1. [Алгоритмы JavaScript и структуры данных](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures)
2. [Базовый JavaScript](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/#basic-javascript)

**Объявить переменные JavaScript**

В информатике *данные* — это все, что имеет смысл для компьютера. JavaScript предоставляет восемь различных *типов данных* : undefined, null, boolean, string, symbol, bigint, numberи object.

Например, компьютеры различают числа, такие как число 12, и strings, такие как "12", "dog", или "123 cats", которые представляют собой наборы символов. Компьютеры могут выполнять математические операции с числом, но не со строкой.

*Переменные* позволяют компьютерам динамически хранить данные и управлять ими. Они делают это, используя «метку» для указания на данные, а не используя сами данные. Любой из восьми типов данных может храниться в переменной.

Переменные похожи на переменные x и y, которые вы используете в математике, что означает, что они представляют собой простое имя для представления данных, на которые мы хотим ссылаться. Компьютерные переменные отличаются от математических переменных тем, что они могут хранить разные значения в разное время.

Мы говорим JavaScript создать или *объявить* переменную, поставив перед varней ключевое слово, например:

var ourName;

создает переменную с именем ourName. В JavaScript операторы заканчиваются точкой с запятой. Имена переменных могут состоять из цифр, букв и $или \_, но не должны содержать пробелов или начинаться с цифры.

**Сохранение значений с помощью оператора присваивания**

В JavaScript вы можете сохранить значение в переменной с помощью оператора *присваивания*= ( ).

myVariable = 5;

Это присваивает Number значение 5.myVariable

Если справа от  = оператора есть какие-либо вычисления, они выполняются до присвоения значения переменной слева от оператора.

var myVar;

myVar = 5;

Во-первых, этот код создает переменную с именем myVar. Затем код 5 присваивается myVar. Теперь, если myVar он снова появится в коде, программа обработает его так, как будто это 5.

**Присвоение значения одной переменной другой**

После присвоения значения переменной с помощью оператора *присваивания* вы можете присвоить значение этой переменной другой переменной с помощью оператора *присваивания .*

var myVar;

myVar = 5;

var myNum;

myNum = myVar;

Вышеприведенное объявляет myVar переменную без значения, а затем присваивает ей значение 5. Затем переменная с именем myNum объявляется без значения. Затем содержимое myVar (то есть 5) присваивается переменной myNum. Теперь myNum также имеет значение 5.

**Инициализация переменных с помощью оператора присваивания**

Обычно переменная *инициализируется начальным значением в той же строке, в которой она объявлена.*

var myVar = 0;

**Объявить строковые переменные**

Ранее вы использовали следующий код для объявления переменной:

var myName;

Но вы также можете объявить строковую переменную следующим образом:

var myName = "your name";

"your name"называется *строковым литералом* . Строковый литерал или строка — это последовательность из нуля или более символов, заключенная в одинарные или двойные кавычки.

# Понимание неинициализированных переменных

При объявлении переменных JavaScript они имеют начальное значение undefined. Если вы выполните математическую операцию над undefinedпеременной, ваш результат будет NaN означать «Не число» . Если вы соедините строку с переменной , undefined вы получите строку .undefined

# Понимание чувствительности к регистру в переменных

В JavaScript все имена переменных и функций чувствительны к регистру. Это означает, что капитализация имеет значение.

MYVAR не то же самое, что MyVar ни myvar. Можно иметь несколько разных переменных с одинаковым именем, но с разным регистром. Настоятельно рекомендуется для ясности не использовать эту языковую функцию.

**Лучшая практика**

Пишите имена переменных в JavaScript в camelCase . В camelCase имена переменных, состоящие из нескольких слов, имеют первое слово в нижнем регистре, а первая буква каждого последующего слова — заглавная.

**Примеры:**

var someVariable;

var anotherVariableName;

var thisVariableNameIsSoLong;

# Исследуйте различия между ключевыми словами var и let

Одна из самых больших проблем с объявлением переменных с помощью varключевого слова заключается в том, что вы можете легко перезаписать объявления переменных:

var camper = "James";

var camper = "David";

console.log(camper);

В приведенном выше коде camperпеременная изначально объявлена ​​как James, а затем переопределена на David. Затем консоль отображает строку David.

В небольшом приложении вы можете не столкнуться с проблемой такого типа. Но по мере того, как ваша кодовая база становится больше, вы можете случайно перезаписать переменную, которую вы не собирались делать. Поскольку такое поведение не приводит к ошибке, поиск и исправление ошибок усложняется.

Ключевое слово letnamed было введено в ES6, крупном обновлении JavaScript, чтобы решить эту потенциальную проблему с varключевым словом. Вы узнаете о других функциях ES6 в следующих задачах.

Если заменить varна letв приведенном выше коде, это приведет к ошибке:

let camper = "James";

let camper = "David";

Ошибку можно увидеть в консоли браузера.

Таким образом, в отличие от var, когда вы используете let, переменная с тем же именем может быть объявлена ​​только один раз.

# Объявите переменную только для чтения с ключевым словом const

Ключевое слово let— не единственный новый способ объявления переменных. В ES6 вы также можете объявлять переменные с помощью constключевого слова.

constимеет все замечательные функции, которые letимеет, с дополнительным бонусом, что переменные, объявленные с использованием const, доступны только для чтения. Они представляют собой постоянное значение, что означает, что после того, как переменной присвоено значение const, ее нельзя переназначить:

const FAV\_PET = "Cats";

FAV\_PET = "Dogs";

Консоль выдаст ошибку из-за переназначения значения FAV\_PET.

Вы всегда должны называть переменные, которые вы не хотите переназначать, используя constключевое слово. Это помогает, когда вы случайно пытаетесь переназначить переменную, которая должна оставаться неизменной.

**Примечание.** Разработчики обычно используют идентификаторы переменных в верхнем регистре для неизменяемых значений и нижний регистр или camelCase для изменяемых значений (объектов и массивов). Вы узнаете больше об объектах, массивах и неизменяемых и изменяемых значениях в следующих задачах. Также в более поздних задачах вы увидите примеры идентификаторов переменных в верхнем, нижнем или верблюжьем регистре.

# Добавить два числа с помощью JavaScript

Numberэто тип данных в JavaScript, который представляет числовые данные.

Теперь попробуем сложить два числа с помощью JavaScript.

JavaScript использует этот +символ как оператор сложения, когда он помещается между двумя числами.

**Пример:**

const myVar = 5 + 10;

myVarтеперь имеет значение 15.

# Вычесть одно число из другого с помощью JavaScript

Мы также можем вычесть одно число из другого.

JavaScript использует -символ для вычитания.

**Пример**

const myVar = 12 - 6;

myVarбудет иметь значение 6.

# Умножение двух чисел с помощью JavaScript

Мы также можем умножать одно число на другое.

JavaScript использует \*символ для умножения двух чисел.

**Пример**

const myVar = 13 \* 13;

myVarбудет иметь значение 169.

# Разделите одно число на другое с помощью JavaScript

Мы также можем разделить одно число на другое.

JavaScript использует /символ для деления.

**Пример**

const myVar = 16 / 2;

myVarтеперь имеет значение 8.

# Увеличение числа с помощью JavaScript

Вы можете легко увеличить или добавить единицу к переменной с помощью ++оператора.

i++;

является эквивалентом

i = i + 1;

**Примечание** . Вся строка становится i++;, что устраняет необходимость в знаке равенства.

# Уменьшить число с помощью JavaScript

Вы можете легко уменьшить или уменьшить переменную на единицу с помощью --оператора.

i--;

является эквивалентом

i = i - 1;

**Примечание** . Вся строка становится i--;, что устраняет необходимость в знаке равенства.

# Создание десятичных чисел с помощью JavaScript

Мы также можем хранить десятичные числа в переменных. Десятичные числа иногда называют числами с плавающей запятой или числами с плавающей запятой .

**Примечание:** когда вы вычисляете числа, они вычисляются с конечной точностью. Операции с плавающей запятой могут привести к результатам, отличным от желаемого. Если вы получаете один из этих результатов, откройте тему на [форуме freeCodeCamp](https://forum.freecodecamp.org/) .

# Умножение двух десятичных дробей с помощью JavaScript

В JavaScript вы также можете выполнять вычисления с десятичными числами, как и с целыми числами.

Давайте умножим два десятичных знака вместе, чтобы получить их произведение.

# Разделите одно десятичное число на другое с помощью JavaScript

Теперь разделим одну десятичную дробь на другую.

**Поиск остатка в JavaScript**

Оператор *остатка* дает %остаток от деления двух чисел.

**Пример**

5 % 2 = 1, потому что  
Math.floor(5 / 2) = 2 (частное)  
2 \* 2 = 4  
5 - 4 = 1 (остаток)

**Использование**  
В математике число можно проверить на четность или нечетность, проверив остаток от деления числа на 2.

17 % 2 = 1 (17 — нечетное)  
48 % 2 = 0 (48 — четное)

**Примечание.** Оператор *остатка* иногда неправильно называют оператором модуля. Он очень похож на модуль, но не работает должным образом с отрицательными числами.

# Составное задание с дополненным дополнением

В программировании обычно используются присваивания для изменения содержимого переменной. Помните, что сначала оценивается все, что находится справа от знака равенства, поэтому мы можем сказать:

myVar = myVar + 5;

добавить 5к myVar. Поскольку это такой распространенный шаблон, существуют операторы, которые выполняют и математическую операцию, и присваивание за один шаг.

Одним из таких операторов является +=оператор.

let myVar = 1;

myVar += 5;

console.log(myVar);

6будет отображаться в консоли.

**Составное задание с расширенным вычитанием**

Как и +=оператор, -=вычитает число из переменной.

myVar = myVar - 5;

будет вычитать 5из myVar. Это можно переписать как:

myVar -= 5;

# Составное задание с расширенным умножением

Оператор \*=умножает переменную на число.

myVar = myVar \* 5;

будет умножаться myVarна 5. Это можно переписать как:

myVar \*= 5;

# Составное задание с расширенным делением

Оператор /=делит переменную на другое число.

myVar = myVar / 5;

Будет делить myVarна 5. Это можно переписать как:

myVar /= 5;

**Экранирование буквенных кавычек в строках**

Когда вы определяете строку, вы должны начинаться и заканчиваться одинарной или двойной кавычкой. Что происходит, когда вам нужна буквальная цитата: "или 'внутри вашей строки?

В JavaScript вы можете *избежать* того , чтобы кавычка рассматривалась как кавычка в конце строки, поместив *обратную косую черту* ( \) перед кавычкой.

const sampleStr = "Alan said, \"Peter is learning JavaScript\".";

Это сигнализирует JavaScript, что следующая кавычка не является концом строки, а вместо этого должна находиться внутри строки. Итак, если вы напечатаете это на консоли, вы получите:

Alan said, "Peter is learning JavaScript".

# Заключение строк в одинарные кавычки

Строковые значения в JavaScript могут быть записаны в одинарных или двойных кавычках, если вы начинаете и заканчиваете кавычками одного типа. В отличие от некоторых других языков программирования, одинарные и двойные кавычки в JavaScript работают одинаково.

const doubleQuoteStr = "This is a string";

const singleQuoteStr = 'This is also a string';

Причина, по которой вы можете захотеть использовать один тип кавычек вместо другого, заключается в том, что вы хотите использовать оба типа в строке. Это может произойти, если вы хотите сохранить беседу в виде строки и заключить беседу в кавычки. Другим использованием для этого было бы сохранение <a>тега с различными атрибутами в кавычках, все в строке.

const conversation = 'Finn exclaims to Jake, "Algebraic!"';

Однако это становится проблемой, если вам нужно использовать самые внешние кавычки внутри него. Помните, что строка имеет одинаковые кавычки в начале и в конце. Но если у вас есть такая же цитата где-то посередине, строка остановится раньше и выдаст ошибку.

const goodStr = 'Jake asks Finn, "Hey, let\'s go on an adventure?"';

const badStr = 'Finn responds, "Let's go!"';

Вот badStrвыдаст ошибку.

В приведенном выше goodStr вы можете безопасно использовать обе кавычки, используя обратную косую черту \в качестве escape-символа.

**Примечание:** обратную косую черту \не следует путать с прямой косой чертой /. Они не делают то же самое.

# Escape-последовательности в строках

Кавычки — не единственные символы, которые можно экранировать внутри строки. Есть две причины использовать экранирующие символы:

1. Чтобы вы могли использовать символы, которые иначе не сможете напечатать, например, возврат каретки.
2. Чтобы позволить вам представлять несколько кавычек в строке без того, чтобы JavaScript неправильно интерпретировал то, что вы имеете в виду.

Мы узнали об этом в предыдущем задании.

| **Код** | **Выход** |
| --- | --- |
| \' | одинарная кавычка |
| \" | двойная кавычка |
| \\ | обратная косая черта |
| \n | новая линия |
| \r | возврат каретки |
| \t | вкладка |
| \b | граница слова |
| \f | подача формы |

Code Output

\' single quote

\" double quote

\\ backslash

\n newline

\r carriage return

\t tab

\b word boundary

\f form feed

Обратите внимание, что саму обратную косую черту необходимо экранировать, чтобы она отображалась как обратная косая черта.