JavaScript. Введение

Что такое JavaScript?

JavaScript — прототипно-ориентированный сценарный язык программирования.
Является диалектом языка ECMAScript

Основные архитектурные черты:

- ✔ динамическая типизация,
- ✔ слабая типизация,
- ✔ автоматическое управление памятью,
- ✔ прототипное программирование,
- ✔ функции как объекты первого класса.

История появления JavaScript

- **1992 год** компания Nombas начала разработку встраиваемого скриптового языка Cmm (Си-минус-минус). Cmm был переименован в ScriptEase
- 1995 год Nombas разработала версию CEnvi, внедряемую в веб-страницы. Страницы, которые можно было изменять с помощью скриптового языка, получили название Espresso Pages
-

История появления JavaScript

- Появление Mocha, который затем был переименован в LiveScript и предназначался как для программирования на стороне клиента, так и для программирования на стороне сервера (там он должен был называться LiveWire). На синтаксис оказали влияние языки Си и Java, и, поскольку Java в то время было модным словом, 4 декабря 1995 года LiveScript переименовали в JavaScript, получив соответствующую

Спецификация ECMAScript

- Спецификация (формальное описание синтаксиса, базовых объектов и алгоритмов) языка Javascript называется ECMAScript.
- JavaScript™ — зарегистрированная торговая марка, принадлежащая корпорации Oracle.
- Название «ECMAScript» было выбрано, чтобы сохранить спецификацию независимой от владельцев торговой марки.
- Современный стандарт — это ECMA-262 5.1 (или просто ES5), поддерживается всеми современными браузерами.

Что умеет JavaScript?

- Современный JavaScript — это «безопасный» язык программирования общего назначения. Он не предоставляет низкоуровневых средств работы с памятью, процессором, так как изначально был ориентирован на браузеры, в которых это не требуется.
- Возможности JavaScript сильно зависят от окружения, в котором он работает. Например, Node.JS поддерживает функции чтения/записи произвольных файлов, выполнения сетевых запросов и т.д.

Что умеет JavaScript?

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

- 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 загружать файлы (технологии AJAXОтправлять сетевые запросы на удалённые сервера, скачивать и загружать файлы (технологии AJAX и COMET).
- Получать и устанавливать cookie, запрашивать данные, выводить сообщения…
- Запоминать данные на стороне клиента («local

Чего НЕ умеет JavaScript?

- **JavaScript — быстрый и мощный язык, но браузер накладывает на его исполнение некоторые ограничения**.
- JavaScript не может читать/записывать произвольные файлы на жесткий диск, копировать их или вызывать программы.
- Он не имеет прямого доступа к операционной системе. Современные браузеры могут работать с файлами, но эта возможность ограничена специально выделенной директорией — *«песочницей»*. Возможности по доступу к устройствам также прорабатываются в современных стандартах и частично доступны в некоторых браузерах.
- JavaScript, работающий в одной вкладке, не может общаться с другими вкладками и окнами, за исключением случая, когда он сам открыл это окно или несколько вкладок из одного источника (одинаковый домен, порт, протокол).
- Из JavaScript можно легко посылать запросы на сервер, с которого пришла страница. Запрос на другой домен тоже возможен, но менее удобен, т.к. и здесь есть ограничения безопасности.

Тенденции развития

- *HTML 5* — эволюция стандарта HTML, добавляющая новые теги и, что более важно, ряд новых возможностей браузерам.
- Вот несколько примеров:
- Чтение/запись файлов на диск (в специальной «песочнице», то есть не любые).
- Встроенная в браузер база данных, которая позволяет хранить данные на компьютере пользователя.
- Многозадачность с одновременным использованием нескольких ядер процессора.
- Проигрывание видео/аудио, без Flash.
- 2d и 3d-рисование с аппаратной поддержкой, как в современных играх.
- Многие возможности HTML5 все еще в разработке, но браузеры постепенно начинают их поддерживать.

Альтернативные браузерные технологии. Java

Java — язык общего назначения, на нем можно писать самые разные программы. Для интернет-страниц есть особая возможность - написание *апплетов*.

Апплет — это программа на языке Java, которую можно подключить к HTML при помощи тега applet:

```
<applet code="BTApplet.class" codebase="/files/tutorial/intro/alt/">
<param name="nodes" value="50,30,70,20,40,60,80,35,65,75,85,90">
<param name="root" value="50">
</applet>
```

Такой тег загружает Java-программу из файла BTApplet.class и выполняет ее с параметрами param.

Конечно, для этого на компьютере должна быть установлена и включена среда выполнения Java. Статистика показывает, что это около 80% компьютеров.

Альтернативные браузерные технологии. Java

Java может делать *все* от имени посетителя, совсем как установленная десктопная программа. В целях безопасности, потенциально опасные действия требуют подписанного апплета и доверия пользователя.

Java требует больше времени для загрузки

Среда выполнения Java должна быть установлена на компьютере посетителя и включена. Таких посетителей в интернет — около 80%.

Java-апплет не интегрирован с HTML-страницей, а выполняется отдельно. Но он может вызывать функции JavaScript.

Подписанный Java-апплет - это возможность делать все, что угодно, на компьютере посетителя, если он вам доверяет. Можно вынести в него все вызовы, которым нужно обойти контекст безопасности, а для самой страницы использовать JavaScript.

Альтернативные браузерные технологии. ActiveX/NPAPI

- ActiveX для IE и NPAPI для остальных браузеров позволяют писать плагины для браузера, в том числе на языке C. Как и в ситуации с Java-апплетом, посетитель поставит их в том случае, если вам доверяет.
- Эти плагины могут как отображать содержимое специального формата (плагин для проигрывания музыки, для показа PDF), так и взаимодействовать со страницей.

Альтернативные браузерные технологии. Adobe Flash

- Adobe Flash — кросс-браузерная платформа для мультимедиа-приложений, анимаций, аудио и видео.
- *Flash-ролик* — это скомпилированная программа, написанная на языке ActionScript. Ее можно подключить к HTML-странице и запустить в прямоугольном контейнере.
- В первую очередь Flash полезен тем, что позволяет **кросс-браузерно** работать с микрофоном, камерой, с буфером обмена, а также поддерживает продвинутые возможности по работе с сетевыми соединениями.

Альтернативные браузерные технологии.
Dart

- Язык Dart предложен компанией Google как замена JavaScript, у которого, по выражению создателей Dart, есть фатальные недостатки.
- Сейчас этот язык, хотя и доступен, находится в стадии разработки и тестирования. Многие из возможностей еще ожидают своей реализации, есть ряд проблем. Другие ведущие интернет-компании объявляли о своей незаинтересованности в Dart. Но в будущем он может составить конкуренцию JS.

Тег SCRIPT. <script> ... </script>

- Программы на языке JavaScript можно вставить в любое место HTML при помощи тега SCRIPT.

```
<!DOCTYPE HTML>
<html>
<head>
<!-- Тег meta для указания кодировки -->  <meta charset="utf-8">
</head>
<body>
<p>Начало документа...</p>
<script>    alert('Привет, Мир!');  </script>
<p>...Конец документа</p>
</body>
</html>
```

Тег SCRIPT. <script> ... </script>

- Тег script содержит исполняемый код. Предыдущие стандарты HTML требовали обязательного указания атрибута type, но сейчас он уже не нужен. Достаточно просто <script>.Браузер, когда видит <script>:
- Начинает отображать страницу, показывает часть документа до script
- Встретив тег script, переключается в JavaScript-режим и не показывает, а исполняет его содержимое.
- Закончив выполнение, возвращается обратно в HTML-режим и отображает оставшуюся часть документа.
- **Пока браузер не выполнит скрипт - он не может отобразить часть страницы после него**.
- alert(...)Отображает окно с сообщением и ждет, пока посетитель не нажмет «Ок»

Внешние скрипты

- **Если JavaScript-кода много — его выносят в отдельный файл, который подключается в HTML:**
- <script src="/path/to/script.js"></script>
- Здесь /path/to/script.js - это абсолютный путь к файлу, содержащему скрипт (из корня сайта).
- Браузер сам скачает скрипт и выполнит.

Как правило, в HTML пишут только самые простые скрипты, а сложные выносят в отдельный файл.

Благодаря этому один и тот же скрипт, например, меню или библиотека функций, может использоваться на разных страницах.

Браузер скачает его только первый раз и в дальнейшем, при правильной настройке сервера, будет брать из своего кэша.

Внешние скрипты

- Если указан атрибут src, то содержимое тега игнорируется.
- В одном теге SCRIPT нельзя одновременно подключить внешний скрипт и указать код.
- Вот так не работает:
- <script **src**="file.js">
- alert(1); // если указан src, то внутренняя часть скрипта игнорируется
- </script>

Команды

- Как правило, новая команда занимает отдельную строку — так код лучше читается.

```
alert('Привет');
alert('Мир');
```

- Точку с запятой во многих случаях можно не ставить, если есть переход на новую строку. Так тоже будет работать. В этом случае JavaScript интерпретирует переход на новую строчку как разделитель команд и автоматически вставляет «виртуальную» точку с запятой между ними.
- Однако, внутренние правила по вставке точки с запятой не идеальны. В некоторых ситуациях JavaScript «забывает» вставить точку с запятой там, где она нужна. Таких ситуаций не так много, но они все же есть, и ошибки, которые при этом появляются, достаточно

Комментарии

- // Однострочный комментарий
  alert('Привет'); alert('Мир');
- /* Это - многострочный комментарий.*/
- alert('Привет');alert('Мир');
- Вложенные комментарии не поддерживаются!

Строгий режим — "use strict"

- В 2009 года, когда появился ECMAScript 5 (ES5). Он добавил новые возможности в язык и изменил некоторые из существующих. Чтобы старый код работал, большинство таких модификаций по умолчанию отключены. Вы должны явно включить их с помощью специальной директивы: "use strict".
- Когда она находится в начале скрипта, весь сценарий работает в «современном» режиме.
- Например:
- "use strict";
- // этот код работает в современном режиме

Переменные до ES5

Для *объявления* или, другими словами, *создания переменной* используется ключевое слово var:

var message;

После объявления, можно записать в переменную данные:

var message;

message = 'Привет'; // сохраним в переменной строку

- var message = 'Привет';

Директива var

В JavaScript вы можете создать переменную и без var, достаточно просто присвоить ей значение:

x = "value"; // переменная создана, если ее не было

Технически, это не вызовет ошибки, но делать так все-таки не стоит.

IE<9 такую переменную использовать нельзя.

Переменные с ES5

- Для создания переменной в JavaScript используйте ключевое слово let

```
let user = 'John';
let age = 25;
let message = 'Hello';
```

или

```
let user = 'John', age = 25, message = 'Hello';
```

Имена переменных

- На имя переменной в JavaScript наложены всего два ограничения.
- Имя может состоять из: букв, цифр, символов $ и _
- Первый символ не должен быть цифрой.
- Например:
- let myName;
- let test123;
- доллар '$' и знак подчеркивания '_' являются такими же обычными символами, как буквы:
- let $ = 5; // объявили переменную с именем '$'
- let _ = 15; // переменная с именем '_'
- alert($);

Имена переменных

- let 1a; // начало не может быть цифрой
- let my-name; // дефис '-' не является разрешенным символом
- **Регистр букв имеет значение.**
  Переменные apple и AppLE - две разные переменные.
- **Русские буквы допустимы, но не рекомендуются**

  Можно использовать и русские буквы:

  let имя = "Вася";

  alert(имя);

  Технически, ошибки здесь нет, но на практике сложилась традиция использовать в именах только английские буквы.

Зарезервированные имена

- Существует список зарезервированных слов, которые нельзя использовать при именовании переменных, так как они используются самим языком, например: **var, class, return, implements** и др.
- Некоторые слова, например, **class**, не используются в современном JavaScript, но они заняты на будущее. Некоторые браузеры позволяют их использовать, но это может привести к ошибкам.
- let class = 5;alert(class);

Правильный выбор имени

- **Никакого транслита**. Только английский.
- Использовать короткие имена только для переменных «местного значения».
- **Называть переменные именами**, не несущими смысловой нагрузки, например a, e, p, mg - можно только в том случае, если они используются в небольшом фрагменте кода и их применение очевидно.
- **Использование CamelCase**
- var borderLeftWidth;
- Этот способ записи называется «верблюжьей нотацией» или, по-английски, «camelCase».

Константы

- *Константа* — это переменная, которая никогда не меняется. Как правило, их называют большими буквами, через подчёркивание. Например:

```
const colorRed = "#F00";
const colorGreen = "#0F0";
const COLOR_BLUE = "#00F";
const COLOR_ORANGE = "#FF7F00";
alert(colorRed); // #F00
```

Технически, константа является обычной переменной, то есть её можно изменить.

Типы данных. Число number

```
let  n = 123;
n = 12.345;
```

- Единый тип *число* используется как для целых, так и для дробных чисел. Существуют специальные числовые значения **Infinity (бесконечность)** и **NaN (ошибка вычислений).** Они также принадлежат типу «число».
- Например, бесконечность **Infinity** получается при делении на ноль:
- `alert( 1 / 0 ); // Infinity`
- Ошибка вычислений **NaN** будет результатом некорректной математической операции, например:
- `alert( "нечисло" * 2 ); // NaN, ошибка`

Строка string:

- let str = «Привет»;
- str = ’Мир’;
- **В JavaScript одинарные и двойные кавычки равноправны.**
- Можно использовать или те или другие.
- **Тип *символ* не существует, есть только *строка***
- В некоторых языках программирования есть специальный тип данных для одного символа. Например, в языке С это char. В JavaScript есть только тип «строка» string.

Булевый (логический) тип boolean

- У него всего два значения - true (истина) и false (ложь).Как правило, такой тип используется для хранения значения типа да/нет, например:
- let checked = true; // поле формы помечено галочкой
- checked = false; // поле формы не содержит галочки

null

- **null** — специальное значение. Оно имеет смысл «ничего». Значение null не относится ни к одному из типов выше, а образует свой отдельный тип, состоящий из единственного значения null:
- var age = null;
- В JavaScript null не является «ссылкой на несуществующий объект» или «нулевым указателем», как в некоторых других языках. Это просто специальное значение, которое имеет смысл «ничего» или «значение неизвестно».В частности, код выше говорит о том, что возраст age неизвестен.

undefined

- Специальное значение, которое, как и null, образует свой собственный тип. Оно имеет смысл «значение не присвоено».Если переменная объявлена, но в неё ничего не записано, то ее значение как раз и есть undefined:
- let u;
- alert(u); // выведет "undefined"
- Можно присвоить undefined и в явном виде, хотя это делается редко:
- let x = 123;
- x = undefined;
- В явном виде undefined обычно не присваивают, так как это противоречит его смыслу. Для записи в переменную «пустого значения» используется null.

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 два синтаксиса:
- Синтаксис оператора: typeof x.
- Синтаксис функции: typeof(x).

- typeof 0 // "number"
- typeof true // "boolean"

Объекты object

- Первые 5 типов называют *«примитивными»*.
- Особняком стоит шестой тип: *«объекты»*. К нему относятся, например, даты, он используется для коллекций данных и для многого другого

Сложение строк, бинарный +

Если бинарный оператор + применить к строкам, то он их объединяет в одну:

```
let  a = "моя" + "строка";
alert(a); // моя строка
```

Если хотя бы один аргумент является строкой, то второй будет также преобразован к строке!

Причем не важно, справа или слева находится операнд-строка, в любом случае нестроковый аргумент будет преобразован. Например:

```
alert( '1' + 2 ); // "12"
alert( 2 + '1' ); // "21"
```

Инкремент/декремент: ++, --

- Одной из наиболее частых операций в JavaScript, как и во многих других языках программирования, является увеличение или уменьшение переменной на единицу.
- Для этого существуют даже специальные операторы:
- **Инкремент** ++ увеличивает на 1:
- let i = 2;
- i++; // более короткая запись для i = i + 1.
- alert(i); // 3
- **Декремент** -- уменьшает на 1:
- let i = 2;
- i--; // более короткая запись для i = i - 1.
- alert(i); // 1
- Инкремент/декремент можно применить только к переменной. **Код 5++ даст ошибку.**

Оператор запятая

- Запятая тоже является оператором. Ее можно вызвать явным образом, например:
- a = (5, 6);
- alert(a);
- Запятая позволяет перечислять выражения, разделяя их запятой ','. Каждое из них — вычисляется и отбрасывается, за исключением последнего, которое возвращается.
- Запятая — единственный оператор, приоритет которого ниже присваивания. В выражении a = (5,6)для явного задания приоритета использованы скобки, иначе оператор '=' выполнился бы до запятой',', получилось бы (a=5), 6.
- Зачем же нужен такой странный оператор, который отбрасывает значения всех перечисленных выражений, кроме последнего?
- Обычно он используется в составе более сложных конструкций, чтобы сделать несколько действий в одной строке. Например:
- // три операции в одной строке
- for (**a = 1, b = 3, c = a*b**; a < 10; a++) {
- ...
- }

Взаимодействие с пользователем: alert

- Синтаксис:
- alert(сообщение);
- alert выводит на экран окно с сообщением и приостанавливает выполнение скрипта, пока пользователь не нажмет «ОК».
- Окно сообщения, которое выводится, является *модальным окном*. Слово «модальное» означает, что посетитель не может взаимодействовать со страницей, нажимать другие кнопки и т.п., пока не разберется с окном. В данном случае - пока не нажмет на «ОК».

Взаимодействие с пользователем: prompt

Функция prompt принимает два аргумента:

result = prompt(title, default);

Она выводит модальное окно с заголовком title, полем для ввода текста, заполненным строкой по умолчанию default и кнопками OK/CANCEL.

Пользователь должен либо что-то ввести и нажать OK, либо отменить ввод кликом на CANCEL или нажатием ESC на клавиатуре.

Вызов prompt возвращает то, что ввел посетитель - строку или специальное значение null, если ввод отменен.

Как и в случае с alert, окно prompt модальное.

```
let  years = prompt('Сколько вам лет?', 100);
 alert('Вам ' + years + ' лет!')
```

Взаимодействие с пользователем: confirm

Синтаксис:

result = confirm(question);

confirm выводит окно с вопросом question с двумя кнопками: OK и CANCEL.

Результатом будет true при нажатии OK и false - при CANCEL(Esc).

```
let  isAdmin = confirm("Вы - администратор?");
alert(isAdmin);
```

Особенности встроенных функций

Место, где выводится модальное окно с вопросом, и внешний вид окна выбирает браузер. Разработчик не может на это влиять.

С одной стороны — это недостаток, т.к. нельзя вывести окно в своем дизайне.

С другой стороны, преимущество этих функций по сравнению с другими, более сложными методами взаимодействия— 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.

Условные операторы: if, '?'

- Оператор if («если») получает условие, в примере ниже это year != 2011. Он вычисляет его, и если результат — true, то выполняет команду.
- Если нужно выполнить более одной команды — они оформляются блоком кода в фигурных скобках:
- **if (year != 2011) {**
- **alert('А вот..');**
- **alert('..и неправильно!');**
- **}**
- **Рекомендуется использовать фигурные скобки всегда, даже когда команда одна.** Это улучшает читаемость кода.

Преобразование к логическому типу

Оператор if (...) вычисляет и преобразует выражение в скобках к логическому типу.

В логическом контексте число 0, пустая строка "", null и undefined, а также NaN являются false, остальные значения — true.

Например, такое условие никогда не выполнится:

```
if (0) { // 0 преобразуется к false
  ...
}
```

… А такое — выполнится всегда:

```
if (1) { // 1 преобразуется к true
  ...
}
```

Вычисление условия в проверке if (year != 2011) может быть вынесено в отдельную переменную:

```
let  cond = (year != 2011);  // true/false
if (cond) {
 ...
}
```

Неверное условие, else

```
let year = prompt(Какой сейчас год, '');
if (year == 2015)
{  alert('Да вы знаток!');}
else
{  alert('А вот и неправильно!'); // любое
   значение, кроме 2015}
```

Несколько условий, else if

В случае, если нужно проверить несколько вариантов условия. Для этого используется блок else if .... Например:

```
let  year = prompt('В каком году появилась спецификация ECMA-262 5.1?', '');
if (year < 2011) {
alert('Это слишком рано..');
}
 else if (year > 2011) {
alert('Это поздновато..');
}
else {
 alert('Да, точно в этом году!');
}
```

В примере выше JavaScript сначала проверит первое условие, если оно ложно — перейдет ко второму — и так далее, до последнего else.

Оператор вопросительный знак '?'

```
let access;
let age = prompt('Сколько вам лет?', '');
if (age > 14) {  access = true;}
else {  access = false;}
alert(access);
```

Оператор вопросительный знак '?' позволяет делать это короче и проще.

Он состоит из трех частей:

условие ? значение1 : значение2

Проверяется условие, затем если оно верно — возвращается значение1 , если неверно — значение2, например:

```
access = (age > 14) ? true : false;
```

Оператор '?' выполняется позже большинства других, в частности — позже сравнений, поэтому скобки можно не ставить:

```
access = age > 14 ? true : false;
```

Задание

- Перепишите if с использованием условного оператора '?':

```
let result;
if (a + b < 4)
{ result = 'Мало'; }
else { result = 'Много'; }
```

Ответ

- result = (a + b < 4) ? 'Мало' : 'Много';

Задание

- Перепишите if..else с использованием нескольких операторов '?'.
- Для читаемости рекомендуется разбить код на несколько строк.

```
let message;
 if (login == 'Сотрудник')
{ message = 'Привет'; }
else if (login == 'Директор')
{ message = 'Здравствуйте'; }
else if (login == '')
{ message = 'Нет логина'; }
else { message = ''; }
```

Ответ

```
let message = (login == 'Сотрудник') ? 'Привет'
    :
(login == 'Директор') ? 'Здравствуйте' :
(login == '') ? 'Нет логина' : '';
```

Конструкция "switch"

```
switch(x) {
case 'value1': // if (x === 'value1')
...
[break]
case 'value2':
// if (x === 'value2')
...
 [break]
default:
...
[break]
}
```

```
let a = 2 + 2;
switch (a) {
    case 3: alert( 'Маловато' ); break;
    case 4: alert( 'В точку!' ); break;
    case 5: alert( 'Перебор' ); break;
    default: alert( "Нет таких значений" );
 }
```

Логические операторы

- **|| (ИЛИ)**
- Оператор ИЛИ выглядит как двойной символ вертикальной черты:
- result = a || b;
- **Логическое ИЛИ в классическом программировании работает следующим образом: «если *хотя бы один* из аргументов true, то возвращает true, иначе — false».**
- Получается следующая таблица результатов:
- **alert(true || true); // true**
- **alert(false || true); // true**
- **alert(true || false); // true**
- **alert(false || false); // false**
- При вычислении ИЛИ в JavaScript можно использовать любые значения. В этом случае они будут интерпретироваться как логические.
- Например, число 1 будет воспринято как true, а 0 — как false:
- **if (1 || 0) { // сработает как if(true || false)**
- **alert('верно');**
- **}**

Короткий цикл вычислений

JavaScript вычисляет несколько ИЛИ слева направо. При этом, чтобы экономить ресурсы, используется так называемый *«короткий цикл вычисления»*.

Допустим, вычисляются несколько ИЛИ подряд: a || b || c || .... Если первый аргумент — true, то результат заведомо будет true (хотя бы одно из значений — true), и остальные значения игнорируются.

Это особенно заметно, когда выражение, переданное в качестве второго аргумента, имеет *сторонний эффект* — например, присваивает переменную.

При запуске примера ниже присвоение x не произойдёт:

```
let  x;
true || (x = 1);  // просто вычислим ИЛИ, без if
alert(x); // undefined, x не присвоен
```

…А в примере ниже первый аргумент — false, так что ИЛИ попытается вычислить второй, запустив тем самым присваивание:

```
let  x;
false || (x = 1);
alert(x); // 1
```

&& (И)

- Оператор И пишется как два амперсанда &&:
- result = a && b;
- **В классическом программировании И возвращает true, если оба аргумента истинны, а иначе — false**
- **alert(true && true); // true**
- **alert(false && true); // false**
- **alert(true && false); // false**
- **alert(false && false); // false**
- Пример:
- let **hour = 12, minute = 30;**
- **if (hour == 12 && minute == 30) {**
- **alert('Время 12:30');**
- **}**
- Как и в ИЛИ, допустимы любые значения:
- **if (1 && 0) { // вычислится как true && false**
- **alert('не сработает, т.к. условие ложно');**
- }

Приоритет оператора И && больше, чем ИЛИ ||, т.е. он выполняется раньше.

Поэтому в следующем коде сначала будет вычислено правое И: 1 && 0 = 0, а уже потом — ИЛИ.

```
alert(5 || 1 && 0); // 5
```

! (НЕ)

- Оператор НЕ — самый простой. Он получает один аргумент. Синтаксис:
- var result = !value;
- Действия !:
- Сначала приводит аргумент к логическому типу true/false.
- Затем возвращает противоположное значение.
- Например:
- **alert(!true) // false**
- **alert(!0) // true**
- **В частности, двойное НЕ используются для преобразования значений к логическому типу:**
- **alert(!!"строка") // true**
- **alert(!!null) // false**

Циклы while

- Цикл while имеет следующий синтаксис:

```
while (condition) {
// код
// также называемый "телом цикла"
}

let i = 0;
while (i < 3) { // выводит 0, затем 1, затем 2 alert( i );
i++;
}
```

Циклы do while

- Проверку условия можно разместить под телом цикла, используя специальный синтаксис do..while:

```
do {
// тело цикла
} while (condition);

let i = 0;
do {
alert( i );
 i++;
} while (i < 3);
```

Циклы for

- Более сложный, но при этом самый распространённый цикл — цикл for.

```
for (начало; условие; шаг) {
// ... тело цикла ...
 }
for (let i = 0; i < 3; i++) {
// выведет 0, затем 1, затем 2
alert(i);
}
```

Прерывание цикла: «break»

```
let sum = 0;
while (true) {
    let value = +prompt("Введите число", '');
    if (!value) break;
    sum += value;
 }
alert( 'Сумма: ' + sum );
```

Переход к следующей итерации: continue

```
for (let i = 0; i < 10; i++) {
    // если true, пропустить оставшуюся часть
       тела цикла
    if (i % 2 == 0) continue;
    alert(i); // 1, затем 3, 5, 7, 9
 }
```

Задание

- Напишите цикл, который предлагает prompt ввести число, большее 100. Если посетитель ввёл другое число – попросить ввести ещё раз, и так далее.
- Цикл должен спрашивать число пока либо посетитель не введёт число, большее 100, либо не нажмёт кнопку Отмена (ESC).
- Предполагается, что посетитель вводит только числа. Предусматривать обработку нечисловых строк в этой задаче необязательно.

Функции

1. Функции объявления

```
function имя(параметры) {
...тело...
 }

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

Локальные переменные

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

```
function showMessage() {
    let message = "Привет, я JavaScript!";
    // локальная переменная
    alert( message );
}
showMessage(); // Привет, я JavaScript!
alert( message ); // <-- будет ошибка, т.к. переменная видна только внутри
```

Внешние переменные

У функции есть доступ к внешним переменным, например:

```
let userName = 'Вася';
function showMessage() {
    let message = 'Привет, ' + userName;
    alert(message);
 }
showMessage(); // Привет, Вася
```

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

```
let userName = 'Вася';
function showMessage() {
    userName = "Петя”;// (1) изменяем значение внешней переменной
    let message = 'Привет, ' + userName;
    alert(message);
}
alert( userName ); // Вася перед вызовом функции
showMessage();
alert( userName ); // Петя, значение внешней переменной было изменено функцией
```

Глобальные переменные

- Переменные, объявленные снаружи всех функций, такие как внешняя переменная userName в вышеприведённом коде – называются *глобальными*.
- *Глобальные переменные* видимы для любой функции (если только их не перекрывают одноимённые локальные переменные).
- Желательно сводить использование глобальных переменных к минимуму. В современном коде обычно мало или совсем нет глобальных переменных. Хотя они иногда полезны для хранения важнейших

1. Функции объявления