 📌

Разве объявлять переменные вне функции не плохая практика?. Можно внутри объявить?

1 0 Ответить

НН **Hick Hickert** → Katya Klep  
🕒 19 дней назад edited

— 📌

а какие переменные внутри функции нужны? здесь в функции просто проверка и по ее результатам два различных возврата.  
Объявление переменных нужно для "создания" страницы в соответствии с условием.  
Вы можете вообще не объявлять переменные, а просто ввести значения в саму функцию:  
например `row(4,4)`; получите 256 на выходе.

0 0 Ответить

K

kerto

→ Влад Колесников

🕒 22 дня назад

— 🗨

Решения могут быть разными главное что работает. У автора просто другой подход.

1 0 Ответить

HH

Hick Hickert

→ kerto

🕒 21 день назад

— 🗨

к сожалению решение в учебнике не работает как прописано)  
если ввести не натуральное число, а с дробью - будет считать  
Даже текст будет считать)

1 0 Ответить

**Загрузить ещё комментарии**

Подписаться

Privacy

Не продавайте мои данные

