# Занятие 1 - Введение в программирование. Hello, World!

*«Даже путь в тысячу* [*ли*](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8_(%D0%B5%D0%B4%D0%B8%D0%BD%D0%B8%D1%86%D0%B0_%D0%B4%D0%BB%D0%B8%D0%BD%D1%8B)) *начинается с первого шага.»*

— Лао-цзы

## Введение

Каждый день, в своей повседневной жизни мы используем компьютеры, иногда, даже сами того не замечая. Вычислительные машины разного размера, формы и мощности окружают нас, и это стало нормой. Мы привыкли к этому. Мы пользуемся смартфонами, компьютерами, планшетами, ноутбуками, телевизорами, автомобилями. В последнее время, появляются все больше "умных" устройств - часы, холодильники, кофеварки, и так далее. Задумывались ли мы, что лежит в основе работы таких устройств? Возможно, нет.

Современным инженерам приходится решать много задач, чтобы вдохнуть "жизнь" в устройство, начиная от проектирования электронного "железа", заканчивая написанием программ для него.

На этом курсе мы не будем вдаваться в подробности проектирования электроники, и даже не будем охватывать всего того многообразия, которое дает нам программирование, а всего лишь остановимся на одном аспекте: программирование для Web, иначе говоря, для сайтов, на языке JavaScript.

## Компьютер

Любой компьютер, будь то планшет, ноутбук, смартфон, или даже умная кофеварка, состоит из 2х основных частей:

* Hardware, Железо - аппаратное обеспечение,
* Software, Софт, ПО - программное обеспечение

### Железо

Аппаратное обеспечение - электронные и механические части вычислительного устройства, входящие в состав системы или сети, исключая программное обеспечение и данные (информацию, которую вычислительная система хранит и обрабатывает).

Наиболее важные элементы для понимания принципов работы:

* Центральный процессор - микрочип, который умеет выполнять инструкции (машинный код).
* Оперативная память - хранилище для данных, которыми оперирует процессор, пока программа выполняется.
* Основная память - память, в которой хранится сама программа и данные для нее - текст, рисунки, видео, звуки. В компьютере и ноутбуке это HDD - жесткий диск, в смартфоне, планшете - встроенная память и съемные SD-устройства (флешки).
* Устройства ввода-вывода. Здесь имеются в виду: клавиатуры, мыши как средства ввода информации (ведь мы набираем ими текст, указываем информацию мышкой, рисуем, создаем музыку и т.п.), монитор и принтер как средства вывода информации. Также, дисплей смартфона, колонки компьютера для вывода звука и так далее.

### Софт

Программа или множество программ, используемых для управления компьютером,

совокупность программ системы обработки информации и программных документов, необходимых для эксплуатации этих программ, все или часть программ, процедур, правил и соответствующей документации системы обработки информации,

компьютерные программы, процедуры и, возможно, соответствующая документация и данные, относящиеся к функционированию компьютерной системы.

Софт условно делится на группы:

* Системное ПО — комплекс программ, которые обеспечивают управление компонентами компьютерной системы, такими как процессор, оперативная память, устройства ввода-вывода, сетевое оборудование. В отличие от прикладного программного обеспечения, системное не решает конкретные практические задачи, а лишь обеспечивает работу других программ, предоставляя им сервисные функции, абстрагирующие детали аппаратной и микропрограммной реализации вычислительной системы.
* Прикладное ПО — программа, предназначенная для выполнения определённых пользовательских задач и рассчитанная на непосредственное взаимодействие с пользователем.
* Инструментальное программное обеспечение — программное обеспечение, предназначенное для использования в ходе проектирования, разработки и сопровождения программ, в отличие от прикладного и системного программного обеспечения.

Таким образом, можно проследить процесс запуска программы на выполнение:

1. Программа читается - копируется из основной памяти компьютера в оперативную память.
2. Исполняется - транслятор (переводчик) читает программу на языке программирования и переводит ее в инструкции понятные центральному процессору. Процессор выполняет каждую инструкцию шаг за шагом.
3. Промежуточные данные сохраняются в оперативной памяти.
4. Результат выполнения отображается на устройствах вывода, либо сохраняется в основную память.

## Программа

Что такое программа на JavaScript? Это список инструкций понятный исполнителю. Исполнитель на лету переводит инструкции с языка JavaScript на язык машинных кодов, которые исполняет центральный процессор.

Язык машинных кодов очень специфичен для процессора и классифицируется как язык низкого уровня, написание прикладных программ на языке низкого уровня затруднительно, что обусловило в результате развития вычислительной техники появление огромного количество языков программирования высокого уровня для написания прикладных программ.

Исполнителем программы является интерпретатор языка, в нашем случае JavaScript, а не центральный процессор, потому что программисту не видно разделения между процессом интерпретации и исполнения, интерпретаторы устроены так, что процессы перевода и исполнения тесно связаны внутри интерпретатора. Не смотря на то, что именно центральный процессор выполняет всю работу, мы будем считать исполнителем интерпретатор JavaScript.

Как и в жизни, существует множество различных языков, только вместо "человеческих" языков, компьютеры понимают языки "программирования". Также как и в человеческих языках, языки программирования подчиняются правилам, но в отличие от человеческого - этот язык должен быть точен.

Если сравнивать человеческий и язык программирования, если ваш начальник пришлет вам письмо, написанное неграмотно и с опечатками, вы все-же сможете понять, о чем речь, и выполнить инструкции, написанные в письме. Компьютер этого сделать не сможет, ему необходим точный набор инструкций, написанный без опечаток и без ошибок, с соблюдением синтаксиса языка.

В большинстве случаев программы создаются для решения каких-либо задач, например, скачать картинку с сайта (браузер типа Google Chrome), прочитать документ (MS Word) или отправить e-mail другу (Outlook), посмотреть серию любимого сериала (Media Player Classic), нарисовать картинку (Paint, Photoshop).

Во всех этих случаях у нас есть входные и выходные данные:

* Вводим название сайта, получаем визуальное отображение сайта
* Вводим адрес картинки, получаем визуальное отображение картинки
* Вводим текст в редактор с клавиатуры, получаем документ
* Открываем видео-файл, получаем изображение на экране и звук в колонках

Получается, что программа по большей части - это некий "черный ящик", который берет входные данные и выдает выходные. Для пользователя совершенно незаметно, и не важно, что же происходит внутри. Но нам, как будущим разработчикам таких программ, это важно.

Также, как вы могли заметить, большинство задач, описанных выше часто решаются с помощью одного лишь веб-браузера:

* Прочитать документ - Google Docs или Microsoft Office 365
* Отправить e-mail - Gmail, mail.ru.
* Посмотреть видео, слушать музыку - Youtube
* Смотреть сериалы - Netfilx, ts.kg.

Выбранный нами язык программирования JavaScript работает в любом современном браузере и с его помощью можно решить очень много задач. И более того

## Инструменты

Мы в процессе курса будем пользоваться компьютером как уже готовым средством, созданным для нас инженерами, и его операционной системой.

Операционная система - это самая главная программа, которая запускается при запуске компьютера, и под ее управлением запускаются все остальные (браузеры, игры, видео-плееры и т.п.)

Работать мы будем в браузере Google Chrome, а редактировать исходные коды - в текстовом редакторе Visual Studio Code, а на более поздних этапах курса в специальной программе для разработки - WebStorm.

Мы будем использовать интерпретатор JavaScript, он будет для нас исполнять наши программы, выдавать нам ошибки, если не сможет понять код нашей программы, а также выдавать ошибки в процессе выполнения программы, например при делении на 0 или другой ситуации, которую интерпретатор не запрограммирован обрабатывать.

Скачать и установить Visual Studio Code: <https://code.visualstudio.com/>

Кстати, Visual Studio Code тоже написан на JavaScript!

## Информация. Экскурс.

Попытаемся понять, как же на самом деле компьютер хранит в себе информацию и может ей манипулировать?

Подумаем, как бы мы могли отобразить какие-нибудь данные, например, число 42? Мы уже пользуемся определенным методом "шифрования" в жизни - когда мы пользуемся привычными арабскими цифрами, и написанное число "42" у нас автоматически распознается как таковое. Мы понимаем, что это, например, 42 спички или 42 монеты, и можем даже прикинуть в уме, как это число выглядит в реальных объектах. Еще мы могли бы посчитать как древние люди и шифровать каждую единицу с помощью палочки. Тогда это число выглядело бы так:

||||||||||

||||||||||

||||||||||

||||||||||

||

Такая система счисления называется "**унарной**" - то есть состоит всего из **одной** единицы информации, с помощью которой мы записываем данные.

Наиболее общепринятой системой счисления среди людей является, конечно, десятичная, именно та, которой мы привыкли пользоваться. Она называется **десятичной**, потому что у нее **десять** символов (цифр) для записи информации:

0 1 2 3 4 5 6 7 8 9

С помощью этих десяти цифр мы легко записываем привычные нам числа, как 42, 123 и даже 1000000.

Каждая из цифр в числе означает количество единиц, десятков, сотен и т.п., например для числа **42**:

10 1

**4** **2**

10 \* 4 + 1 \* 2 = 42

Для числа **246**:

100 10 1

**2** **4** **6**

100 \* 2 + 10 \* 4 + 1 \* 6 = 246

Бинарная система счисления. Из названия понятно, что используется всего две цифры для записи информации - 0 и 1. Вы уже, возможно, знаете, что все в компьютере хранится в виде нулей и единиц. Попробуем зашифровать несколько чисел в бинарной системе:

1

**1**

1 \* 1 = 1

Здесь все понятно. Для шифрования чисел 0 и 1 мы можем использовать их же. А что, если надо зашифровать число два? Мы увеличиваем разрядность числа:

2 1

**1 0**

2 \* 1 + 0 \* 1 = 2

Получается, что для записи числа 2 в бинарном виде мы запишем "10". Попробуем что-то сложнее:

8 4 2 1

**1 0 1 1**

8 \* 1 + 4 \* 0 + 2 \* 1 + 1 \* 1 = 11

Запись десятичного числа 11 в бинарном виде выглядит как "1011"

Число 42:

32 16 8 4 2 1

**1 0 1 0 1 0**

32 \* 1 + 16 \* 0 + 8 \* 1 + 4 \* 0 + 2 \* 1 + 1 \* 0 = 42

42 в десятичной = "101010" в двоичной.

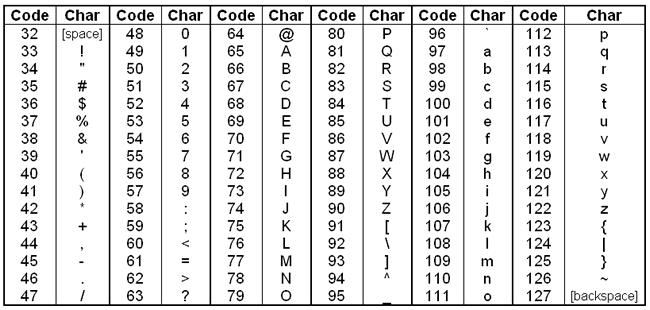
Здесь, каждый последующий разряд является следующей степенью числа 2, а 1 или 0 - показывает, входит это число в сумму или нет.

Теперь, когда мы узнали, что в компьютере данные хранятся в двоичной системе и мы можем охарактеризовать хранение информации:

* Каждая единица или ноль - это один бит
* Каждые 8 бит - это один байт. С помощью одного байта (8 нулей или единиц) можно зашифровать число от 0 до 255
* Дальше вы знаете: 1000 байт = 1 кБайт, но не совсем. В компьютерном мире принято измерять кило- мега- и т.п. единицы не в тысячу раз большую, чем предыдущую, а в 1024, т.е.:
  + 1 Кбайт (килобайт) = 1024 байт
  + 1 Мбайт (мегабайт) = 1024 Кбайт (1024 \* 1024 байт)
  + 1 Гбайт (гигабайт) = 1024 Мбайт (1024 \* 1024 \* 1024 байт)

Теперь, когда мы умеем шифровать числа, мы можем с помощью чисел также шифровать и данные. Например, что мы можем придумать для шифрования текста?

Мы знаем, что с помощью 1 байта можно зашифровать число от 0 до 255. В современном английском алфавите всего 26 букв. Если даже мы возьмем все заглавные и прописные буквы, цифры, специальные символы - нам хватит чисел, чтобы сопоставить каждому какой-то из этих символов. Например:



Такая таблица, конечно, уже существует, и называется ASCII. Расшифровка:

**American Standard Code for Information Interchange.**

Таким образом, если мы захотим зашифровать сообщение, например:

Hello!

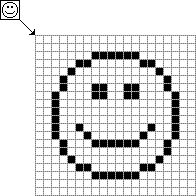
То мы запишем его в виде 6 чисел:

72 101 108 108 111 33

Каждое из этих чисел может быть зашифровано с помощью одного байта, а значит длина такого сообщения: 6 байт.

Для других языков мира используется более продвинутая система символов UTF-8, в которой каждый символ занимает для хранения от 8 до 32 бит (от 1 до 4 байт), и с помощью которой можно зашифровать любой символ любого языка в мире.

Изображения также можно зашифровать с помощью чисел (или в нашем случае, байт). Все изображения на экране состоят из пикселей, или мелких точек. Если приглядеться поближе, они выглядят, например, вот так:



В случае такого монохромного изображения (одноцветного), то есть у нас есть либо белый цвет, либо черный, мы можем шифровать пиксели напрямую с помощью бит:



0 0 0 0 0 0

0 0 1 1 0 0

0 1 0 0 1 0

0 1 0 0 1 0

0 0 1 1 0 0

0 0 0 0 0 0

Если же изображение у нас цветное, то мы должны шифровать каждый пиксель с указанием его цвета по очереди. Наиболее распространенный формат: RGB, при котором мы для указания цвета используем три байта: три числа от 0 до 255, каждое из которых выражает количество определенного цвета из трех основных:

* Красного (Red)
* Зеленого (Green)
* Синего (Blue)

Здесь хорошо работает пример из реального мира: если мы возьмем 0 красного, 0 зеленого и 0 синего, получим черный цвет, а смешать максимум всех цветов, то белый. В нашем случае, это 255, 255 и 255.

Получается, что цветное изображение будет зашифровано в виде трех байт на каждый пиксель. Так работает, например, формат BMP.

И так далее, в зависимости от контекста в компьютере данные шифруются в виде бит и байт, и означают различные вещи.

## Алгоритмы

При написании программ, мы все время будем сталкиваться с таким понятием, как "**алгоритм**". Алгоритм - это некоторая последовательность действий, выполняя которые можно получить некий результат.

Алгоритмы бывают **линейными** и **нелинейными**.

Пример линейного алгоритма:

1. Взять продукты для салата
2. Приготовить салат.
3. Съесть салат.

Линейные алгоритмы не подразумевают, что что-то может пойти не так.

Нелинейные алгоритмы часто усложняют поведение линейных алгоритмов в зависимости от некоторых условий. Например:

1. Если нет нужных продуктов для салата:
   1. Сходить на базар за продуктами
2. Приготовить салат
3. Съесть салат

В этом случае, мы проверяем сначала условие, есть ли у нас или нет нужные продукты для приготовления салата. Если их нет, мы отправляемся на базар. Если есть, соответственно сразу переходим к шагу 2. Такое поведение называется **ветвлением**.

Также, алгоритмы могут быть еще более сложными. Например, алгоритм поиска фамилии в телефонной книге:

1. Открыть телефонную книгу
2. Посмотреть, есть ли на текущей странице нужная фамилия
3. Если фамилия не найдена, то:
   1. Перелистнуть страницу
   2. Перейти к шагу 2
4. Если она найдена, то:
   1. Закончить поиск.

Такой алгоритм включает в себя **цикл**, когда мы возвращаемся на шаг 2 потенциально много раз, пока не найдем нужную фамилию, а также **ветвление**, которое определяет, закончился ли наш алгоритм или нет.

Компьютерное программирование - это процесс построения и проектирования исполняемой компьютерной программы для выполнения конкретной задачи.

Программирование включает в себя такие виды деятельности, как анализ поставленной задачи, формулирование конечного результата, генерация алгоритмов, реализация алгоритмов на выбранном языке программирования, причесывание кода, оптимизация использования ресурсов.

Цель программирования - найти последовательность инструкций, которая автоматизирует выполнение задачи для решения данной проблемы, а также обеспечить понятность кода для человека для внесения изменений в код программы в будущем.

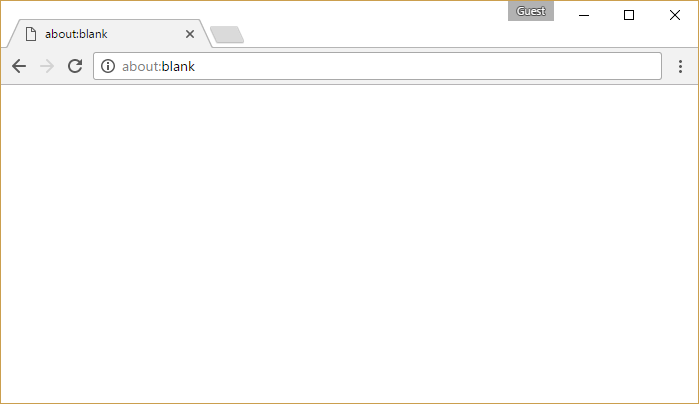
Сначала мы научимся писать программы понятные компьютеру, а позже научимся причесывать код, так чтобы было понятно человеку.

Как описывать нужные нам действия, с использованием ветвлений и циклов, мы разберем на следующих занятиях.

## Hello, World!

Попробуем написать нашу первую программу на JS, используя браузер Google Chrome.

Откроем страницу в браузере, введя about:blank в панель адреса. Это специальное обозначение пустой страницы, на которой ничего нет.

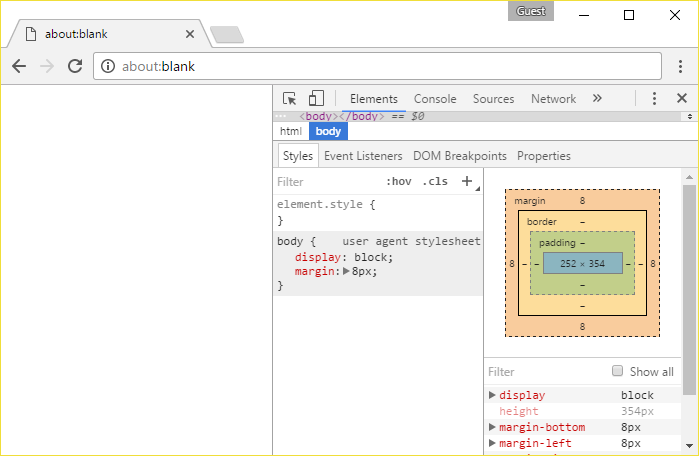


Далее, откроем панель инструментов разработчика, которой будем пользоваться часто в дальнейшем.

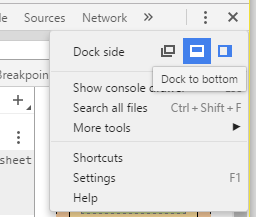
Для этого нажмем:

* На Windows, Linux: Ctrl + Shift + I
* На Mac: Cmd + Opt + I

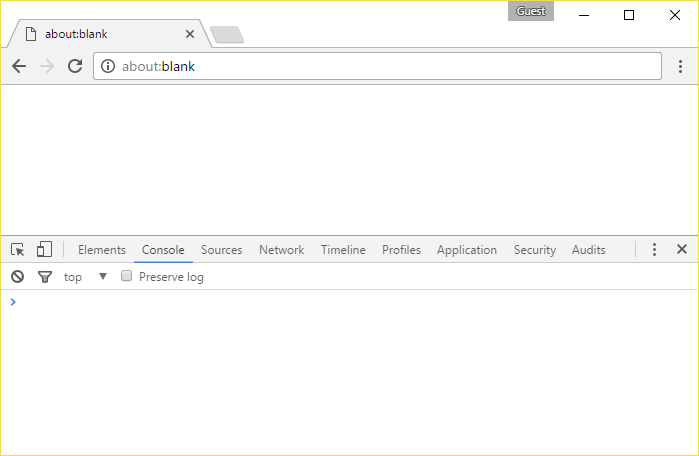
Должно открыться такое окно:



Для удобства просмотра информации, переместим панель в нижнюю часть страницы. Для этого нажмем на кнопку "Меню" (три точки) и затем "Dock to bottom"



Также, переключимся во вкладку "Console". В результате должно получиться так:

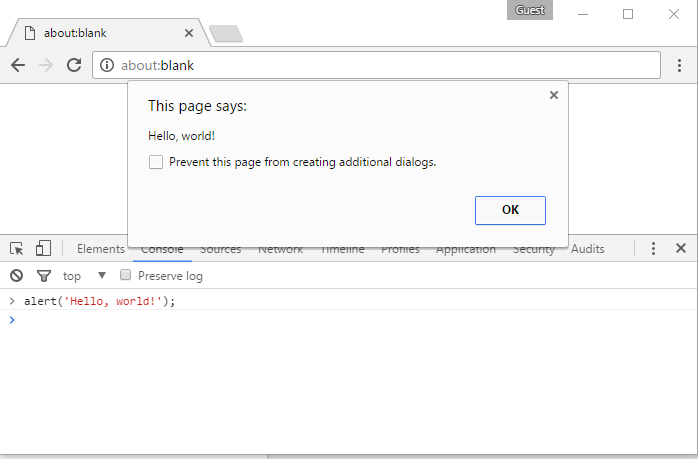


В таком режиме открывается консоль JavaScript, которая готова принимать команды: символ > и мигающий курсор после него.

Перепишем туда в точности вот такую строку:

alert('Hello, world');

И нажмем Enter на клавиатуре. Мы должны получить такой результат:



Поздравляем, вы сделали самый первый, хоть и маленький шаг к мастерству программирования.

Не вдаваясь в подробности, мы выполнили программу на языке JavaScript, состоящую из одной строки. Эта строка сообщает браузеру, что нужно вывести на экран стандартное сообщение (которые мы в будущем будем называть alert-ами), со строковым значением "Hello, world!". Точка с запятой в конце говорит об окончании текущей команды.

В следующих уроках мы будем разбирать в подробностях, что означает все вышеперечисленное.

А сейчас попробуем сделать то же самое, но запустив уже программу из файла. Будучи знакомыми с версткой, вы должны суметь создать файл index.html и написать в него следующий код. Воспользуйтесь редактором Atom, который вы только что установили.

<!DOCTYPE HTML>

<html>

<head>

<title>Hello, world!</title>

</head>

<body>

<script type="text/javascript">

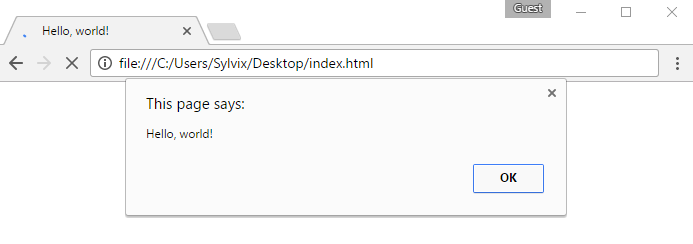
alert('Hello, world!');

</script>

</body>

</html>

Откройте файл в браузере двойным щелчком или перетащив файл в новую вкладку. Результат должен быть предсказуем:



Таким образом, JavaScript запускается внутри тега <script> в тот момент, когда до него доходит браузер.

Действие данного кода предсказуемо одинаковое.

# Основы JS. Типы данных. Простые операции.

## Псевдокод

Мы узнали, что такое "алгоритм" и попробовали свои силы в их написании при выполнении домашнего задания. На самом деле, мы постепенно переходим от программирования "на словах" к программированию собственно в коде, все более формализуя шаги выполнения алгоритмов. Следующий уровень формализации - это так называемый "псевдокод", который по сути является написанием программы на человеческом языке.

Итак, представим себе следующую бытовую задачу: в кафе необходимо подсчитать стоимость заказа клиента с процентами за обслуживание. Алгоритм подсчета на псевдокоде будет выглядеть так:

Заказ клиента:

1 плов, стоимостью 1000 тенге

1 салат, стоимостью 500 тенге

1 чайник чая, стоимостью 150 тенге

3 кусочка хлеба, стоимостью 50 тенге

Общая стоимость заказа = 0

Процент обслуживания = 10%

Для каждого элемента заказа:

стоимость элемента = стоимость блюда \* количество блюд

общая стоимость заказа = общая стоимость заказа + стоимость элемента

Общая стоимость заказа = общая стоимость заказа + Процент обслуживания

Вывести общую стоимость заказа

Получается, что подобный подсчет очень легко формализуется с помощью фраз на обычном человеческом языке, однако, с использованием определенной структуры. Здесь можно заметить, что при подсчете мы используем промежуточные значения, а также цикл. Промежуточное значение "общая стоимость заказа" накапливается с течением времени, и затем выводится в конце алгоритма. Также, можно заметить, что данная "программа" тоже выполняет описанную в прошлом занятии функцию - она берет **входные данные**, в виде заказа клиента, делает с ними что-то, а затем выдает **результат** - в виде рассчитанной общей стоимости заказа, который имеет практическую пользу - мы можем сказать клиенту, сколько денег он должен заплатить.

В дальнейшем мы будем часто сталкиваться с различными **алгоритмами**, которые будут представлять из себя **последовательность действий**, (вычислений и прочих манипуляций с данными, объектами на экране и т.п.), и поэтому следует расценивать такие термины как синонимы.

При определенной сноровке, вы сможете писать программы сразу на "компьютерном" языке, то есть, языке программирования, на JavaScript, вместо псевдокода. Вы сможете переписывать программы с псевдокода на JavaScript и обратно. Но пока вы учитесь, бывает очень полезно прорабатывать последовательность действий на бумаге, или в текстовом документе, на человеческом языке (на каком вы говорите и думаете чаще всего), а уже потом писать программу.

## Языки программирования

На прошлом занятии мы рассмотрели как информация шифруется в компьютере - с помощью нулей и единиц. На самом деле процессор тоже работает с нулями и единицами и выполняет инструкции, зашифрованные ими же. И процессор понимает только один язык - машинный код. Он выглядит примерно так:

01111111 01000101 01001100 01000110 00000010 00000001 00000001 00000000  
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  
00000010 00000000 00111110 00000000 00000001 00000000 00000000 00000000  
10110000 00000101 01000000 00000000 00000000 00000000 00000000 00000000  
01000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  
11010000 00010011 00000000 00000000 00000000 00000000 00000000 00000000  
00000000 00000000 00000000 00000000 01000000 00000000 00111000 00000000  
00001001 00000000 01000000 00000000 00100100 00000000 00100001 00000000  
00000110 00000000 00000000 00000000 00000101 00000000 00000000 00000000

...

Это конечно, несравнимо по читаемости с тем, что мы делали на прошлом занятии:

alert('Hello, world!');

Переводить подобный **человеко-читаемый** код в машинный нам помогают специальные программы, написанные другими людьми, которые называются **трансляторами**. JavaScript - является **интерпретируемым**. То есть программу на JavaScript строка за строкой читает и исполняет (**интерпретирует**) другая программа, в нашем случае - интерпретатор Javascript встроенный в браузер. Поэтому, мы получим выполнение кода JavaScript в рамках HTML-страницы, и сможем в дальнейшем изменять и создавать элементы на ней.

Конечно, где-то глубоко внутри браузера, каждая строка JavaScript так или иначе выполняется процессором в виде нулей и единиц, но нам это не так важно, как в некоторых других языках программирования.

Итого, получается, что все языки программирования - создаются в первую очередь для людей, а не для компьютеров, и в результате для своей работы все равно преобразуются в машинный код, который человеку прочитать и написать сложно, с помощью специальных программ - **компиляторов** или **интерпретаторов**, которые были написаны другими людьми ранее.

## Переменные

Промежуточное значение, которое мы рассмотрели в предыдущем параграфе, в языках программирования называется **переменной**. Она так называется именно потому, что она может менять свое значение в ходе работы программы, наподобие того как меняется общая сумма заказа в предыдущем примере.

В языке JavaScript переменная - это некое хранилище для данных, имеющее свое собственное имя, с помощью которого мы этими данными можем пользоваться.

Чтобы объявить переменную, мы напишем следующее:

**let totalOrderSum = 0;**

1 2 3 4

1. Сначала идет ключевое слово let, которое означает о том, что мы объявляем новую переменную.
2. Затем через пробел идет имя переменной, в нашем случае totalOrderSum. Названия переменных могут содержать английские буквы, числа, некоторые символы (\_, $). Названия переменных не могут начинаться с чисел. Следует называть переменные со смыслом - имя должно однозначно характеризовать информацию, которую она в себе **хранит**.
   * Пример допустимых названий переменных:
     + \_privateVariable
     + price
     + $element
     + agent\_007
   * Пример недопустимых названий переменных:
     + 5th\_element
     + 1love
3. Одиночный знак равенства в программировании означает "присвоить", то есть присвоить значение после этого знака переменной до знака.
4. Значение в нашем случае - это число 0 (ноль). Это как раз и есть та информация, которая **хранится** в переменной. На данном этапе мы можем сказать, что переменная **totalOrderSum** хранит в себе значение 0.

Также, переменная может быть объявлена вообще без значения, в случае, если ее потом понадобится использовать:

let userName;

Таким образом, можно сказать, что мы создали пустое хранилище, пустую коробочку с надписью "userName", в которую мы потом сможем положить какое-то значение.

После того, как мы объявили переменную, с помощью ключевого слова let, мы можем присваивать ей новые значения, уже без этого ключевого слова:

let orderNumber = 123;

// … что-то делаем с номером заказа 123.

orderNumber = 456; // теперь номер заказа равен 456.

Переменную можно также объявить, используя ключевое слово const (константа). Развница в том, что let можно присвоить новое значение, а const - нет.

const seconsdsPerMinute = 60;

В старых программах вы также можете встретить объявление переменных через var, но мы будем пользоваться новым синтаксисом во всех программах.

## Точка с запятой, конец мысли

Обратите внимание на знак точки с запятой, стоящий после каждой завершенной команды в JavaScript. Как вы помните, его мы также использовали на первом занятии, когда выводили на экран строку "Hello, world!". Этот знак означает - конец выражения. Мы будем ставить его в основном везде в конце строки, кроме некоторых исключений, там, где конца выражения еще нет, или конец выражения определяется другим символом.

Привыкайте к правильной расстановке точки с запятой. Несмотря на то, что в JavaScript этот знак не обязателен, в дальнейших занятиях мы рассмотрим, почему его лучше ставить.

## Типы данных

### Целое число

В прошлых примерах мы рассмотрели, что переменным можно присваивать некоторые значения. В этих примерах мы пользовались в основном числами, такими как 0, 123, 456 и так далее. Когда мы записываем число без кавычек, этим самым мы говорим о том, что это - **целое число** (Integer).

const meaningOfLife = 42;

const orderNumber = 123;

const orderPrice = 400;

С целыми числами можно производить математические операции, такие, как вы привыкли видеть в школе:

|  |  |  |
| --- | --- | --- |
| + | сложение | > 2 + 2  < 4 |
| - | вычитание | > 100 - 99  < 1 |
| \* | умножение | > 5 \* 7  < 35 |
| / | деление | > 20 / 4  < 5 |
| % | остаток от деления | > 3 % 2  < 1 |

Мы можем открыть консоль Chrome, как мы это делали на прошлом занятии:

* На Windows, Linux: Ctrl + Shift + I
* На Mac: Cmd + Opt + I

И попробовать вышеописанные операции прямо в ней. В дальнейшем, в раздаточных материалах символы в начале строки > и < будут означать ввод в консоль и вывод из консоли соответственно.

### Строка

Следующим из наиболее распространенных типов данных является строка. Строковый тип позволяет хранить в переменных текстовые данные:

const message = 'Hello, world!';

const greetings = "Congratulations!";

const yearOfBirth = '1989';

Как вы заметили, строки начинаются и заканчиваются на символ "одинарная кавычка" или "апостроф", либо на "двойную кавычку". Важно, чтобы начальный и конечный символ были одинаковыми. Принципиальной разницы между кавычками нет, но мы будем использовать в основном одинарные кавычки.

Как вы заметили, мы можем в строке также хранить и число. Какая же принципиальная разница между числом и строкой в этом случае? Дело в том, что в зависимости от типа данных, меняется и поведение этих данных при различных операциях. Например, мы уже знаем, что можем сложить два целых числа:

> 123 + 3

< 126

Что будет, если мы сложим две строки?

> '123' + '3'

< "1233"

Правильно, строки объединятся. Получается, что в строковом типе число '123' хранится как три текстовых символа (как мы проходили ранее, закодированных, например, с помощью например таблиц ASCII или UTF-8), а не как число.

В программировании это называется **конкатенацией**, когда мы объединяем две и более строк.

> 'Hello, ' + 'John!'

< "Hello, John!"

### Дробные числа

Кроме целых чисел существуют также дробные. Дробные образуются автоматически, если при делении целых чисел получается дробное:

> 12 / 5

< 2.4

А также мы можем ввести их самостоятельно, использовав точку, как разделитель целой и дробной части:

const pi = 3.14;

C дробными числами, как и с целыми, можно производить такие же математические операции.

## Порядок операций

В JavaScript, также как и в математике, существует определенный порядок, которому подчиняются все операции. С математическими операциями все просто - порядок в них точно такой же, как и в математике - сначала умножение и деление, затем сложение и вычитание. Также как в математике, мы можем воспользоваться скобками, чтобы вручную назначить приоритет некоторых операций.

Классический пример:

> 2 + 2 \* 2;

< 6

> (2 + 2) \* 2;

< 8

В случае же с операцией присваивания, у нее самый низкий приоритет, то есть при присваивании этот оператор будет ждать, пока не выполнится все, что справа от него, и только потом полученное значение будет присвоено переменной:

let result = (3 \* 6 + 8 \* 12) / 4;

5 1 3 2 4 -> порядок операций

Переменной result будет присвоено значение только в самом конце - когда будет посчитано значение справа от оператора присваивания.

## Операции с переменными

Теперь мы знаем, какие бывают основные типы данных в JavaScript (на самом деле их больше, но с остальными мы познакомимся немного позже) и умеем производить некоторые вычисления с данными. Но JavaScript - это все-таки нечто большее, чем хитроумный калькулятор. Зная, что мы можем также присваивать значения переменным, мы можем использовать их в тех же операциях. Например, конкатенация переменных, содержащих строки со строками:

const name = 'Jack';

const message = 'Hello, ' + name + '! Welcome to JS!';

alert(message);

Как видите, мы задали переменную name со строковым значением "Jack", затем собрали из двух строк и одной переменной одно сообщение, которое потом вывели с помощью alert на экран.

Таким образом, переменные созданы для того, чтобы ими манипулировать, чтобы ими пользоваться. То есть значение переменной при ее использовании как-бы подставляется вместо самой этой переменной.

## Получение данных пользователя

Пока что мы писали код, который не особо зависел от пользователя. Зная основы HTML и побывав на различных сайтах, вы понимаете, что пользователь может взаимодействовать с сайтом множеством различных способов - вводя текст в поля, выбирая мышью значения и так далее. Мы же рассмотрим сейчас один из самых простых способов ввода данных - через специальное окошко, которое умеет показывать браузер, наподобие окна alert.

Функция для вызова такого окна называется prompt, и мы будем называть его точно так же. Посмотрим на пример:

const name = prompt('Enter your name');

const message = 'Hello, ' + name + '! Welcome to JS!';

alert(message);

Пример практически одинаков с тем, что мы разбирали ранее, только теперь имя получается с помощью вызова функции prompt, которая запрашивает имя у пользователя.

Запрошенное имя мы храним в переменной name, которой присваивается значение, которое вводит пользователь.

Затем это значение мы будем использовать для конкатенации сообщения.

Теперь наша программа стала "умнее", и может выдавать разный результат в зависимости от введенных пользователем данных.

Следует отметить, что тип введенных пользователем данных с помощью prompt - это всегда строка. Если мы хотим получить число для расчетов, нам придется обработать введенные данные с помощью еще одной встроенной в JavaScript функцией - parseInt (дословный перевод - получить Целое Число). Посмотрим на пример:

let number = prompt('Please, enter a number:');

number = parseInt(number);

let squared = number \* number;

alert('Square of your number = ' + squared);

Функция parseInt может понять только целое число. Если вы введете дробное - то получите только целую часть от него. Если вы введете совсем не число, то получите специальное значение NaN - означающее в JavaScript, что получилось "не число".

Различные вариации на тему, как понять, что пользователь ввел не число, а что-то другое, а также про NaN мы рассмотрим на следующих занятиях.

Еще пример:

let name = prompt('Please, enter your name:');

let age = prompt('Please, enter your age:');

age = parseInt(age);

let message = 'Hello, ' + name + '! You are ' + age + ' years old!';

alert(message);

Здесь мы получаем два значения от пользователя сразу друг после друга. Как можно заметить, когда срабатывает функция prompt, браузер ждет ввода пользователя, и только потом продолжает действие программы. Поэтому, если запустить данную программу в браузере, два окна не выскочат одновременно, а будут появляться друг за другом.

Затем мы храним эти данные в переменных, которые потом собираем в единое сообщение с помощью конкатенации, и отображаем на экране.

## Комментарии

В программировании часто бывает нужно кроме текста программы написать в нее некоторые заметки или пояснения для коллег - других программистов, чтобы разъяснить какую-то часть алгоритма или поведения. Для этого существуют комментарии - строки, начинающиеся с двойного символа "слэш" ( / ) :

// Это комментарий. Он никак не повлияет на код.

const image = '/images/123.jpg';

// alert(image);

В примере, alert не выполнится, поскольку он "закомментирован". Часто такой подход применяют, чтобы временно убрать часть кода, не удаляя его, чтобы можно было потом быстро восстановить.

Многострочные комментарии также возможны:

/\*

Это комментарий. Он может растянуться на много строк

И это все еще комментарий.

Тут что-то еще.

\*/

alert('А это уже не комментарий');

## Оформление кода. Чистый код. Названия переменных

Вот мы уже и знаем немного синтаксиса JavaScript: как объявлять переменные, как производить с ними некоторые манипуляции. Это значит, что мы уже освоили некоторый **синтаксис** языка. Теперь поговорим об оформлении нашего кода.

Почему это важно? Дело в том, что плохо оформленный, "грязно" написанный код читается намного хуже, чем красивый и аккуратный. И пока вы не привыкли к какому-либо стилю написания кода, мы поговорим о том, как это делать правильно, чтобы вы сразу привыкали к хорошему.

Разделяйте все операторы пробелами:

const sum = 2 + 3;

Между знаками "равно", "плюс" и числами есть пробелы. JavaScript сработает и без них, но код будет написан неопрятно, слишком сжато.

Ставьте в конце строки точку с запятой.

Называйте ваши переменные понятно и в меру развернуто, чтобы было понятно, что именно хранится в этой переменной. Например, плохо:

const a = 3;

const b = 4;

const c = a + b;

Хорошо:

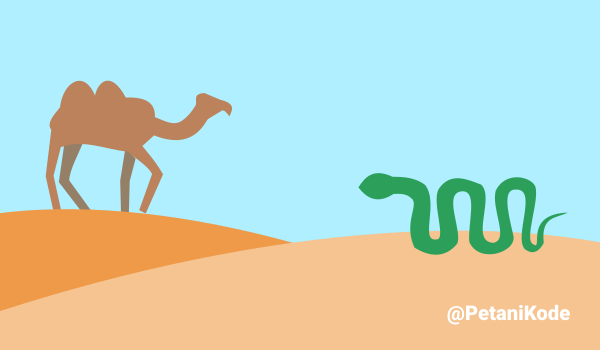
const firstTrip = 3;

const secondTrip = 4;

const totalTripDistance = firstTrip + secondTrip;

Используйте camelCase для создания переменных, которые состоят из двух слов и более. Это означает, что первое слово начинается с маленькой буквы, а все последующие слова написаны вплотную и начинаются с заглавной буквы (как горбы у верблюда). В противопоставление ему обычно приходится так называемый snake\_case, когда все слова пишутся с маленькой буквы, а между словами ставится символ подчеркивания, как разделитель.

Мы будем пользоваться Camel Case во всех случаях.



Не пишите много комментариев. Если вы делаете это для себя, чтобы разобраться в программе - удалите их после того, как разобрались. Не нужно писать в комментариях очевидные вещи:

const x = 0; // Присваиваем переменной x значение 0

Комментарии в основном нужны для других читателей кода, но будет лучше, если ваш код будет говорить сам за себя - с помощью правильных и красивых названий переменных и чистого кода. В дальнейших занятиях будут рассмотрены другие аспекты чистого кода и красивого оформления.

## Полезные ссылки

1. [When to use double or single quotes in JavaScript?](https://stackoverflow.com/questions/242813/when-to-use-double-or-single-quotes-in-javascript)
2. [Псевдокод (язык описания алгоритмов)](https://ru.wikipedia.org/wiki/%D0%9F%D1%81%D0%B5%D0%B2%D0%B4%D0%BE%D0%BA%D0%BE%D0%B4_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BE%D0%BF%D0%B8%D1%81%D0%B0%D0%BD%D0%B8%D1%8F_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%BE%D0%B2))
3. [JavaScript Operators Reference](https://www.w3schools.com/jsref/jsref_operators.asp)
4. [Число с плавающей запятой](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%BE_%D1%81_%D0%BF%D0%BB%D0%B0%D0%B2%D0%B0%D1%8E%D1%89%D0%B5%D0%B9_%D0%B7%D0%B0%D0%BF%D1%8F%D1%82%D0%BE%D0%B9)
5. [Технический долг](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%85%D0%BD%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B4%D0%BE%D0%BB%D0%B3)
6. [Динамическая типизация](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F)