**ИЗУЧИТЕ JAVASCRIPT В CONSTRUCT, ЧАСТЬ 1: НАЧАЛО РАБОТЫ**

Это руководство рекомендовано командой Construct! Это означает, что он содержит полезную и качественную информацию, которая поможет вам вырасти как разработчику игр.

[JavaScript](https://www.construct.net/out?u=https%3a%2f%2fen.wikipedia.org%2fwiki%2fJavaScript) — [один из самых популярных языков программирования в мире](https://www.construct.net/out?u=https%3a%2f%2finsights.stackoverflow.com%2fsurvey%2f2021%23technology-most-popular-technologies) , с широким спектром приложений, от браузеров до серверов, [множеством компаний, желающих нанять разработчиков JavaScript](https://www.construct.net/out?u=https%3a%2f%2fdevskiller.com%2fit-skills-report-2020%2f) , и многими другими преимуществами, такими как выдающаяся производительность. Все это делает его отличным выбором для начала обучения программированию. Эта серия руководств поможет вам изучить основы языка программирования JavaScript в [Construct](https://www.construct.net/" \o "Делайте свои собственные игры" \t "_blank) !

Этот урок предназначен для новичков. Если вы уже знаете JavaScript, ознакомьтесь с [кратким руководством для разработчиков Construct for JavaScript](https://www.construct.net/en/tutorials/construct-javascript-2866) .

## JAVASCRIPT — ЭТО НЕ JAVA

Одна из распространенных путаниц, которую следует сразу прояснить, заключается в том, что **Java** и **JavaScript** — это два совершенно разных языка программирования! Именование действительно несколько сбивает с толку, но постарайтесь не путать их. Например, если вы ищете в Интернете справку по JavaScript, обязательно ищите именно JavaScript , а не Java , иначе вы получите результаты для неправильного языка программирования!

## ОБ ЭТОМ РУКОВОДСТВЕ

Это руководство поможет вам **изучить язык программирования JavaScript,** используя функцию кодирования JavaScript Construct. Это хороший способ начать, поскольку вы можете сразу начать писать код без необходимости настраивать среду разработки (например, локальный HTTP-сервер). Вся серия руководств предназначена для работы в рамках бесплатной версии, поэтому вы можете изучать JavaScript с помощью этого руководства, не покупая ничего. Функция кодирования JavaScript в Construct также предоставляет множество способов сделать больше в ваших проектах, поэтому она будет полезна и для более продвинутых пользователей Construct.

Construct использует **стандартный язык программирования JavaScript** , поэтому вы изучаете то же самое, что используется повсюду в отрасли. Construct никоим образом не меняет язык — фактически браузер запускает ваш код напрямую.

Блочная система Construct в таблицах событий разработана таким образом, чтобы новичкам было проще начать работу. Если вы хотите попробовать начать работу с блоками или просто сначала немного познакомиться с Construct, рассмотрите возможность попробовать [руководство для начинающих по Construct](https://www.construct.net/en/tutorials/beginners-guide-construct-1) .

**Вам не обязательно использовать Construct раньше, чтобы следовать этому руководству.** Однако некоторое знакомство будет полезно. Если вы также знакомы с переменными, циклами и функциями в таблицах событий, это также будет полезно. Эти функции таблиц событий намеренно созданы так, чтобы быть похожими на работу эквивалентных функций языка программирования. Итак, если вы знаете, как они работают в таблицах событий, это поможет вам понять, как они работают в JavaScript. В этом руководстве будет указано, где таблицы событий имеют концепции, схожие с JavaScript. Однако эти ссылки не будут необходимы для следования руководству, это просто дополнительный контекст для пользователей Construct, которые уже знакомы с таблицами событий.

## ЧТО ВЫ УЗНАЕТЕ

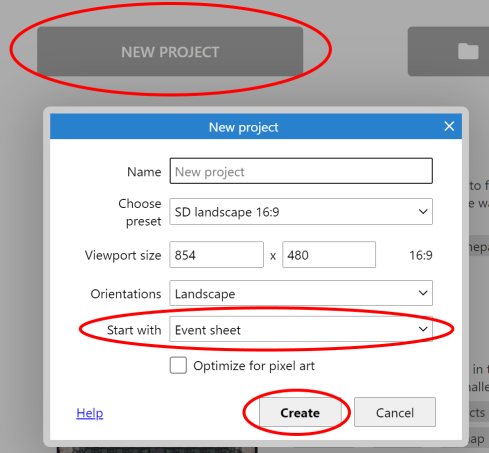
В этом руководстве будут рассмотрены основные возможности языков программирования JavaScript, включая такие темы, как переменные, операторы if, циклы и функции. JavaScript — это развитый язык с огромным набором функций, и данное руководство не пытается охватить его полностью. Цель состоит в том, чтобы научить вас достаточному количеству JavaScript, чтобы вы могли изучить его из других руководств в Интернете и научиться применять его в Construct. (Некоторые дополнительные руководства будут приведены в конце серии.)

JavaScript был изобретен в 1995 году, и, как и большинство старых языков программирования, JavaScript имеет немало особенностей и устаревших функций. Однако в этом руководстве основное внимание будет уделено наиболее полезным частям с использованием современного стиля и лучших практик, и мы не будем вдаваться в подробности об особенностях или старых функциях, которые больше не используются широко. Другими словами, это практическое руководство, ориентированное на написание реального кода.

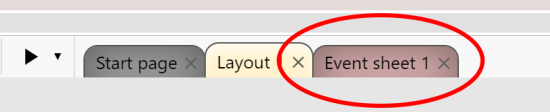
Готовы начать? Перейдите на следующую страницу, чтобы написать свою первую строку кода!

## ДОБАВЛЕНИЕ КОДА JAVASCRIPT В ЛИСТ СОБЫТИЙ

Для простого начала давайте напишем нашу первую строку кода в листе событий. Сначала [откройте редактор Construct](https://editor.construct.net/) и создайте новый пустой проект, убедившись, что для параметра «Начать с» установлено значение «Лист событий» .

[](https://construct-static.com/images/v1202/uploads/articleuploadobject/0/images/66137/create-new-project.png)

Затем щелкните вкладку «Лист событий 1» , чтобы переключиться на просмотр листа событий.

[](https://construct-static.com/images/v1202/uploads/articleuploadobject/0/images/66138/event-sheet-tab.png)

Теперь добавьте событие «В начале макета» :

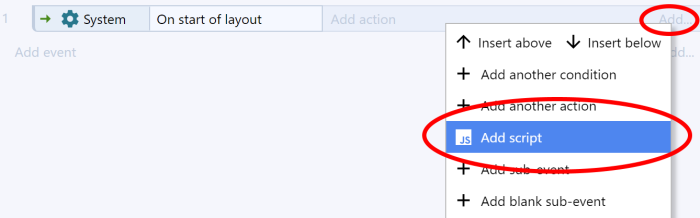
1. Нажмите **«Добавить событие»** на вкладке «Лист событий».
2. Дважды щелкните **Система**
3. Дважды щелкните **«В начале макета»** (в группе «Начало и конец »).

В итоге у вас должно получиться следующее:

[В начале макета](https://construct-static.com/images/v1202/uploads/articleuploadobject/0/images/65533/on-start-of-layout.png)

Это блок событий, который будет выполнять свои действия при запуске.

Мы можем добавить некоторый код JavaScript вместо действия в этом блоке событий, чтобы наш код запускался при запуске. Нажмите «Добавить...» справа от блока и выберите « Добавить скрипт» .

[](https://construct-static.com/images/v1202/uploads/articleuploadobject/0/images/65562/add-script-action.png)

Текстовое поле появляется как действие. Это небольшой редактор сценариев, в который вы можете ввести код JavaScript для запуска.

[Редактирование действия сценария](https://construct-static.com/images/v1202/uploads/articleuploadobject/0/images/65536/edit-script-action.png)

А пока давайте просто скопируем и вставим следующую строку кода в это текстовое поле:

**console.log("Hello world!");**

Обратите внимание, что если вы перейдете в другое окно, редактор сценариев может завершить работу и превратиться в пустой блок кода. В этом случае **дважды щелкните** пустой блок кода, чтобы отредактировать его еще раз, а затем вставьте или скопируйте приведенный выше пример кода. Затем щелкните за пределами редактора сценариев или нажмите Ctrl+ Enter, чтобы закрыть редактор сценариев.

Нажатие Enterдобавляет еще одну строку в редактор сценариев на случай, если вы захотите написать несколько строк кода, поэтому вместо этого для завершения редактирования используется Ctrl+ .Enter

Теперь вы должны увидеть свою первую строку кода JavaScript в таблице событий!

[Ваша первая строка кода JavaScript!](https://construct-static.com/images/v1202/uploads/articleuploadobject/0/images/65537/console-log-code.png)

Эта строка кода зарегистрирует сообщение Hello world! в консоль браузера при запуске. Вскоре мы найдем, как просмотреть это сообщение. Чтобы помочь вам быстро начать работу, мы пока не будем полностью объяснять каждую часть этого кода — это руководство будет подробно рассмотрено позже. А пока вот краткое изложение того, что означает эта строка кода:

* **console.log**означает «получить доступ к **log**функции объекта **console**»
* Круглые скобки **(**и **)**означают «вызов **log**функции»
* **"Hello world!"**это некоторый текст, заключенный в двойные кавычки и передаваемый функции **log**.
* Точка с запятой **;**в конце обозначает конец оператора

Конечный результат означает «добавление сообщения с текстом Hello world! на консоль». Давайте узнаем, как увидеть наше сообщение!

## ИСПОЛЬЗОВАНИЕ ИНСТРУМЕНТОВ РАЗРАБОТЧИКА

Консоль браузера является частью инструментов разработчика браузера , или для краткости «инструментов разработки». Когда вы только начинаете, знакомство с инструментами разработчика может показаться утомительным занятием, но оно того стоит по двум причинам:

1. Инструменты разработчика и такие вещи, как запись сообщений в консоль, являются важной частью повседневной работы по программированию. Определенно стоит узнать о них и ознакомиться с тем, как они работают.
2. Консоль браузера также является очень полезным инструментом, помогающим вам изучить JavaScript. Вы можете ввести фрагменты JavaScript, запустить их и сразу же увидеть результат, что является отличным способом поэкспериментировать и узнать больше. Мы попробуем это позже.

### ОТКРЫТИЕ ИНСТРУМЕНТОВ РАЗРАБОТЧИКА

Вы можете открыть инструменты разработки для любого окна или вкладки браузера. Однако нам нужны инструменты разработки только для окна предварительного просмотра, в котором запускается ваш проект, поскольку именно там запускается и ваш код JavaScript. Хотя вы можете открыть инструменты разработки для редактора Construct, здесь это бесполезно.

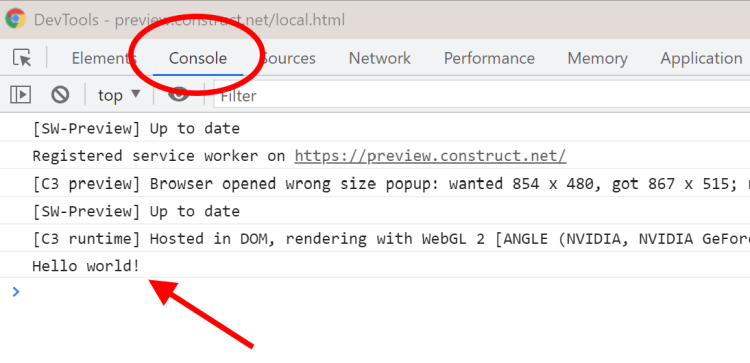
Поэтому прежде всего **просмотрите свой проект,** щелкнув значок «Воспроизвести» на панели инструментов рядом с кнопкой «Меню». (Вы также можете просмотреть, нажав F5.)

[Кнопка предварительного просмотра в Construct 3](https://construct-static.com/images/v1202/uploads/articleuploadobject/0/images/1096/preview-button.png)

Construct откроет всплывающее окно с пустым белым экраном, так как в макете ничего нет. Убедитесь, что окно предварительного просмотра является активным. Вы можете щелкнуть внутри окна предварительного просмотра, чтобы убедиться, что оно активно, а затем:

* В Safari нажмите Command+ Option+ I.
* В любом другом браузере нажмите F12.

Должно открыться новое окно для инструментов разработчика браузера. Перейдите на вкладку **«Консоль»** вверху, чтобы убедиться, что вы просматриваете список сообщений консоли. Вы должны увидеть Привет, мир! в списке сообщений! Могут быть и другие сообщения, которые Construct записал сам, но вы можете их игнорировать.

[](https://construct-static.com/images/v1202/uploads/articleuploadobject/0/images/65547/dev-tools-console.png)

Поздравляем! Вы запустили свою первую строку кода JavaScript и увидели записанное сообщение.

В этом руководстве мы будем использовать снимки экрана из инструментов разработчика Chrome. Инструменты разработчика других браузеров могут выглядеть немного иначе, но их все равно можно использовать таким же образом.

Если вы не можете открыть инструменты разработчика с помощью F12 в браузере Edge, перейдите в настройки и установите флажок « Использовать клавишу F12 для открытия инструментов разработчика» .

Если вы не видите свое сообщение в консоли или видите красную надпись «Стоп!» сообщение, возможно, вы открыли инструменты разработчика не в том окне. Закройте окно и повторите попытку, прежде чем нажимать сочетание клавиш, убедившись, что окно предварительного просмотра является активным.

## ИЗМЕНЕНИЕ СООБЩЕНИЯ

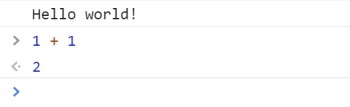
В Construct вы можете попробовать изменить зарегистрированное сообщение. (Не забудьте дважды щелкнуть блок сценария, чтобы отредактировать его, и щелкнуть снаружи, чтобы завершить.) Любой текст в двойных кавычках будет показан в консоли. Например, вы можете попробовать:

**console.log("My super cool message!");**

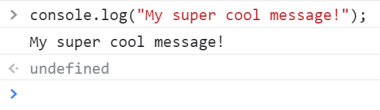
Просмотрите проект, откройте инструменты разработки, и вы увидите обновленное сообщение.

## ТЕСТИРОВАНИЕ КОДА В КОНСОЛИ

Как упоминалось ранее, замечательной особенностью консоли является то, что вы также можете ввести код и сразу же увидеть его запуск. Это очень удобный способ попробовать что-то. Внизу списка сообщений в консоли должно быть поле ввода, куда можно ввести какой-нибудь код. Попробуйте ввести **1 + 1**и нажать Enter. Он мгновенно запустит это как код JavaScript и зарегистрирует результат.

[](https://construct-static.com/images/v1202/uploads/articleuploadobject/0/images/65550/console-evaluation.png)

Вы также можете попробовать ввести код журнала консоли внутри консоли и увидеть, как он печатает там сообщение.

[](https://construct-static.com/images/v1202/uploads/articleuploadobject/0/images/65551/console-log-evaluation.png)

Вы можете задаться вопросом, почему **undefined**также появился. Это потому, что он регистрирует результат написанного вами кода. Когда писал **1 + 1**, результат был **2**. Однако **console.log()**не возвращает результат, поэтому результат **undefined**.

Вы можете ввести код в консоль таким образом в любой момент и сразу увидеть результат — или ошибку, если вы что-то ввели неправильно. Это отличный способ быстро поэкспериментировать с множеством различных вариантов кода и узнать больше о том, как работает JavaScript и какие результаты вы получаете от различных фрагментов кода. Не стесняйтесь опробовать его в любой момент в этом руководстве и поэкспериментировать с различными примерами кода — это отличный способ узнать больше.

## ЗАКЛЮЧЕНИЕ

В этой части мы рассмотрели:

* Об этом руководстве и о том, что вы узнаете из этой серии
* Как добавить код JavaScript в лист событий в Construct
* Использование **console.log()**для добавления сообщения в консоль браузера
* Как открыть инструменты разработчика браузера и просмотреть консоль
* Как ввести код в консоль, чтобы быстро опробовать

В этом руководстве для регистрации результатов выполнения кода часто используется консоль браузера, и она предоставляет отличное место для экспериментов с кодом, поэтому стоит потратить время на ее ознакомление.

# ИЗУЧИТЕ JAVASCRIPT В CONSTRUCT, ЧАСТЬ 2: ОСНОВЫ ЯЗЫКА

Это вторая часть серии руководств [«Изучите JavaScript в Construct»](https://www.construct.net/en/courses/learn-javascript-construct-79) . Это начинается с того места, где остановилась часть 1. Так что, если вы это пропустили, см. [«Изучаем JavaScript в Construct, часть 1: начало работы»](https://www.construct.net/en/tutorials/learn-javascript-construct-2800" \o "Делайте свои собственные игры" \t "_blank) .

Мы продолжим использовать тот же проект из предыдущей части. Вы можете отредактировать блок кода в событии «В начале макета», дважды щелкнув его, изменив код, а затем щелкнув за пределами или нажав Ctrl+ Enter, чтобы завершить редактирование. Затем вы можете просмотреть проект и открыть консоль браузера в инструментах разработчика, чтобы увидеть результаты. Стоит ознакомиться с этим процессом, поскольку мы будем в значительной степени полагаться на него. Попробуйте таким образом следующие фрагменты кода, чтобы увидеть, как они работают.

Иногда фрагменты кода предлагают вам попробовать их в консоли браузера. В этом случае не вводите их в свой блок кода в Construct — просмотрите проект, откройте консоль браузера и вместо этого попробуйте ввести туда каждую строку фрагмента кода.

## ЗАЯВЛЕНИЯ

Код JavaScript состоит из ряда операторов , разделенных точкой с запятой **;**. Например, этот код регистрирует два консольных сообщения, и каждая строка представляет собой оператор:

**console.log("First message!");**

**console.log("Second message!");**

Вы можете поместить два оператора в одну строку, поскольку на самом деле операторы разделяет точка с запятой. Однако традиционно в каждой строке содержится один оператор.

Точка с запятой в конце строки технически необязательна. Если вы забудете его туда положить, скорее всего, все будет работать. Однако это зависит от того, как JavaScript автоматически определяет, где ставить точки с запятой (функция под названием « Автоматическая вставка точки с запятой »), и это иногда может привести к неожиданным результатам. Поэтому лучше всего всегда использовать точки с запятой.

## КОММЕНТАРИИ

В программировании комментарий — это просто что-то, что вы можете написать в своем коде и что компьютер будет игнорировать. Другими словами, это примечание для людей, читающих код, а не инструкция для выполнения компьютером. Это аналогичная идея с комментариями в таблицах событий, которые также представляют собой просто заметки. В JavaScript есть два типа комментариев.

### ОДНОСТРОЧНЫЕ КОММЕНТАРИИ

Везде, где появляются два символа косой черты, **//**это означает однострочный комментарий. Все, начиная с косой черты и заканчивая концом строки, является комментарием. Комментарий может занимать отдельную строку или идти после некоторого кода.

**// This is a single-line comment!**

**console.log("Hello world!"); // And this!**

В этом учебном руководстве такие комментарии часто используются для аннотирования кода, указывая, что он делает или какой результат даст тот или иной код.

Вы заметите, что комментарии и другие фрагменты кода окрашены по-разному в этом руководстве, в Construct и в инструментах разработчика браузера. Это сделано для улучшения читаемости. (Цвет никак не влияет на работу кода.) В разных инструментах используются разные цветовые темы, но цель состоит в том, чтобы облегчить обнаружение таких вещей, как комментарии, поскольку они выделяются разным цветом.

### МНОГОСТРОЧНЫЕ КОММЕНТАРИИ

Однострочные комментарии естественным образом заканчиваются в конце строки, поэтому, если вам нужен длинный комментарий, вам нужно комментировать каждую строку:

**// First line of comment**

**// Second line of comment**

**// Third line of comment**

Многострочные комментарии могут быть более простым способом написать это. Многострочные комментарии начинаются с косой черты и звездочки **/\***и заканчиваются звездочкой и косой чертой **\*/**.

**/\* First line of comment**

**Second line of comment**

**Third line of comment \*/**

Несмотря на название, вы также можете написать такой комментарий в одну строку. Это может быть полезно, поскольку вы все равно можете писать код после окончания комментария.

**/\* comment before code \*/ console.log("Hello world!");**

**console.log(/\* comment inside code \*/ "Hello world!");**

В общем, вы можете оставлять комментарии где угодно, и это не повлияет на код. Как вы, возможно, заметили и в таблицах событий, добавление комментариев часто является хорошей идеей, чтобы помочь себе или другим людям понять код. Профессионально написанный код часто подробно комментируется, чтобы помочь любому, кто его читает, понять, как он работает. Лучший код ясен и прост для понимания, а не непонятен и сложен.

## ПЕРЕМЕННЫЕ

Переменная — это имя значения , хранящегося в памяти компьютера. Если вы использовали переменные в таблице событий Construct, например, глобальные или локальные переменные, концепция аналогична. Например, у вас может быть переменная с именем Score, в которой будет храниться число очков игрока. Значение может быть изменено или варьироваться , отсюда и название переменной .

### ОБЪЯВЛЕНИЕ И ПРИСВОЕНИЕ ПЕРЕМЕННЫХ

В JavaScript переменные объявляются с помощью **let**ключевого слова, за которым следует имя переменной. Код ниже создает переменную с именем message .

**let message;**

Значение, в данном случае строка (т.е. некоторый текст), может быть присвоено этой переменной с помощью оператора присваивания **=**.

**message = "Hello world!";**

Обратите внимание, что в данном случае **=**это не означает равенство . Это означает « присвоить правую часть левой». Вы также можете читать **=**как «установлено», например, для «сообщения» установлено значение «Привет, мир!» . Использование **=**этого способа — очень распространенный подход к присваиванию, используемый во многих языках программирования.

Теперь мы можем записать эту переменную в консоль. Вот полный фрагмент кода.

**let message;**

**message = "Hello world!";**

**console.log(message);**

Очень часто объявляют переменную и сразу же присваивают ей значение, поэтому и то, и другое можно объединить в один оператор, например:

**let message = "Hello world!";**

Несколько переменных также можно объявить с помощью запятых, с присвоением или без него. Например:

**let firstName, lastName, food = "pizza", drink = "cola";**

Эквивалентно:

**let firstName;**

**let lastName;**

**let food = "pizza";**

**let drink = "cola";**

Обычно мы рекомендуем использовать последний стиль с **let**объявлением каждой переменной, поскольку он более понятен.

Вы можете присваивать переменной разные значения. Следующий код устанавливает сообщение как Первое сообщение! а затем Второе сообщение! . Второе присвоение заменяет содержимое переменной, поэтому будет записано только второе сообщение! .

**let message;**

**message = "First message!";**

**message = "Second message!";**

**console.log(message);**

Что произойдет, если вы никогда не присвоите значение? Значением переменной по умолчанию в JavaScript является **undefined**специальное значение, означающее «ничего».

**let message;**

**console.log(message);**

Избегайте этого: использование **undefined**переменной обычно означает, что вы допустили ошибку. Чтобы избежать этого, рекомендуется всегда инициализировать переменную каким-либо значением.

### ПРАВИЛА ИСПОЛЬЗОВАНИЯ ПЕРЕМЕННЫХ

Вы должны объявить переменную перед ее использованием. Следующий код приведет к ошибке, поскольку он попытается использовать сообщение перед его объявлением.

**// ReferenceError: Cannot access 'message' before initialization**

**console.log(message);**

**let message = "Hello world!";**

Вы не можете объявить одно и то же имя переменной дважды.

**let message = "First message!";**

**// SyntaxError: Identifier 'message' has already been declared**

**let message = "Second message!";**

Имена переменных могут содержать только буквы, цифры или символы **$**и **\_**. Однако первый символ не может быть цифрой. Также имя переменной не может быть [зарезервированным словом](https://www.construct.net/out?u=https%3a%2f%2fdeveloper.mozilla.org%2fen-US%2fdocs%2fWeb%2fJavaScript%2fReference%2fLexical_grammar%23keywords) , то есть ключевым словом, используемым в языке JavaScript, например **let**.

**// Valid variable names:**

**let userName;**

**let user\_name;**

**let testing123;**

**// Invalid variable names:**

**let user-name; // dash '-' not allowed**

**let 9slice; // starting with digit not allowed**

**let let; // keyword not allowed**

Также обратите внимание, что JavaScript **чувствителен к регистру** , т. е. **message**рассматривается как имя переменной, отличное от имени **Message**или **MESSAGE**. В этом отличие от листов событий Construct, которые обычно не чувствительны к регистру.

Рекомендуется всегда использовать описательные имена переменных, чтобы облегчить понимание кода. Существуют также различные соглашения об именах, например, используете ли вы подчеркивание ( **first\_name**) или «верблюжий регистр» ( **firstName**). Не имеет большого значения, что вы выберете — просто будьте последовательны, и если вы работаете в команде, убедитесь, что все согласны с одним и тем же стилем именования.

## КОНСТАНТЫ

Вы также можете объявить константу, которая похожа на переменную, которую нельзя изменить. Они объявлены **const**вместо **let**. Некоторые значения никогда не должны меняться во время работы вашего кода, например математическая константа «пи» или максимальное разрешенное количество попыток входа в систему. Попытка изменить эти значения должна быть ошибкой и **const**требует этого.

**// Declare a constant**

**const PI = 3.141592653589793;**

**// TypeError: Assignment to constant variable.**

**PI = 4;**

Правила именования констант идентичны переменным, но общепринятым соглашением является присвоение константам имен в верхнем регистре, чтобы отличать их от переменных.

## ВАР

Вы можете встретить старый код JavaScript, используемый **var**для объявления переменных:

**var message = "Hello world!";**

Вы можете предположить, что это работает так же, как **let**. Однако избегайте его использования, поскольку это старая функция с некоторыми странными особенностями, в которые мы не будем здесь вдаваться. Современный код должен использовать только **let**.

## ТИПЫ ДАННЫХ

До сих пор мы работали только со строками (т.е. с текстом), такими как string **"Hello world!"**. JavaScript поддерживает несколько других типов данных. Здесь мы рассмотрим только два других: числа и логические значения.

### ЧИСЛА

Вы также можете использовать числовые значения вместо строк для переменных и констант. (На самом деле мы видели это ранее, когда использовали **1 + 1**в консоли и для константы **PI**.)

**let score = 100;**

**console.log(score);**

В JavaScript все числа являются числами с плавающей запятой , то есть они могут хранить как целые числа, например, **10**так и дробные значения, например **1.5**.

Вы можете использовать математические операторы с числами: **+**для сложения, **-**вычитания, **\***умножения и **/**деления. Вы можете попробовать это в консоли браузера — попробуйте ввести каждую строку ниже. В комментарии показан результат. Вы также можете попробовать свои собственные комбинации.

**// Try entering in to the console:**

**10 + 5 // 15**

**7 - 4 // 3**

**6 \* 3 // 18**

**20 / 4 // 5**

Эти расчеты можно включать в задания.

**let value = 10 + 5;**

**value = 6 \* 3; // replaces value**

**console.log(value + 1); // 19**

Вы также можете комбинировать вычисления, например **6 \* 3 + 1**, который соответствует стандартному порядку операций, при этом сначала выполняется умножение. Круглые скобки **(**и **)**могут использоваться для изменения порядка операций, например **6 \* (3 + 1)**.

Вы также можете использовать имя переменной в расчете:

**let a = 2 + 1.5;**

**let b = a \* 2;**

**console.log(b); // 7**

#### ТОЧНОСТЬ

Вы можете заметить странный результат с некоторыми дробями:

**// Try entering in to the console:**

**0.1 + 0.2 // 0.30000000000000004**

Это не точно! Это очень близко, но не совсем к правильному результату 0,3.

Это не ошибка. На самом деле именно так и происходят все дробные вычисления на компьютерах. Причина относительно сложна, но вкратце: компьютеры не имеют неограниченной памяти и поэтому вычисляют дроби только с ограниченной степенью точности. Рассмотрите возможность расчета **(2 / 3) \* 3**только до 6 знаков после запятой. **2 / 3**вычислил бы **0.666667**. Умножение этого числа на 3 даст **2.000001**- близко, но не совсем к предполагаемому результату - 2. В компьютерах этот тип округления происходит в двоичном формате (по основанию 2 или все 0 и 1), что означает, что он происходит в разных ситуациях, например как и **0.1 + 0.2**.

Нам, программистам, просто приходится с этим жить. Часто дробные вычисления будут немного неправильными, но иногда это не имеет значения, или программист следит за тем, чтобы значения в небольшом диапазоне считались приемлемыми, поэтому ошибки округления не влияют на результат.

#### СПЕЦИАЛЬНЫЕ НОМЕРА

Числа в JavaScript могут содержать три возможных специальных значения: положительную бесконечность ( **Infinity**), отрицательную бесконечность ( **-Infinity**) и «Не число» ( **NaN**).

Положительная и отрицательная бесконечность работают так, как и следовало ожидать:

**// Try entering in to the console:**

**1 / 0 // Infinity**

**-1 / 0 // -Infinity**

**Infinity + 1 // Infinity**

«Не число» или **NaN**, менее очевидно. Это результат, используемый, когда результат не может быть представлен математически, например, квадратный корень из отрицательного числа или неверный расчет.

**// Try entering in to the console:**

**0 / 0 // NaN**

В общем, если вы **NaN**это видите, значит, вы ошиблись в расчетах. Возможно, вам придется проверить правильность математических вычислений или добавить дополнительные проверки, чтобы убедиться, что вы не пытаетесь вычислить что-то вроде **0 / 0**. Это не всегда так очевидно, как кажется: вы можете использовать такие переменные, как **numberOfSweets / numberOfPeople**, и если обе переменные равны 0, результатом будет **NaN**.

### СТРУНЫ

Мы уже использовали строки, но давайте рассмотрим еще несколько деталей.

Почему мы называем текст в коде строкой ? Имя является сокращением от «строка символов», что на самом деле и представляет собой этот тип данных — последовательность отдельных символов, следующих один за другим. Строка имени обычно используется для обозначения типа данных текста в компьютерном программировании.

Для обозначения строки в строке могут использоваться символы **'**, **"**или **`**, при условии, что начальный и конечный символы совпадают. Все три примера ниже эквивалентны.

**"Hello world!"**

**'Hello world!'**

**`Hello world!`**

Не имеет значения, какой из них вы используете, но желательно соблюдать последовательность. В этом руководстве обычно для строк используются двойные кавычки.

Если вы хотите включить двойную кавычку внутри строки в двойных кавычках, вы можете использовать символ обратной косой черты \, чтобы экранировать ее. Это означает, что двойная кавычка является частью текста, а не отмечает конец строки.

**// Try entering in to the console:**

**"Hello \"world\"!" // Hello "world"!**

Это также означает, что сами обратные косые черты должны быть экранированы как \\. Вы можете использовать различные другие escape-последовательности, например \t для табуляции и \n для новой строки.

Альтернативой является использование для строки другой кавычки, например одинарных кавычек, например **'Hello "world"!'**.

Мы расскажем больше о строках в следующей части этого руководства.

### ЛОГИЧЕСКИЕ ЗНАЧЕНИЯ

Логическое значение может быть только либо **true**или **false**. Это полезно для значений включения/выключения, таких как настройки, а также для сравнения.

**let b = true;**

**console.log(b);**

Мы вернемся к логическим значениям позже в руководстве, поскольку они имеют некоторые ключевые применения в программировании на JavaScript, особенно при сравнениях и **if**операторах. Однако сейчас полезно знать, что логические значения — это еще один тип данных в JavaScript.

## ДИНАМИЧЕСКАЯ ТИПИЗАЦИЯ

JavaScript — это динамически типизированный язык, что означает, что вы можете присваивать переменным тип, отличный от того, с которого они начались. Код ниже действителен.

**let foo = "string";**

**foo = 10; // changing to number**

**foo = false; // changing to boolean**

JavaScript позволяет вам это сделать, но вам это не обязательно. Возможно, вам будет легче понять код, если переменные обычно сохраняют один и тот же тип. Используете ли вы эту функцию динамической типизации или нет, зависит от вашего предпочтительного стиля кодирования.

Также стоит упомянуть, что другие языки программирования имеют разные типы чисел, например, представление целых чисел и чисел с плавающей запятой (дробных) с отдельными типами данных. Однако JavaScript этого не делает, и существует только один тип чисел, который, как уже упоминалось, является типом чисел с плавающей запятой.

## ТИП

JavaScript также предоставляет **typeof**оператор, который может сказать вам, к какому типу относится что-либо. Он возвращает строку.

**// Try entering in to the console:**

**typeof 100 // number**

**typeof "Hello" // string**

**typeof true // boolean**

Это может помочь вам определить тип объекта, если вы не уверены, например:

**let myVar = "Hello"; // start off with string**

**myVar = 100; // change to number**

**// ... lots more code ...**

**// What type is myVar?**

**console.log(typeof myVar); // number**

Специальный тип **undefined**также возник ранее, и у него **"undefined"**также есть свой собственный тип.

**// Try entering in to the console:**

**typeof undefined // "undefined"**

Это также будет тип переменной, которой никогда ничего не присваивалось.

**let unassignedVariable;**

**console.log(typeof unnasignedVariable); // undefined**

## ЗАКЛЮЧЕНИЕ

В этой части мы рассмотрели;

* Заявления
* Комментарии
* Переменные, присваивание и константы
* Числа, строки и логические типы данных
* Динамическая типизация
* Оператор**typeof**​

Надеюсь, теперь вы уже знакомы с вводом кода как в Construct, так и в консоли браузера. Руководство продолжит использовать оба варианта. В следующей части мы узнаем больше об операторах, логических значениях, сравнениях и типах преобразования.

## УЗНАТЬ БОЛЬШЕ

В конце каждой части этого руководства мы добавим ссылки, чтобы узнать больше о функциях JavaScript, упомянутых в [веб-документах MDN](https://www.construct.net/out?u=https%3a%2f%2fdeveloper.mozilla.org%2fen-US%2fdocs%2fWeb) , которые являются одним из лучших справочников по веб-разработке. Вы можете пропустить это и просто продолжить работу с руководством, если хотите, но если вас устраивает то, что вы узнали, и вы хотите копнуть глубже с более полной справкой или углубиться в более сложные детали, вы можете использовать эти ссылки, чтобы узнать больше.

# ИЗУЧИТЕ JAVASCRIPT В CONSTRUCT, ЧАСТЬ 3: ОПЕРАТОРЫ

Это третья часть серии руководств [«Изучите JavaScript в Construct»](https://www.construct.net/en/courses/learn-javascript-construct-79) . Эта часть является продолжением части 2. Если вы ее пропустили, см. раздел [«Изучаем JavaScript в Construct, часть 2: основы языка»](https://www.construct.net/en/tutorials/learn-javascript-construct-2802) .

В этой части мы продолжим использовать тот же проект, работая с фрагментами кода, введенными в Construct и в консоль браузера.

Если вы использовали таблицы событий Construct, вы можете заметить некоторое сходство между тем, как работают выражения Construct и выражения JavaScript. Однако имейте в виду, что выражения Construct не являются кодом JavaScript — это другая система, которая в некоторых отношениях работает аналогично.

## ОПЕРАТОРЫ

Во второй части мы использовали математические расчеты, такие как **10 + 5**. Вот **+**оператор сложения . Мы уже рассмотрели несколько операторов, таких как **-**вычитание, **\***умножение и **=**присваивание. В этой части мы рассмотрим ряд других операторов, которые часто используются в программировании.

### ДРУГИЕ МАТЕМАТИЧЕСКИЕ ОПЕРАТОРЫ

В JavaScript можно использовать еще два математических оператора:

* Остаток**%**
* Возведение в степень**\*\***

#### ОСТАТОК

Оператор **остатка** вычисляет остаток от деления, т.е. **a % b**вычисляет остаток от **a**деления на **b**. Например, **13 % 5**возвращается 3, потому что 13/5 равно 2 остатку 3. Другой способ подумать об этом: 13 — это 3 выше следующего наименьшего кратного 5 (то есть 10). Обратите внимание, что, несмотря на использование знака процента, это не связано с процентами.

**// Try entering in to the console:**

**13 % 5 // 3**

**5 % 2 // 1**

**27 % 10 // 7**

Это также работает с дробями в JavaScript:

**// Try entering in to the console:**

**5.5 % 2 // 1.5**

**9 % 3.5 // 2**

**3.25 % 1.25 // 0.75**

#### ВОЗВЕДЕНИЕ В СТЕПЕНЬ

Оператор возведения **в степеньa \*\* b** вычисляет **a**возведение в степень **b**. Например, **5 \*\* 2**возвращает **25**. Это также может работать с дробями — вы, возможно, помните, что возведение в степень половинной — это то же самое, что извлечение квадратного корня, поэтому **25 \*\* (1/2)**возвращается **5**.

**// Try entering in to the console:**

**6 \*\* 2 // 36**

**2 \*\* 3 // 8**

**100 \*\* (1/2) // 10**

**1.5 \*\* 2 // 2.25**

**6.25 \*\* 0.5 // 2.5**

Возможно, вы видели **^**символ, используемый для возведения в степень, но в JavaScript этот символ имеет другое значение, поэтому **\*\***используется вместо него.

В JavaScript также есть встроенные математические функции, такие как **Math.pow()**и **Math.sqrt()**которые вы также можете использовать вместо **\*\***.

## ОПЕРАТОРЫ ПРИСВАИВАНИЯ

Раньше мы **=**присваивали значение переменной. При работе с числами обычно присваивается переменная вычислению, которое включает в себя самого себя, например:

**let number = 7;**

**number = number \* 2;**

**console.log(number); // 14**

Посмотрите, как присваивание **number = number \* 2**присваивает номер вычислению, включающему его самого. В этом случае эта строка удвоит значение числа , поскольку она присваивает себе собственное значение, умноженное на 2.

В JavaScript для этого предусмотрены несколько дополнительных операторов присваивания, например **\*=**. Другими словами, **number \*= 2**это сокращение от **number = number \* 2**. Фрагмент кода ниже делает то же самое, что и раньше.

**let number = 7;**

**number \*= 2;**

**console.log(number); // 14**

Для каждого математического оператора, который мы пробовали до сих пор, существует специальный оператор присваивания:

* **a += b**такой же как**a = a + b**
* **a -= b**такой же как**a = a - b**
* **a \*= b**такой же как**a = a \* b**
* **a /= b**такой же как**a = a / b**
* **a %= b**такой же как**a = a % b**
* **a \*\*= b**такой же как**a = a \*\* b**

### ПРИСВАИВАНИЕ ВОЗВРАЩАЕТ ЗНАЧЕНИЕ

Интересно отметить, что присваивание фактически возвращает значение. Например, вы можете записать результат задания.

**let a = 2;**

**console.log(a += 2); // logs 4**

Присваивание возвращает значение переменной после завершения присваивания. В этом случае, поскольку **a**к нему добавлено 2, результатом присваивания является значение **a**после сложения, равное 4.

Это не имеет значения, если вы используете присваивание в отдельной строке, поскольку возвращаемое значение игнорируется. Однако это актуально для следующего раздела.

## УВЕЛИЧЕНИЕ И УМЕНЬШЕНИЕ

В программировании очень часто прибавляют или вычитают 1. Вы можете использовать это значение **a += 1**для увеличения **a**. Однако JavaScript предоставляет для этого дополнительное сокращение: **a++**.

**// These lines all do the same thing:**

**a = a + 1;**

**a += 1;**

**a++;**

Аналогично нужно **a--**вычесть 1.

Вы также можете поставить **++**или **--** перед именем переменной, например **++a**. Это делает почти то же самое, но возвращает другое значение:

* Постфиксная форма добавляет 1 к и возвращает старое значение до того , как было выполнено добавление.**a++a**
* Форма префикса**++a** добавляет 1 к **a**и возвращает новое значение после завершения добавления .

Вот пример кода, демонстрирующий разницу:

**let a = 1;**

**console.log(a++);**

**console.log(a);**

**1**Тогда это будет зарегистрировано **2**. В журнале **a++**будет отображено начальное значение **a**, а затем добавлено 1; в следующей строке снова отобразится значение, то есть 2. Вместо этого попробуйте изменить приращение на форму префикса:

**let a = 1;**

**console.log(++a);**

**console.log(a);**

**2**Тогда это будет зарегистрировано **2**. При ведении журнала **++a**сначала будет добавлено 1, а затем отображено результирующее значение 2. В следующей строке снова будет отображено значение, которое по-прежнему равно 2.

Поэтому, если вы используете возвращаемое значение присваивания **a += 1**и форму префикса, **++a**они одинаковы, поскольку они оба добавляют 1 и возвращают новое значение. Однако **a += 1**постфиксная форма **a++**не будет точно такой же, как **a++**при добавлении 1, но возвращает старое значение. Если вы не используете возвращаемое значение, например, если оно находится в отдельной строке, то **a += 1**, **++a**и **a++**все эквивалентны.

## СТРОКОВЫЕ ОПЕРАТОРЫ

До сих пор мы в основном рассматривали операторов, работающих с числами. Существуют также некоторые специальные применения операторов для строк.

Строки могут быть объединены , т.е. объединены вместе с помощью **+**. В этом случае, пока левая или правая часть **+**является строкой, строки объединяются. Только если и левая, и правая часть числа **+**являются числами, произойдет математическое сложение.

**// Try entering in to the console:**

**"Hello " + "world!" // Hello world!**

Таким образом вы также можете вставлять числа, включая переменные, при условии, что одна сторона **+**является строкой.

**let score = 100;**

**console.log("Your score is " + score + "!");**

**// Your score is 100!**

Строки, использующие обратный апостроф, **`**также имеют специальную функцию, позволяющую легко вставлять в текст другие значения с помощью **${value}**:

**let score = 100;**

**console.log(`Your score is ${score}!`);**

**// Your score is 100!**

Эти строки иногда называют литералами шаблона или строками шаблона , поскольку они похожи на текстовый шаблон, заполняемый значениями в определенных точках.

Эти функции являются полезными способами отображения значений в виде текста как для входа в консоль, так и для отображения информации пользователю. В этом руководстве также будут использоваться литералы шаблонов, чтобы упростить отображение сообщений, включая цифры.

### СМЕШИВАНИЕ ТИПОВ

Остерегайтесь распространенной ошибки — случайного смешивания типов. Обратите внимание **+**: если любая из сторон является строкой, выполняется конкатенация строк, включающая преобразование чисел в строку для их объединения. Иногда у вас может быть строка, содержащая последовательность цифр, поэтому она выглядит как число, например **"5"**. Однако это все еще строка, тип которой отличается от числа! Затем, если вы добавите что-нибудь к нему, вы получите конкатенацию строк.

**// Try entering in to the console:**

**"5" + "5" // "55" (string)**

**"5" + 5 // "55" (string)**

**5 + "5" // "55" (string)**

**5 + 5 // 10 (number)**

Это может произойти в менее очевидных обстоятельствах, например, если вы сохраняете строку, введенную пользователем, в переменную, а затем добавляете к этой переменной 1.

Поскольку такого рода ошибка может привести к запутанным результатам, обычно предпочтительнее использовать литералы шаблона вместо **+**создания строк и использовать их только **+**для математического сложения чисел.

## ПРЕОБРАЗОВАНИЕ МЕЖДУ СТРОКАМИ И ЧИСЛАМИ

JavaScript часто автоматически преобразует значения, например, строка + число автоматически преобразует число в строку, а затем добавляет его к первой строке. Однако иногда вам нужно намеренно преобразовать типы. Например, пользователь может ввести число в поле ввода текста, которое вернет строку, но вместо этого вы хотите использовать его как число. Также часто считается, что лучший стиль программирования — явное преобразование значений, а не полагаться на неявные функции языка, которые преобразуют значения за вас.

На самом деле существует множество способов явного преобразования значений в JavaScript, но сейчас мы сосредоточимся только на одном: использовании **Number()**и **String()**функциях. Использование **Number(string)**преобразует строку в число.

**// Try entering in to the console:**

**Number("5") // 5 (number)**

**Number("-6.5") // -6.5 (number)**

**Number(5) // 5 (still a number)**

**Number("Hello") // NaN**

Не забудьте обратить внимание на тип результатов — строки будут в кавычках, а числа — нет. В большинстве консолей браузеров для каждого типа также используются разные цвета, чтобы подчеркнуть разницу.

Обратите внимание, что в последнем числе мы снова получили специальный результат Not A Number (NaN). Это связано с тем, что строка «Привет» не представляет собой допустимое число, поэтому мы получаем специальный результат, указывающий на это.

Аналогично **String()**можно преобразовать число в строку.

**String(5) // "5" (string)**

**String(-6.5) // "-6.5" (string)**

**String("Hello") // "Hello" (still a string)**

**String(NaN) // "NaN" (string)**

Обратите внимание, что числа всегда можно преобразовать в строку. Если вы преобразуете специальное значение NaN в строку, вы получите строку с надписью «NaN». (Это преобразование прошло успешно; числовое значение было правильно преобразовано в соответствующую строку.)

Также обратите внимание, что в обоих случаях, если вы преобразуете тип в тот же тип, например **String("Hello")**, он просто возвращает одно и то же значение без изменений. Это полезно, поскольку если вы когда-либо не уверены, является ли значение строкой или числом, вы можете использовать **String(value)**, и вы знаете, что результат определенно является строкой.

## БУЛЕВЫ ОПЕРАТОРЫ

Помните, ранее мы рассмотрели логический тип, который представляет собой значения либо **true**или **false**. В этом разделе мы рассмотрим операторы, которые либо работают с логическими значениями, либо выдают логический результат.

### СРАВНЕНИЯ

JavaScript предоставляет несколько операторов сравнения, которые возвращают логическое значение. Например, **a < b**возвращается **true**, если **a**меньше **b**, в противном случае возвращается **false**. Операторы сравнения:

* Меньше **<**и больше чем**>**
* Меньше или равно **<=**и больше или равно**>=**
* Равно **==**и строгое равенство**===**
* Не равны **!=**и строги не равны**!==**

Обратите внимание: поскольку **=**используется для присваивания, нам нужно использовать что-то еще, чтобы проверить, действительно ли числа равны. Вот почему используются два или три равенства подряд, чтобы отличить сравнение от присваивания.

Почему существует два вида равенства? Стандартному методу равенства **==**разрешено преобразовывать типы, поэтому он может возвращать значения, **true**даже если типы различаются. Строгое равенство **===**не преобразует типы, поэтому всегда возвращает результат **false**, если типы различаются.

**// Try entering in to the console:**

**5 == 5 // true**

**5 == "5" // true (converts types)**

**5 === 5 // true**

**5 === "5" // false (types are different)**

Аналогичным образом **!=**можно преобразовывать типы, а **!==**при проверке на неравенство — нет.

**// Try entering in to the console:**

**5 != "5" // false (converts types)**

**5 !== "5" // true (types are different)**

Обычно считается лучшей практикой отдавать предпочтение сравнениям строгого равенства **===**и **!==**. Как отмечалось ранее, автоматическое преобразование типов может привести к запутанным результатам. Например, если вы сравниваете число со строкой, это часто является ошибкой, и вы хотели что-то преобразовать. Последовательное использование строгого равенства **===**и строгого не равно **!==**позволяет избежать путаницы, поскольку они включают сравнение типов. По этой причине в остальной части руководства будет предпочтительнее использовать строгие сравнения.

Распространенной ошибкой является путать присваивание **=**и равенство **==**или **===**. Например, если вы напишете **a = 5**вместо **a === 5**, вы фактически присвоите значение **a**вместо его сравнения. Постарайтесь убедиться, что вы используете правильные операторы!

Операторы меньше/больше не имеют строгих эквивалентов; к сожалению, они всегда могут конвертировать значения, **"3" < 5**например **true**. Нам просто приходится с этим жить, но, к счастью, проблема возникает реже, чем при проверке равенства.

Также стоит отметить, что строки можно сравнивать как меньше или больше. Строка считается меньшей, чем другая строка, если она предшествует ей в порядке сортировки. Например **"apple" < "banana"**, потому **true**, что в отсортированном списке яблоко будет стоять перед бананом (поскольку это сравнение основано на алфавитном порядке сортировки).

## ПРЕОБРАЗОВАНИЕ В ЛОГИЧЕСКОЕ ЗНАЧЕНИЕ

Строки и числа также можно преобразовать в логические значения, сведя их к значению **true**или **false**. Как и в случае с другими типами, это можно сделать с помощью **Boolean()**.

В JavaScript имеется множество правил преобразования различных типов данных и специальных значений в логические значения. Однако сейчас мы сосредоточимся только на следующих случаях.

* С **числами** и **0**специальным значением **NaN**преобразуется в **false**, а все остальное преобразуется в **true**.
* При использовании **строк** пустая строка **""**преобразуется в **false**, а все остальное преобразуется в **true**.
* Специальное значение **undefined**преобразуется в **false**.

См. следующие примеры.

**// Try entering in to the console:**

**Boolean(0) // false (is zero)**

**Boolean(NaN) // false (is NaN)**

**Boolean(1) // true (is not zero or NaN)**

**Boolean(42) // true (is not zero or NaN)**

**Boolean(Infinity) // true (is not zero or NaN)**

**Boolean("") // false (is empty string)**

**Boolean("Hello") // true (is not empty string)**

**Boolean("0") // true (is not empty string)**

**Boolean(undefined) // false**

**Boolean(true) // true (no change)**

**Boolean(false) // false (no change)**

В совокупности все значения, которые преобразуются в, **true**называются истинными значениями, а все значения, которые преобразуются в, **false**называются ложными значениями. Например, число 5 является правдивым, а пустая строка — ложной.

Обратите внимание, что правила истинности и ложности зависят от типа. Число **0**является ложным, поскольку оно равно нулю, но строка с содержимым **"0"**является правдивой, поскольку она не является пустой строкой.

## ЛОГИЧЕСКИЕ ОПЕРАТОРЫ

Есть некоторые операторы, которые особенно полезно использовать с логическими значениями.

### НЕТ

Вы можете инвертировать логическое значение – превратить его в **true**или **false**наоборот – с помощью оператора not **!**, который должен идти перед логическим значением, например **!b**.

**let b = true;**

**console.log(!b); // false**

### ИЛИ/И

Предположим, вы хотите узнать, истинно ли одно из двух условий или оба условия верны. Это можно определить с помощью логических операторов ИЛИ **||**и И.**&&**

Поскольку слова «или» и «и» регулярно встречаются в обычном письме, мы используем заглавные буквы OR и AND, чтобы было понятно, что мы говорим об операторах.

Например, **a < b || a < c**вернет **true**, если **a**меньше **b** или если **a**меньше **c**; в противном случае он возвращается **false**. Аналогичным образом **a < b && a < c**будет возвращено значение **true**if **a**is less then **b** и if **a**is less than **c**; в противном случае он возвращается **false**.

Вы можете использовать консоль для проверки всех комбинаций сравнений с логическими операторами:

**// Try entering in to the console:**

**false || false // false**

**false || true // true**

**true || false // true**

**true || true // true**

**false && false // false**

**false && true // false**

**true && false // false**

**true && true // true**

Эти операторы также работают с любыми истинными или ложными значениями, и результат также будет истинным или ложным, так же, как и с истинными и ложными. Например, **0 || 5**возвращает 5 (истинное значение) и **"Hello" && ""**возвращает «» (ложное значение). Это может показаться странным, но это может оказаться полезным при дальнейшем использовании таких вещей, как операторы if.

## УСЛОВНЫЙ ?: ОПЕРАТОР

Единственный тернарный оператор (оператор, который включает три значения) в Javascript — это условный оператор ?:. Это принимает форму **comparison ? valueIfTruthy : valueIfFalsy**. Например, следующий фрагмент кода будет регистрировать хороший результат! потому что переменная оценки 120 больше или равна 100.

**let score = 120;**

**console.log(score >= 100 ? "Nice score!" : "Try again!");**

Попробуйте изменить оценку для инициализации на **50**. Обратите внимание, что текст, записанный на консоль, меняется на « Повторите попытку!». , поскольку условие **score >= 100**стало ложным, поэтому возвращается последнее значение.

JavaScript часто принимает любое истинное или ложное значение при сравнении, и это применимо и здесь. Первое значение сравнения может быть другого типа, например числа или строки, и оно будет проверять, является ли значение правдивым или ложным.

**// Try entering in to the console:**

**0 ? "truthy" : "falsy" // "falsy"**

**1 ? "truthy" : "falsy" // "truthy"**

**"" ? "truthy" : "falsy" // "falsy"**

**"Hello!" ? "truthy" : "falsy" // "truthy"**

Еще раз, это пригодится, когда мы будем использовать операторы «если».

## ЗАКЛЮЧЕНИЕ

В этой части мы рассмотрели:

* Математические операторы **%**и**\*\***
* Дополнительные операторы присваивания, такие как **+=**и**\*=**
* Операторы увеличения и уменьшения **++**и**--**
* Создание строк с помощью **+**литералов шаблона и
* Преобразование между типами, в том числе между числами и строками, а также в логические значения и концепция истинных и ложных значений.
* Операторы сравнения, такие как **<**и**===**
* Логические операторы **!**, **||**,**&&**
* Условный **?:**оператор

В следующей части мы начнем использовать поток управления , такой как оператор if, о котором мы уже упоминали пару раз. Здесь вы можете начать писать программы, которые меняют свои действия в зависимости от условий — например, игру «Угадай число»!