

Цели занятия

- познакомиться с понятием алгоритма и его свойствами;
- узнать что такое JavaScript и что он умеет делать;
- научиться подключать JavaScript к странице сайта;

Полезные материалы

- [Современный учебник JavaScript](#)
- [Книга “Грокаем алгоритмы”](#)
- [Урок по блок-схемам: что это и как их создавать?](#)

Алгоритм — это...

Алгоритмы окружают нас повсюду, просто за пределами информатики мы называем их иначе. Например, один из самых ярких примеров алгоритма это приготовление блюда по рецепту, где сам рецепт и является алгоритмом.

По своей сути алгоритм — это набор инструкций, который описывает порядок действий для достижения определенного результата. Практически любую задачу можно оформить в виде алгоритма, чтобы другие люди могли повторить её решение.

Алгоритмы в информатике отличаются тем, что стараются решить задачу наиболее оптимальным и эффективным способом. Для огромного количества задач уже придумано самое оптимальное решение (или даже несколько), которые полезно знать и применять в своей работе, более подробно об этом можно прочесть [в книге “Грокаем алгоритмы”](#)

Виды и типы алгоритмов

1. Линейный алгоритм — это последовательное выполнение инструкций в строгой очередности их расположения.

Например, представим что нам нужно написать простейший алгоритм создания бутерброда с сыром:

- взять батон хлеба;
- отрезать кусок хлеба;
- взять сыр;
- отрезать кусок сыра;
- положить сыр на хлеб
- ...

2. Ветвление — последовательность действий может изменяться в зависимости от заданных условий.

Пример: Если идет сильный дождь, тогда возьми зонт, а иначе брать зонт не нужно.

В большинстве случаев слово «иначе» опускается, так как из контекста первой части фразы уже понятна дальнейшая логика.

3. Циклические — это последовательность действий, которую необходимо повторять несколько раз для достижения результата.

Пример: В одном ящике лежат груши, необходимо отобрать гнилые и хорошие. Для этого совершаем следующие действия:

- взять из ящика грушу;
- посмотреть, гнилая она или нет;
- если гнилая, то выбросить;
- если нет, положить в другой ящик;
- повторить операцию до перебора всех груш в ящике.

В большинстве случаев стоит избегать закливания программы, т.е. такого поведения, когда цикл выполняется бесконечно и программа не может достигнуть результата.

Способы записи алгоритма

Словесный способ – способ записи алгоритма на естественном языке. Данный способ очень удобен, если нужно приближенно описать суть алгоритма. Однако при словесном описании не всегда удастся ясно и точно выразить логику действий.

Например, алгоритм расчёта площади прямоугольника:

- Начало алгоритма.
- Задать численное значение стороны a .
- Задать численное значение стороны b .
- Умножить a на b .
- Вывести результат вычислений.
- Конец алгоритма.

Графический способ — позволяет более наглядно выразить суть алгоритма, т.к. каждое действие представлено в виде определенной фигуры. Существует несколько видов графического способа, но самый популярный это блок-схемы. В рамках данного курса это не будет рассматриваться, но можно самостоятельно углубиться в тему, начав с [этой статьи](#).

Программный способ – это запись алгоритма на языке программирования, позволяющем на основе строго определенных правил формировать последовательность предписаний, однозначно отражающих смысл и содержание алгоритма, с целью его последующего исполнения на компьютере. Именно этот способ и будет рассмотрен в курсе с использованием JavaScript.

Введение в JavaScript

Изначально JavaScript был создан, чтобы «сделать веб-страницы живыми».

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. Это отличает JavaScript от другого языка – [Java](#).

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](#) – в Chrome, Opera и Edge.
- [SpiderMonkey](#) – в Firefox.
- ...Ещё есть «Chakra» для IE, «JavaScriptCore», «Nitro» и «SquirrelFish» для Safari и т.д.

Что может JavaScript в браузере?

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

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.

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 загружать файлы (технологии [AJAX](#) и [COMET](#)).
- Получать и устанавливать куки, задавать вопросы посетителю, показывать сообщения.
- Запоминать данные на стороне клиента («local storage»).

Подключение JavaScript к странице

Для того чтобы подключать скрипты к страницам сайта был создан отдельный парный тег `<script>`, который можно размещать как внутри `<head>`, так и внутри `<body>`.

Причём содержимое скрипта можно написать прямо внутри данного тега, например, вот так:

```
<script>
    alert('Hello, Frontenders!');
</script>
```

Но в большинстве случаев скрипты всё-таки пишут в отдельных файлах (по аналогии со стилями) с расширением `.js` и подключают следующим образом:

```
<script src="./script.js"></script>
```

Строгий режим “use strict”

На протяжении долгого времени JavaScript развивался без проблем с обратной совместимостью. Новые функции

добавлялись в язык, в то время как старая функциональность не менялась.

Преимуществом данного подхода было то, что существующий код продолжал работать. А недостатком – что любая ошибка или несовершенное решение, принятое создателями JavaScript, застревали в языке навсегда.

Так было до 2009 года, когда появился ECMAScript 5 (ES5). Он добавил новые возможности в язык и изменил некоторые из существующих. Чтобы устаревший код работал, как и раньше, по умолчанию подобные изменения не применяются. Поэтому нам нужно явно их активировать с помощью специальной директивы: `"use strict"`.

```
"use strict";

// этот код работает в современном режиме
alert("Hello, Frontenders!");
```

Переменные

Большинству JavaScript-приложений необходимо работать с информацией. Например, если вы создаёте интернет-магазин, то вам нужно хранить информацию о корзине пользователя, а если чат, то информацию о сообщениях. Именно для этого и используются переменные.

Переменная – это «именованное хранилище» для данных. Мы можем использовать переменные для хранения товаров, 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 = "Hello, Frontenders!"; // сохранить строку  
'Hello' в переменной с именем message
```

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

```
let message;  
message = "Hello, Frontenders!";  
  
alert(message); // показывает содержимое переменной
```

Для краткости обычно совмещают объявление переменной и запись данных в одну строку:

```
let message = "Hello, Frontenders!"; // определяем  
переменную и присваиваем ей значение  
  
alert(message);
```

Типы данных

Всего в JavaScript существует 8 типов данных. В этом занятии мы рассмотрим их в общем, а в следующих поговорим подробнее о каждом.

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

```
// Не будет ошибкой  
  
let message = "hello";  
  
message = 123456;
```

Языки программирования, в которых такое возможно, называются «динамически типизированными». Это значит, что типы данных есть, но переменные не привязаны ни к одному из них.

Число

Числовой тип данных (number) представляет как целочисленные значения, так и числа с плавающей точкой (дробные).

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

BigInt

В JavaScript тип «number» не может содержать числа больше, чем $(2^{53}-1)$ (т. е. 9007199254740991), или меньше, чем $-(2^{53}-1)$ для отрицательных чисел. Это техническое ограничение вызвано их внутренним представлением.

Для большинства случаев этого достаточно. Но иногда нам нужны действительно гигантские числа, например, в криптографии или при использовании метки времени («timestamp») с микросекундами.

```
// символ "n" в конце означает, что это BigInt  
  
const bigInt = 1234567890123456789012345678901234567890n;
```

Строки

Строка (string) в JavaScript должна быть заключена в кавычки.

В JavaScript существует три типа кавычек:

- Двойные кавычки: "Привет".
- Одинарные кавычки: 'Привет'.
- Обратные кавычки: `Привет`

Булевый (логический) тип

Булевый тип (boolean) может принимать только два значения: true (истина) и false (ложь).

```
let nameFieldChecked = true; // да, поле отмечено
let ageFieldChecked = false; // нет, поле не
отмечено
```

null

Специальное значение null не относится ни к одному из типов, описанных выше.

Оно формирует отдельный тип, который содержит только значение null. Это просто специальное значение, которое представляет собой «ничего», «пусто» или «значение неизвестно».

```
let age = null;
```

undefined

Специальное значение undefined также стоит особняком. Оно формирует тип из самого себя так же, как и null.

Оно означает, что «значение не было присвоено».

Если переменная объявлена, но ей не присвоено никакого значения, то её значением будет `undefined`:

```
let age;
```

Object

Тип `object` (объект) – особенный. Все предыдущие типы называются «примитивными», потому что их значениями могут быть только простые значения (будь то строка, или число, или что-то ещё). В объектах же хранят коллекции данных или более сложные структуры.

```
let user = { firstName: "Yury", lastName: "Kundin" };
```

Тип `symbol` (символ)

Используется для создания уникальных идентификаторов в объектах. Упоминается здесь о нём для полноты картины, изучим этот тип после объектов. В реальной работе используется очень редко.