Видео недели:

1. [Что такое JavaScript?](https://www.youtube.com/watch?v=vTM8qrnWKhY)
2. [Как подключать скрипты](https://www.youtube.com/watch?v=hH7Xo_IxzAQ)
3. [Переменные, константы и типы данных](https://www.youtube.com/watch?v=b9-jkpYp3Ck)
4. [Операторы](https://www.youtube.com/watch?v=G5D1sKaqZPE)
5. [Преобразование типов](https://www.youtube.com/watch?v=_MESGqUdh_0)
6. [Функции](https://www.youtube.com/watch?v=bd_E7cH5EbA)

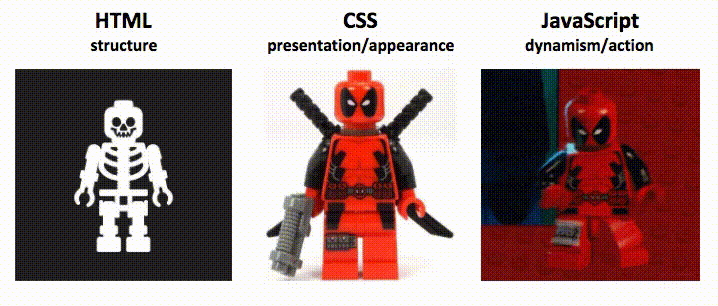
**Что такое JavaScript?**

**JavaScript** – один из самых популярных языков программирования в мире с более чем двадцатилетней историей. Это также один из трёх главных языков программирования для веб-разработчиков:

**HTML**: позволяет добавить контент и структурировать веб-страницу

**CSS**: отвечает за стили и внешний вид

**JavaScript**: задает интерактивное поведение веб-страницы, *«делает веб-страницы живыми»*



<http://internet-step.com/wp-content/uploads/2017/11/5948abfc0e2df5.02876591.gif>

Каждый раз, когда происходит что-то большее, чем просто статичное отображение сайта: обновление контента, или интерактивных карт, или анимация 2D/3D графики, или прокрутка видео в проигрывателе, и т.д. — можете быть уверены, что скорее всего, не обошлось без JavaScript.

JavaScript можно использовать в самых разных вариантах применения - от простых улучшений функциональности сайта до запуска игр и веб-приложений.

Программы на этом языке называются *скриптами*. Они могут встраиваться в HTML и выполняться автоматически при загрузке веб-страницы.

Скрипты распространяются и выполняются, как простой текст. Им не нужна специальная подготовка или компиляция для запуска.

**Движки JavaScript**

Сегодня JavaScript может выполняться не только в браузере, но и на сервере или на любом другом устройстве, которое имеет специальную программу, называющуюся «движком» JavaScript.

У браузера есть собственный движок, который иногда называют «виртуальная машина JavaScript».

Разные движки имеют разные «кодовые имена». Например:

* [V8](https://ru.wikipedia.org/wiki/V8_(%D0%B4%D0%B2%D0%B8%D0%B6%D0%BE%D0%BA_JavaScript)) – в Chrome и Opera.
* [SpiderMonkey](https://ru.wikipedia.org/wiki/SpiderMonkey) – в Firefox.
* …Ещё есть «Trident» и «Chakra» для разных версий IE, «ChakraCore» для Microsoft Edge, «Nitro» и «SquirrelFish» для Safari и т.д.

Эти названия полезно знать, так как они часто используются в статьях для разработчиков. Например, если «функциональность X поддерживается V8», тогда «Х», скорее всего, работает в Chrome и Opera.

**Как работают движки?**

1. Движок читает («парсит») текст скрипта.
2. Затем он преобразует («компилирует») скрипт в машинный язык.
3. После этого машинный код запускается и работает достаточно быстро.

Движок применяет оптимизации на каждом этапе. Он даже просматривает скомпилированный скрипт во время его работы, анализируя проходящие через него данные, и применяет оптимизации к машинному коду, полагаясь на полученные знания. В результате скрипты работают очень быстро.

**Что может и что не может JavaScript?**

Современный JavaScript – это высокоуровневый язык программирования. Он не предоставляет низкоуровневый доступ к памяти или процессору, потому что изначально был создан для браузеров, не требующих этого.

В браузере для JavaScript доступно всё, что связано с манипулированием веб-страницами, взаимодействием с пользователем и веб-сервером.

Например, в браузере JavaScript может:

* Добавлять новый HTML-код на страницу, изменять существующее содержимое, модифицировать стили.
* Реагировать на действия пользователя, щелчки мыши, перемещения указателя, нажатия клавиш.
* Отправлять сетевые запросы на удалённые сервера, скачивать и отправлять файлы.
* Получать и устанавливать куки, задавать вопросы посетителю, показывать сообщения.
* Запоминать данные на стороне клиента («local storage»).

Но возможности JavaScript в браузере ограничены ради безопасности пользователя. Цель заключается в предотвращении доступа недобросовестной веб-страницы к личной информации или нанесения ущерба данным пользователя.

JavaScript на веб-странице не может читать/записывать произвольные файлы на жёстком диске, копировать их или запускать программы. Он не имеет прямого доступа к системным функциям ОС.

Современные браузеры позволяют ему работать с файлами, но с ограниченным доступом, и предоставляют его, только если пользователь выполняет определённые действия, такие как «перетаскивание» файла в окно браузера или его выбор с помощью тега <input>.

Существуют способы взаимодействия с камерой/микрофоном и другими устройствами, но они требуют явного разрешения пользователя. Таким образом, страница с поддержкой JavaScript не может незаметно включить веб-камеру, наблюдать за происходящим и отправлять информацию в ФСБ.

**Что делает JavaScript особенным?**

Как минимум, *три* сильные стороны JavaScript:

* Полная интеграция с HTML/CSS.
* Простые вещи делаются просто.
* Поддерживается всеми основными браузерами и включён по умолчанию.

JavaScript – это единственная браузерная технология, сочетающая в себе все эти три вещи.

Вот что делает JavaScript особенным. Вот почему это **самый распространённый инструмент** для создания интерфейсов в браузере.

Хотя, конечно, JavaScript позволяет делать приложения не только в браузерах, но и на сервере, на мобильных устройствах и т.п.

**Стандарты EcmaScript**

Сначала немного истории. JavaScript создавался как скриптовый язык для Netscape. После чего он был отправлен в ECMA International для стандартизации (ECMA — это ассоциация, деятельность которой посвящена стандартизации информационных и коммуникационных технологий). Это привело к появлению нового языкового стандарта, известного как ECMAScript.

Последующие версии JavaScript уже были основаны на стандарте ECMAScript. Проще говоря, ECMAScript — стандарт, а JavaScript — самая популярная реализация этого стандарта.

**Что такое ES? История версий**

ES — это просто сокращение для ECMAScript. Каждое издание ECMAScript получает аббревиатуру ES с последующим его номером. Всего существует 8 версий ECMAScript. ES1 была выпущена в июне 1997 года, ES2 — в июне 1998 года, ES3 — в декабре 1999 года, а версия ES4 — так и не была принята. Не будем углубляться в эти версии, так как они морально устарели, а рассмотрим только последние четыре.

ES5 был выпущен в декабре 2009 года, спустя 10 лет после выхода третьего издания.

Версия ES6/ES2015 вышла в июне 2015 года. Это также принесло некую путаницу в связи с названием пакета, ведь ES6 и ES2015 — это одно и то же. С выходом этого пакета обновлений комитет принял решение перейти к ежегодным обновлениям. Поэтому издание было переименовано в ES2015, чтобы отражать год релиза. Последующие версии также называются в соответствии с годом их выпуска.

В версии ES6 было внедрено очень много изменений относительно первых версий языка, поэтому в вакансиях очень часто можно встретить требования к знаниям именно этого стандарта.

**Подключение JS к странице**

Практически не отличается от способов подключения уже знакомого нам CSS.

У нас есть три варианта:

1. Писать JavaScript код прямо в странице внутри тега <script> (его можно размещать либо внутри тега <head>, либо внутри тега <body>, лучше всего перед закрытием тега:
2. Подключить через внешний файл:
3. <script src="путь\_к\_файлу\_скрипта/script.js"></script>
4. Использовать строковую запись прямо внутри тега вместе с событием, по наступлению которого этот код должен выполниться (здесь это событие onClick - нажатие на кнопку):
5. <button onClick="alert('Привет!')">Нажми меня</button>

**Переменные**

Переменная – это «именованное хранилище» для данных. Мы можем использовать переменные для хранения названий товаров, имен посетителей и других данных.

Для создания переменной в JavaScript используйте ключевое слово let.

Приведённая ниже инструкция создаёт (другими словами: *объявляет* или *определяет*) переменную с именем «message»:

let message;

Теперь можно поместить в неё данные, используя оператор присваивания =:

let message;

message \*= '\*Привет\*'; // запишем в переменную message слово 'Привет'\*

Строка сохраняется в области памяти, связанной с переменной. Мы можем получить к ней доступ, используя имя переменной:

let message;

message = 'Привет!';

alert(message); // выведет 'Привет!'

Для краткости можно совместить объявление переменной и запись данных в одну строку:

let message = 'Привет!'; // определяем переменную и присваиваем ей значение

alert(message); // Привет!

Мы также можем объявить несколько переменных в одной строке:

let user = 'Алиса', age = 34, message = 'Привет';

Такой способ может показаться короче, но мы не рекомендуем его. Для лучшей читаемости объявляйте каждую переменную на новой строке.

Многострочный вариант немного длиннее, но легче для чтения:

let user = 'Алиса';

let age = 34;

let message = 'Hello';

Некоторые люди также определяют несколько переменных в таком вот многострочном стиле:

let user = 'Алиса',

age = 34,

message = 'Привет';

Все эти варианты работают одинаково. Так что это вопрос личного вкуса и эстетики.

Подобная запись может быть нам знакома из математики. Помните x = 5 или y = sin(x)? Вот здесь x и y - это переменные. В программировании они задаются похожим образом и имеют аналогичный смысл - там, где мы переменную указываем, вместо нее подставляется ее значение.

**var вместо let** В старых скриптах вы также можете найти другое ключевое слово: var вместо let:

\*var\* message = 'Привет!';

Ключевое слово var – *почти* то же самое, что и let. Оно объявляет переменную, но немного по-другому, «устаревшим» способом. Есть еще тонкие различия между let и var, но мы их рассмотрим чуть позже.

Мы легко поймём концепцию «переменной», если представим её в виде «коробки» для данных с уникальным названием на ней.

Например, переменную message можно представить как коробку с названием "message" и значением "Привет!" внутри:

Мы можем положить любое значение в коробку.

Мы также можем изменить его столько раз, сколько захотим:

let message;

message = 'Привет!';

message = 'мир!'; // значение изменено

alert(message);

При изменении значения старые данные удаляются из переменной:

Мы также можем объявить две переменные и скопировать данные из одной в другую.

let hello = 'Привет, мир!';

let message;

\*// копируем значение '\*Привет, мир!\*' из переменной hello в переменную message

message = hello;\*// теперь две переменные содержат одинаковые данные

alert(hello); // Привет, мир!

alert(message); // Привет, мир!

**Повторное объявление вызывает ошибку**

Переменная может быть объявлена только один раз.

Повторное объявление той же переменной является ошибкой:

let message = "Это";

// повторение ключевого слова 'let' приводит к ошибке

let message = "Другое"; // SyntaxError: 'message' has already been declared

Поэтому следует объявлять переменную только один раз и затем использовать её уже без let.

**const**

Объявление const задаёт константу, то есть такое значение, которое нельзя менять:

const apple = 5;

apple = 10; // ошибка

В остальном объявление const полностью аналогично let.

**константы и КОНСТАНТЫ**

Константы, которые жёстко заданы всегда, во время всей программы, обычно пишутся в верхнем регистре. Например: const ORANGE = "#ffa500".

**Типы данных**

Значение в JavaScript всегда относится к данным определённого типа. Например, это может быть строка или число.

Переменная в JavaScript может содержать любые данные. В один момент там может быть строка, а в другой – число:

// Не будет ошибкой

let message = "Привет";

message = 123456;

Языки программирования, в которых такое возможно, называются «динамически типизированными». Это значит, что типы данных есть, но имя переменной не привязано ни к одному из них (а вот значение переменной всегда имеет тип).

* **Число**
* **Строка**
* **Логический тип (булевый)**
* **Специальные "типы"-значения**

**Оператор typeof**

Оператор typeof возвращает тип аргумента. Это полезно, когда мы хотим обрабатывать значения различных типов по-разному или просто хотим сделать проверку.

У него есть две синтаксические формы:

1. Синтаксис оператора: typeof x.
2. Синтаксис функции: typeof(x).

Другими словами, он работает со скобками или без скобок. Результат одинаковый.

Вызов typeof x возвращает строку с именем типа:

alert(typeof(undefined)); // "undefined"

alert(typeof(0)); // "number"

alert(typeof(true)); // "boolean"

alert(typeof("foo")); // "string"

Мы разобрали основные типы в JavaScript. На самом деле их чуть больше, но остальные мы раскроем в следующих уроках.

**Операторы**

Мы можем производить различные операции над переменными. Большинство из них знакомы нам ещё со школы: сложение +, умножение \*, вычитание - и так далее.

Прежде, чем мы двинемся дальше, давайте разберёмся с терминологией.

* *Операнд* – то, к чему применяется оператор. Например, в умножении 5 \* 2 есть два операнда: левый операнд равен 5, а правый операнд равен 2. Иногда их называют «аргументами» вместо «операндов» (зачастую операнды - это переменные).
* *Унарным* называется оператор, который применяется к одному операнду. Например, оператор унарный минус "-" меняет знак числа на противоположный:
* let x = 1;
* x = -x;
* \*\*alert( x ); // -1, применили унарный минус
* *Бинарным* называется оператор, который применяется к двум операндам. Тот же минус существует и в бинарной форме:
* let x = 1, y = 3;
* alert( y - x ); // 2, бинарный минус вычитает значения

Формально, в последних примерах мы говорим о двух разных операторах, использующих один символ: оператор отрицания (унарный оператор, который меняет знак) и оператор вычитания (бинарный оператор, который вычитает одно число из другого).

**Математические**

В JavaScript поддерживаются следующие математические операторы:

* Сложение +
* Вычитание -
* Умножение \*
* Деление /
* Взятие остатка от деления %
* Возведение в степень \*\*

Первые четыре оператора очевидны, а про % и \*\* стоит сказать несколько слов.

* Оператор взятия остатка %, несмотря на обозначение, никакого отношения к процентам не имеет.

Результат a % b – это [остаток](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D1%81_%D0%BE%D1%81%D1%82%D0%B0%D1%82%D0%BA%D0%BE%D0%BC) от целочисленного деления a на b.

Например:

alert( 5 % 2 ); // 1, остаток от деления 5 на 2

alert( 8 % 3 ); // 2, остаток от деления 8 на 3

Чаще всего он применяется в определении четности/нечетности числа (у четного числа остаток от деления на 2 равен 0):

alert( 5 % 2 ); // 1, нечетное число

alert( 8 % 2 ); // 0, четное число

* Возведение в степень

В выражении a \*\* b оператор возведения в степень умножает a на само себя b раз.

Например:

alert( 2 \*\* 2 ); // 4 (2 умножено на себя 2 раза)

alert( 2 \*\* 3 ); // 8 (2 \* 2 \* 2, 3 раза)

alert( 2 \*\* 4 ); // 16 (2 \* 2 \* 2 \* 2, 4 раза)

**Сложение строк при помощи +**

Давайте рассмотрим специальные возможности операторов JavaScript, которые выходят за рамки школьной арифметики.

Обычно при помощи плюса '+' складывают числа.

Но если бинарный оператор '+' применить к строкам, то он их объединяет в одну:

let s = "моя" + "строка";

alert(s); // моястрока

Обратите внимание, если хотя бы один операнд является строкой, то второй будет также преобразован в строку.

Например:

alert( '1' + 2 ); // "12"

alert( 2 + '1' ); // "21"

Как видите, не важно, первый или второй операнд является строкой.

Вот пример посложнее:

alert(2 + 2 + '1'); // будет "41", а не "221"

Здесь операторы работают один за другим. Первый + складывает два числа и возвращает 4, затем следующий + объединяет результат со строкой, производя действие 4 + '1' = 41.

Сложение и преобразование строк — это особенность бинарного плюса +. Другие арифметические операторы работают только с числами и всегда преобразуют операнды в числа.

**Приоритеты операторов**

В том случае, если в выражении есть несколько операторов – порядок их выполнения определяется *приоритетом*, или, другими словами, существует определённый порядок выполнения операторов.

Из школы мы знаем, что умножение в выражении 1 + 2 \* 2 выполнится раньше сложения. Это как раз и есть «приоритет». Говорят, что умножение имеет более высокий приоритет, чем сложение.

Скобки важнее, чем приоритет, так что, если мы не удовлетворены порядком по умолчанию, мы можем использовать их, чтобы изменить приоритет. Например, написать (1 + 2) \* 2.

**Инкремент/декремент**

Одной из наиболее частых числовых операций является увеличение или уменьшение на единицу.

Для этого существуют даже специальные операторы:

* **Инкремент** ++ увеличивает переменную на 1:
* let counter = 2;
* counter++; // работает как counter = counter + 1, просто запись короче
* alert( counter ); // 3
* **Декремент** -- уменьшает переменную на 1:
* let counter = 2;
* counter--; // работает как counter = counter - 1, просто запись короче
* alert( counter ); // 1

**Присваивание**

Оператор простого присваивания, который задает значение переменной (инициализирует его).

let x = 5;

let y = 10;

let z = 25;

x = y; // x равен 10

Цепочка операторов присваивания может быть использована для назначения нескольким переменным одного и того же значения.

x = y = z; // все значения переменных равны 25

**Присваивание со знаком**

В статьях вы можете иногда встретить запись типа x += y. Что это значит?

Такая запись эквивалентна x = x + y, то есть, к старому значению переменной добавляется новое

let str = "Hello "

str += "world"; //результат Hello world т.к. строки поддерживают склеивание

var num = 5;

num += 2 // результат 7

Аналогичные команды можно писать с любыми знаками. Например, x \*= 2 будет эквивалентно x = x \* 2

**Преобразование типов**

Чаще всего операторы и функции автоматически приводят переданные им значения к нужному типу.

Например, alert автоматически преобразует любое значение к строке. Математические операторы преобразуют значения к числам.

Но иногда встречаются случаи, когда нам нужно явно преобразовать значение в ожидаемый тип.

* **Численное преобразование**
* **Строковое преобразование**
* **Логическое преобразование**

**Взаимодействие с пользователем: alert, prompt, confirm**

В этом разделе мы рассмотрим базовые UI операции: alert, prompt и confirm, которые позволяют работать с данными, полученными от пользователя.

* **alert**

<https://www.youtube.com/watch?v=vTM8qrnWKhY>

* **prompt**
* **confirm**

**Особенности встроенных функций**

Конкретное место, где выводится модальное окно с вопросом – обычно это центр браузера, и внешний вид окна выбирает браузер. Разработчик не может на это влиять.

С одной стороны – это недостаток, так как нельзя вывести окно в своём, особо красивом, дизайне.

С другой стороны, преимущество этих функций по сравнению с другими, более сложными методами взаимодействия, которые мы изучим в дальнейшем – как раз в том, что они очень просты.

Это самый простой способ вывести сообщение или получить информацию от посетителя. Поэтому их используют в тех случаях, когда простота важна, а всякие «красивости» особой роли не играют.

**Функции**

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

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

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

С некоторыми встроенными функциями мы уже знакомы - например, alert() и Number().

Для создания своих собственных функций мы сначала должны написать *объявление функции*.

Пример объявления функции:

function showMessage() {

alert( 'Всем привет!' );

}

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

function имя(параметры) {

...тело...

}

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

Например:

function showMessage() {

alert( 'Всем привет!' );

}

\*showMessage();

showMessage();\*

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

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

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

**Параметры функций**

Мы можем передать внутрь функции любую информацию, используя параметры (также называемые *аргументами функции*).

В примере ниже функции передаются два параметра: from и text.

function showMessage(\*from, text\*) { // параметры: from, text

alert(from + ': ' + text);

}

\*showMessage('Аня', 'Привет!'); // Аня: Привет!

showMessage('Аня', "Как дела?"); // Аня: Как дела?\*

Когда функция вызывается в строках 5 и 6, то переданные значения копируются в локальные переменные from и text. Затем они используются внутри (в теле) функции.

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

function showMessage(from, text) {

\*from = '\*' + from + '\*'; // немного украсим "from"\* alert( from + ': ' + text );

}

let from = "Аня";

showMessage(from, "Привет"); // \*Аня\*: Привет

// значение "from" осталось прежним, функция изменила значение локальной переменной

alert( from ); // Аня

**Параметры по умолчанию**

Если параметр не указан, то его значением становится undefined.

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

showMessage("Аня"); // выведет "Аня: undefined"

Это не приведёт к ошибке. Такой вызов выведет "Аня: undefined". В вызове не указан параметр text, поэтому вместо него подставляется undefined.

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

function showMessage(from, \*text = "текст не добавлен"\*) {

alert( from + ": " + text );

}

showMessage("Аня"); // выведет "Аня: текст не добавлен"

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

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

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

Например, явная проверка на undefined:

function showMessage(from, text) {

\*if (text === undefined) {

text = 'текст не добавлен';

}\*

alert( from + ": " + text );

}

**Возврат значения**

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

Простейшим примером может служить функция сложения двух чисел:

function sum(a, b) {

\*return\* a + b;

}

let result = sum(1, 3);

alert( result ); // 3

Директива return может находиться в любом месте тела функции. Как только выполнение доходит до этого места, функция останавливается, и значение возвращается в вызвавший её код (присваивается переменной result выше). Вызовов return внутри одной функции может быть несколько в зависимости от разных условий.

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

**Результат функции с пустым return или без него – undefined**

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

function doNothing() { /\* пусто \*/ }

alert( doNothing() === undefined ); // true

Пустой return аналогичен return undefined:

function doNothing() {

return;

}

alert( doNothing() === undefined ); // true

**Никогда не добавляйте перевод строки между return и его значением**

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

return

(some + long + expression + or + whatever \* f(a) + f(b))

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

return\*;\*(some + long + expression + or + whatever \* f(a) + f(b))

Таким образом, это фактически стало пустым return.

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

return (

some + long + expression

+ or +

whatever \* f(a) + f(b)

)

И тогда всё сработает, как задумано.

**Выбор имени функции**

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

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

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

Функции, начинающиеся с…

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

Примеры таких имён:

showMessage(..) // показывает сообщение

getAge(..) // возвращает возраст (в каком либо значении)

calcSum(..) // вычисляет сумму и возвращает результат

createForm(..) // создаёт форму (и обычно возвращает её)

checkPermission(..) // проверяет доступ, возвращая true/false

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

**Домашнее задание на эту неделю:**

**Вопросы 💎**

1. Какие бывают варианты объявления переменных?
2. Что выведет этот скрипт?
3. Какое значение будет в *z*?
4. let x = 5;
5. let y = 2;
6. let z = x + y;
7. alert(z);
8. Какими тремя способами можно подключить JavaScript код?
9. Есть ли какая-то разница между записями typeof str и typeof(str)?
10. Что вернет код?
11. function showX(x)
12. {
13. return
14. x;
15. }
16. alert(showX(28));
17. Какой результат будет у выражений ниже?
18. "" + 1 + 0
19. "" - 1 + 0
20. true + false
21. 6 / "3"
22. "2" \* "3"
23. 4 + 5 + "px"
24. "$" + 4 + 5
25. "4" - 2
26. "4px" - 2
27. 7 / 0
28. " -9 " + 5
29. " -9 " - 5
30. null + 1
31. undefined + 1
32. " \\t \\n" - 2
33. Ниже приведён код, который запрашивает у пользователя два числа и показывает их сумму.

Он работает неправильно. Код в примере выводит 12 (для значения полей по умолчанию).

В чём ошибка? Исправьте её. Результат должен быть 3.

let a = prompt("Первое число?", 1);

let b = prompt("Второе число?", 2);

alert(a + b); // возвращает 12

1. Что будет в x?
2. x = 5;
3. x++;
4. x -= 2;
5. alert(x);
6. Что выведет этот код?
7. function sum(x, y=0) {
8. return x+y;
9. }
10. alert(sum(5));

**Практическое задание 💎**

1. Создайте страницу, которая по нажатию на кнопку спрашивает имя у пользователя и выводит персонифицированное приветственное сообщение типа "Привет, Маша!".
2. Сделайте простой калькулятор на 4 действия (сложение, вычитание, деление, умножение). 4 кнопки, по нажатию которые запрашиваются два числа и выдается результат их вычисления, соответствующий действию на кнопке.

ОТВЕТЫ :

1. Ключевое слово let + имя переменной, далее используя оператор присваивания = помещаем данные в переменную. Например:

let name = ‘Valerie’

alert(name);

или :

let name = ‘Valerie’, surname = ‘Gagieva’, age = 25

или:

let name = ‘Valerie’;

let surname = ‘Gagieva’;

let age = 25;

или:

let name = ‘Valerie’,

surname = ‘Gagieva’,

age = 25;

1. let name = "Ilya";

alert( `hello ${1}` ); // hello 1

alert( `hello ${"name"}` ); //hello name

alert( `hello ${name}` ); //hello Ilya

1. 7
2. - Внутри тега script html-документа, лучше перед закрывающим тегом body

- через <script src=”script.js”></script>

- через строковую запись прямо внутри тега вместе с событием, по наступлению которого этот код должен выполниться

5. никакой разницы

6. undefined

7. "" + 1 + 0 =10

"" - 1 + 0 = -1

true + false = 1

6 / "3" = 2

"2" \* "3" = 6

4 + 5 + "px" = 9px

"$" + 4 + 5 = $45

"4" - 2 = 2

"4px" - 2 = NaN

7 / 0 = infinity

" -9 " + 5 = -9 5

" -9 " - 5 = -14

null + 1 = 1

undefined + 1 = undefined

" \t \n" – 2 = -2

8. let a = prompt("Первое число?", 1);

let b = prompt("Второе число?", 2);

alert( +a + +b);

9. 4

10. 5