Яндекс

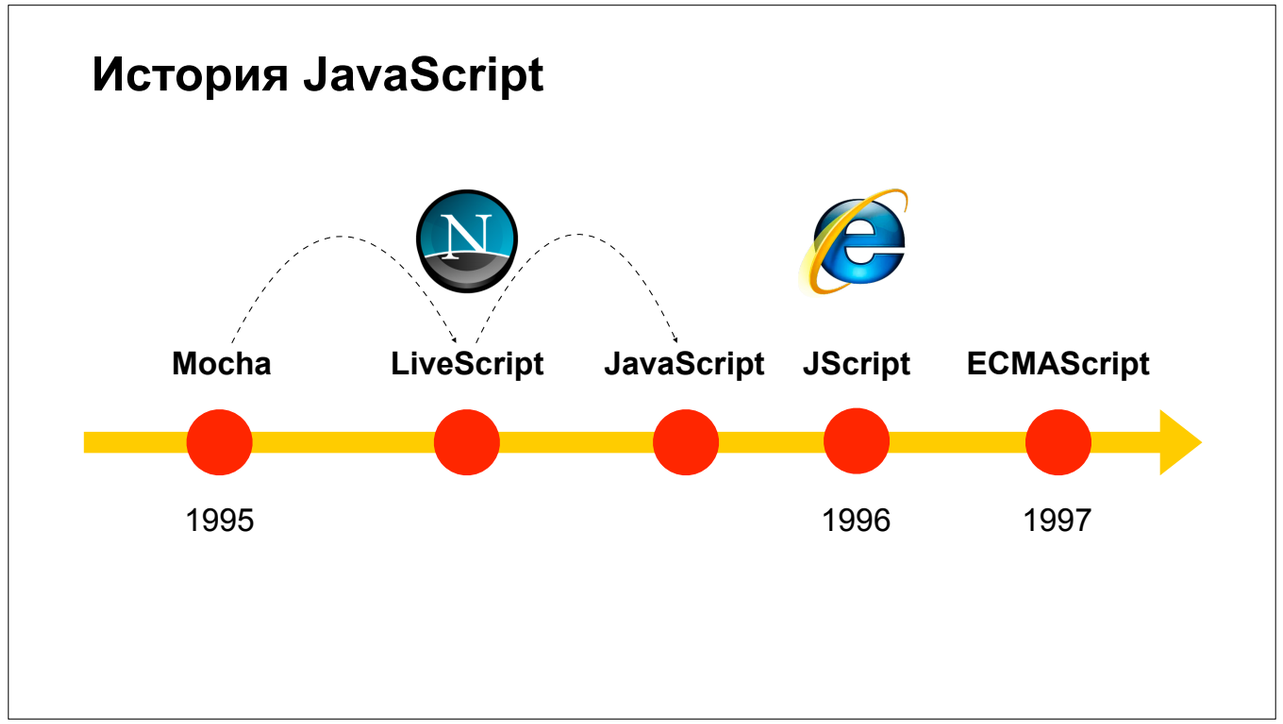
Введение в JavaScript

Евгений Марков, разработчик интерфейсов

Сегодня

- История JavaScript
- Характеристики языка
- Область применения и среды исполнения
- Базовый синтаксис (переменные, ветвление, etc)
- Инструменты разработки
- Как сдавать домашние задания

На каких языках вы уже программировали?



История языка началась в 1995 году в компании Netscape Communications. Web в то время представлял собой набор документов написанных на HTML и связанных между собой гиперссылками. Иногда на эти страницы добавлялись стили написанные на CSS. Компания Netscape разрабатывала в то время Web-браузер и основатель считал, что Web должен быть более динамичным. Он хотел, чтобы странички, которые видят пользователи, стали более интерактивными, чтобы появились анимации. Для этого в браузер нужно было встроить какой-то язык программирования. Важным критерием для языка была простота изучения. Он должен был предназначаться не для профессиональных разработчиков, а для дизайнеров. Из этого предположения родилась идея языка Mocha (Мокко).

В это же время компания Sun развивала язык Java, который в то время назывался Oak. В Netscape детально изучили возможность встроить в браузеры Java, но столкнулись с тем, что язык «навороченный», сложный и предназначен не для аудитории любителей, скриптеров, дизайнеров. Инженеры даже попытались написать свою реализацию виртуальной машины Java, но остановили свои исследования так как не смогли достичь совместимости со спецификацией.

Сроки были сжатые, компания пыталась получить конкурентное преимущество на рынке и Брендану далее пришлось выбирать: Python, Tcl, Scheme или создавать язык с нуля. Он выбрал последний вариант и за несколько недель написал первый вариант языка, который был впоследствии интегрирован в Netscape Navigator 2.0. Практически тут же он был переименован из Mocha в LiveScript.

Язык вышел в небольшой степени похожим на Java и многое заимствовал из Scheme. Sun и Netscape заключили сделку, в ходе которой договорились на маркетинговый ход – LiveScript переименовали в JavaScript и стали позиционировать как простой язык для выполнения

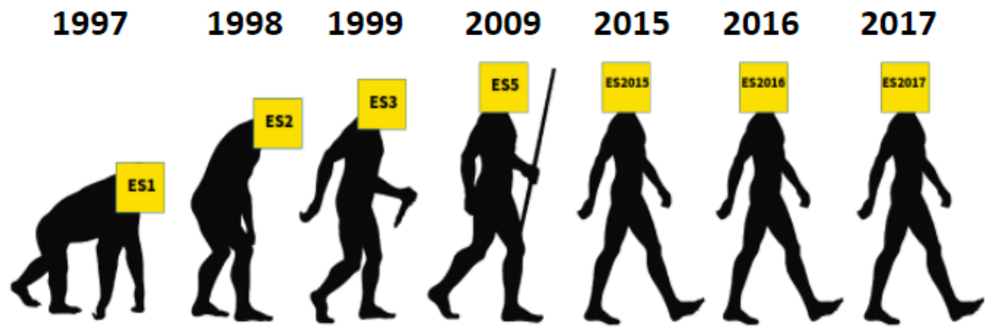
клиентских задач, а Java, в свою очередь, позиционировалась как «профессиональный» язык программирования.

Таким образом, если бы не Брендан, мы могли бы писать сейчас фронтенд на Java, а может быть на Python. А может быть и вовсе на Visual Basic если бы Netscape не успели до запуска Internet Explorer.

Первый Internet Explorer появился в 1996 и Microsoft пришлось писать свою реализацию языка JavaScript. Они назвали её JScript. Реализации получились схожие, но некоторые детали отличались.

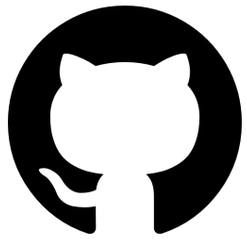
Различия в реализации и открытая природа Web'a несовместимы поэтому для того чтобы предотвратить в будущем ещё большее расхождение в реализациях, был создан стандарт ECMAScript и это официально название языка. JavaScript же является просто коммерческим названием ECMAScript. В стандарте ECMAScript 1 описали синтаксические конструкции и поведение, которые были заложены Бренданом Айком при разработке JavaScript для Netscape.

Развитие ECMAScript



Как уже было сказано, в ECMAScript 1 было заложено всё то, что накодил Брендан за 2 недели (объявление переменных, циклы, ветвление, функции, объекты). ECMAScript 2 не добавил в описание стандарта никаких возможностей, в нём лишь поправили формулировки.
В ES3 появились регулярные выражения, do-while, try {} catch и исключения, встроенные функции для массивов и строк, операторы in и instance of.
Как можно заметить, между ES3 и ES5 провал в 10 лет. В это время Adobe, Mozilla и Microsoft занимались разработкой стандартов ES3.1 и ES4. Процесс вышел из под контроля, язык становился всё сложнее и сложнее, хотели внедрить классы, модули, интерфейсы, генераторы, итераторы, дженерики, рефлексию. А ведь изначально язык проектировался лёгким и простым! Комитет решил отказаться от публикации и приступили к разработке ES5. Все наработки попали в ActionScript, который вы могли встретить если кодили под Adobe Flash Player.
В ES5 не стали внедрять всё что придумали за 10 лет, а вдумчиво проредили весь список фич и добавили геттеры и сеттеры, новые методы для объектов, строк, дат, и массивов, проинтегрировали формат данных JSON прямо в язык, и самое главное – внедрили строгий режим исполнения, который делает язык более надёжным и запрещает много вольностей разработчикам.
Стандарт ES2015 приблизил ECMAScript к «нормальным» языкам программирования, добавив классы, модули, интерполяцию строк и много-много фич, но все их мы разберём последовательно на следующих лекциях. После ES2015 закончились неадекватные скачки в развитии и теперь редакции языка выходят раз в год (ES2016, ES2017...).

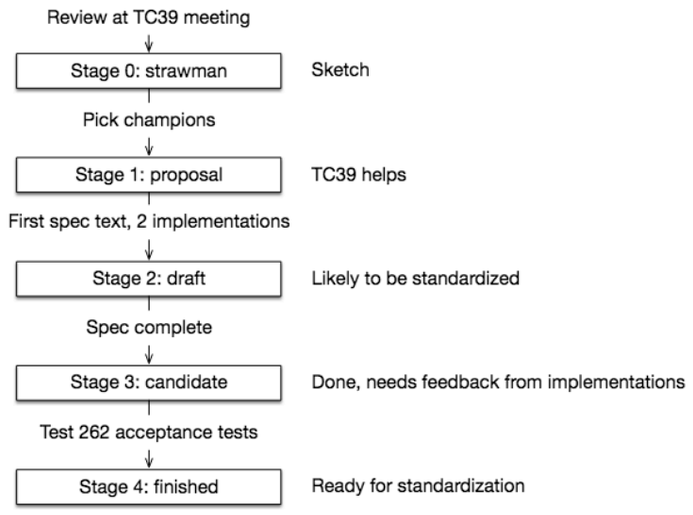
Разработка языка



tc39/ecma262

Разработка стандарта ведётся открыто. В ней принимают участие представители больших компаний, сами члены комитета, а также любители. Вы можете придти вечером, открыть ссылку, почитать там спецификацию языка и оформить Pull Request, предложить в нём новый синтаксис, исправить старое поведение, добавить новые методы, глобальные объекты и функции. Заходите, делайте свой вклад, получайте лайки и дизлайки. Может быть через полгода-год ваша идея окажется в реализациях языка, во всех браузерах и других средах исполнения JavaScript.

Процесс принятия предложений



Почему полгода-год? Процессы в комитете протекают не быстро, а оно и понятно, потому что то что вы напишете завтра им на GitHub'е вскоре может задеть каждого пользователя в мире.

Характеристики языка

- Интерпретируемый*
- Мультипарадигменный
- С динамической слабой типизацией

Поговорим наконец по делу.
JavaScript – интерпретируемый язык. Вам не нужно предварительно компилировать его чтобы где-то исполнить. Просто пишем код в .js файликах, отдаём интерпретатору и он его исполняет. Почему я пометил этот пункт звёздочкой? Для нас, как для программистов-пользователей это выглядит именно так, на самом деле js ещё как компилируется и мы скоро рассмотрим более подробно.
Язык поддерживает разные парадигмы программирования, он императивный, функциональный и объектно-ориентированный.
Этими свойствами сейчас обладает любой язык программирования, будь то Java, C#, C++, etc. Поэтому нас скорее интересует последний пункт.
Разберём его по кусочкам.

Динамическая типизация

```java
String name;
name = "John";
```

Java

```javascript
let name;
name = 'John';
```

JavaScript

Тип – это область допустимых значений. Например, целое число может обладать значениями 0, 1, 2 и т.д., булево – ложь или истина.
Статически типизированные языки ограничивают **типы переменных**, язык программирования знает, например, что name это String. И программисту запрещается делать name = true. Компилятор просто откажется компилировать такой код.
Динамически типизированные языки помечают **значения** типами. Язык знает, что "John" это string, 2 это number, но он не может знать, что переменная name всегда содержит string.
Статически типизированные языки проверяют типы в программе ещё во время компиляции.
Динамически типизированные языки не требуют указывать тип, и не определяют его сами. Типы переменных неизвестны до момента запуска программы.
В JavaScript используется динамическая типизация.

Слабая типизация

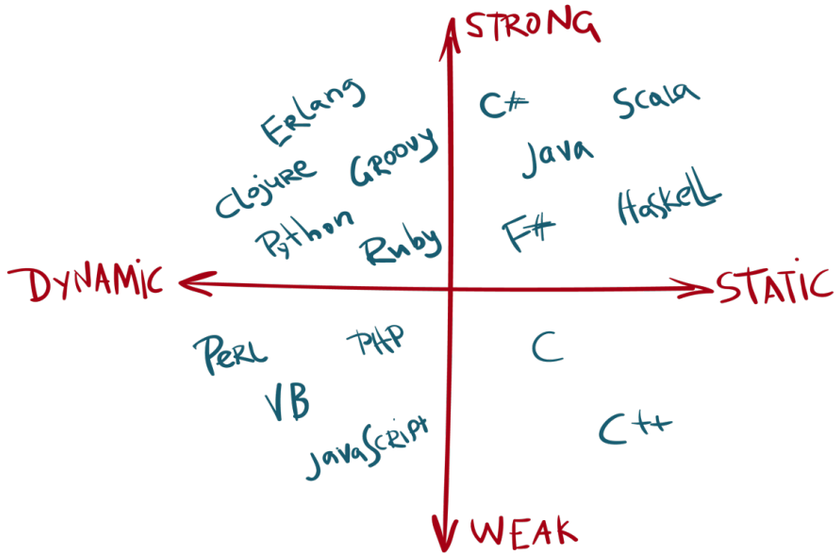
```
>>> 'x' + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

Python

```
> 'x' + 1
'x1'
```

JavaScript

Понятия "Слабая" и "Сильная" типизация неоднозначные. Чаще всего когда их употребляют, то говорят о неявных преобразованиях типов.
Если язык позволяет себе много вольностей, позволяет складывать float'ы с int'ами, строки с булевыми значениями, то его можно охарактеризовать как язык со слабой типизацией. Чем строже проверки типов при операциях тем "сильнее" типизация в языке.
Так, в Python хоть и динамическая типизация, но она сильная и во время исполнения происходят проверки на типы, вызываются эксепшены если операция над переданным типом недопустима.
JavaScript же, в противовес, не запрещает ничего. В JS слабая типизация и нужно быть аккуратным в вычислениях.



Не путайте характеристики слабая/сильная и статическая/динамическая в контексте типизации, это разные вещи. Помните же что JS был спроектирован как язык для любителей, скриптеров, дизайнеров? Из-за таргетинга на такую аудиторию, он и обладает слабой динамической типизацией. В противовес – Java для «профессионалов», которая защищает программистов от невалидного кода, некорректных операций над типами ещё на стадии компиляции.

За последние 3 слайда вы уже могли расстроиться, похоронить в себе всё желание стать JavaScript разработчиками и пойти после лекции учить Java. Но не всё так плохо. У JS ниже порог вхождения, практически ничего не нужно знать и настраивать чтобы писать на нём. Он позволяет быстро начать реализовывать свои идеи в коде. В этом его сильные стороны. Если ваш проект разрастается, то да, JS вам не помощник и поддерживать огромную кодовую базу написанную на нём сложно. От этой болезни есть лекарство – TypeScript. Как говорится в слогане Microsoft'а, это JavaScript, который масштабируется. Мы в Яндексе долгое время писали на JS, но сейчас общий вектор развития – переход на TypeScript и React. До TypeScript мы дойдём к концу этого семестра, а весь код в следующем семестре будет писаться на TypeScript.

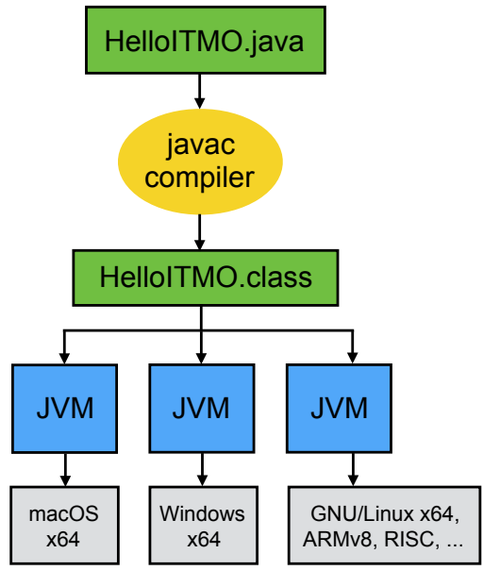
"Hello, ITMO!" Java Edition

```
class HelloITMO
{
    public static void main(String args[])
    {
        System.out.println("Hello, ITMO!");
    }
}
```


Давайте перейдём от болтовни к коду и поймём каким образом и где исполняется код на JavaScript. Но сначала рассмотрим, в упрощённом виде, как выглядит этот процесс в Java мире.

Напишем вариант классического примера с первых страниц любого учебника по программированию.

Исполнение кода



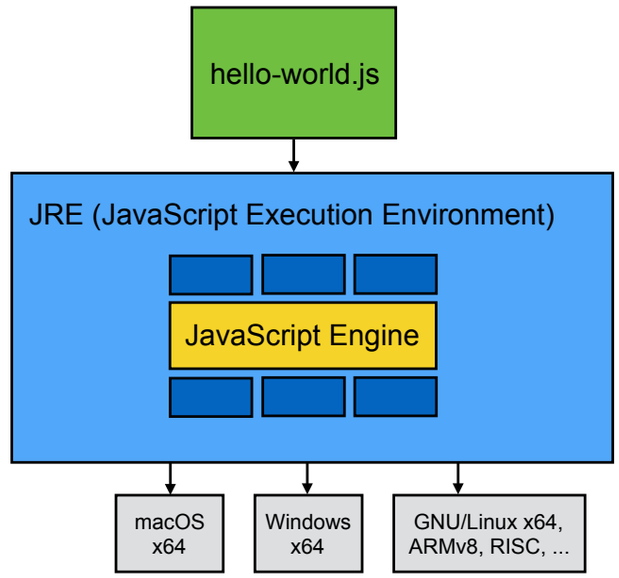
Посмотрим какие шаги необходимы для того чтобы запустить исходный код на Java.
Берём компилятор Java, он преобразует наш .java файл с исходниками в байткод который понимает виртуальная машина и запишет его в HelloITMO.class.
Далее байткод отправляется на исполнение в JVM – виртуальную машину Java. Она, в свою очередь, байткод преобразует в машинный код той платформы, на которой она исполняется. Далее в ходе исполнения она последовательно оптимизирует машинный код и делает его самые горячие куски быстрее.

"Hello, ITMO!" JavaScript Edition

```
console.log('Hello, ITMO!');
```

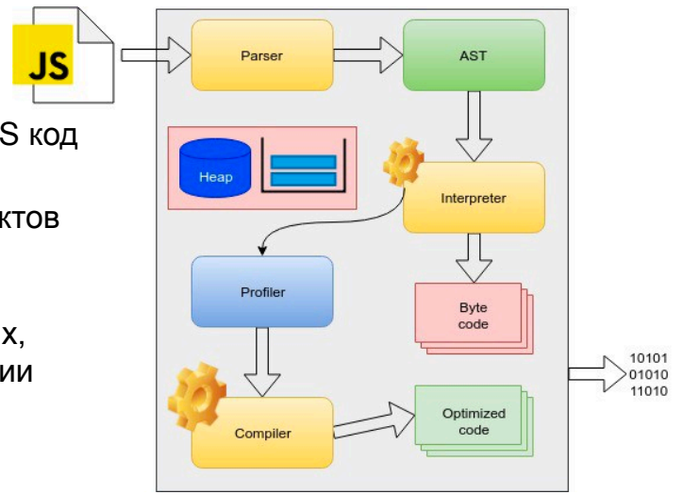

Теперь точно такой же пример, но на JavaScript. Ко можно видеть, прелюдий нужно существенно меньше.
На скрине мы видим некоторый объект console, у которого вызывается метод log и в качестве аргумента ему передаётся строка. Метод, как вы уже догадались, печатает в консоль Hello, ITMO. А что под капотом? Где и как мы можем этот код исполнить? Давайте посмотрим.

Исполнение кода



Поместим наш hello-world в файл с исходным кодом. Его может исполнить движок JavaScript (JS Engine).
JS Движок – это программа, которая построчно читает исходный код на JS и преобразует его в машинный, а затем исполняет результат.
JS движки не запускаются в изоляции, они всегда встраиваются в контексте какой-то среды исполнения. Но давайте сначала чуть больше о движках.

Функции JavaScript движка



〉 Компилирует и исполняет JS код
〉 Управляет стеком вызовов
〉 Выделяет память для объектов
〉 Очищает память от неиспользуемых объектов
〉 Предоставляет типы данных, операторы, объекты и функции

〉 Ничего не знает о среде исполнения

Популярные JavaScript движки работают по следующему принципу:
Берут JS код, парсят его в AST, отправляют интерпретатору (программе, которая преобразует JS код в неоптимальный машинный и быстро исполнит). Затем в дело вступает профилировщик, который соберёт с программы во время исполнения статистику использования тех или иных кусков кода, а затем по наиболее горячим из них проходится оптимизирующий компилятор, который сгенерирует более производительный машинный код.
В ходе работы программы вызываются JS функции, за порядок их вызова отвечает call stack. Функциям нужно работать с данными, создавать объекты и затем очищать их. Память выделяется JS движком в Memory Heap (куче), и затем неиспользуемые объекты очищаются сборщиком мусора (Garbage Collector'ом). При этом JS движок сам по себе не знает где он исполняется, на сервере ли или в браузере, не знает какие функции, объекты, библиотеки доступны, а это определяет JRE (JavaScript Runtime Environment).

JavaScript движки

- V8 от Google (Node.js, Chrome, Yandex.Browser, ...)
- JavaScriptCore (Safari, все браузеры на iOS)
- SpiderMonkey от Mozilla (Firefox, Gnome)
- Chakra (Edge) †
- Nashorn (JDK)
- GraalVM
- NJS (Nginx)
- ...Over 9000 других

Таблица совместимости

JavaScript движков масса. Их общая черта – все они реализуют в той или иной мере спецификацию ECMAScript. Есть таблица совместимости на которой отмечаются все живые движки и насколько полно они поддерживают стандарт.

Самые известные это:

- V8 от Google, он применяется в серверной платформе Node.js, в браузерах на основе Chromium, а сейчас почти все они строятся именно на базе него.
- JavaScriptCore – движок применяемый в iOS устройствах и в десктопном Safari на маках.
- SpiderMonkey – Реинкарнация самого первого JS движка, который разработали в Mozilla в 1995-ом. Он до сих пор в тонусе и активно развивается.
- Chakra – Можно похоронить, Microsoft развивал его как альтернативу V8 и в этом году отказались от своей разработки в пользу использования V8 в Edge.
- В Java мире JS тоже активно используется для скриптинга и JDK, GraalVM умеют интерпретировать JavaScript.
- В самый популярный Web-сервер в мире Nginx, встроили движок NJS для пре- и пост- обработки соединений и запросов.
- И ещё помимо этих движков есть сотни других, многие из них узкоспециализированные.

Среды исполнения

V8

?

V8

~~Chakra~~ V8

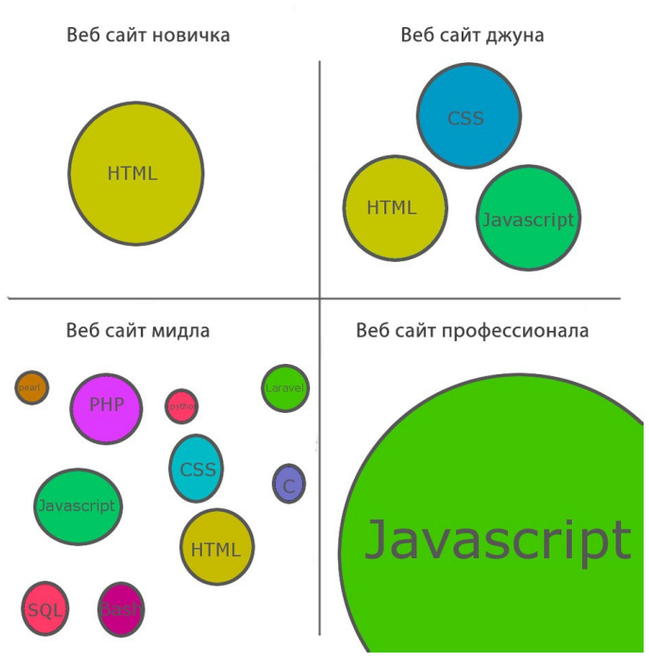
SpiderMonkey

V8

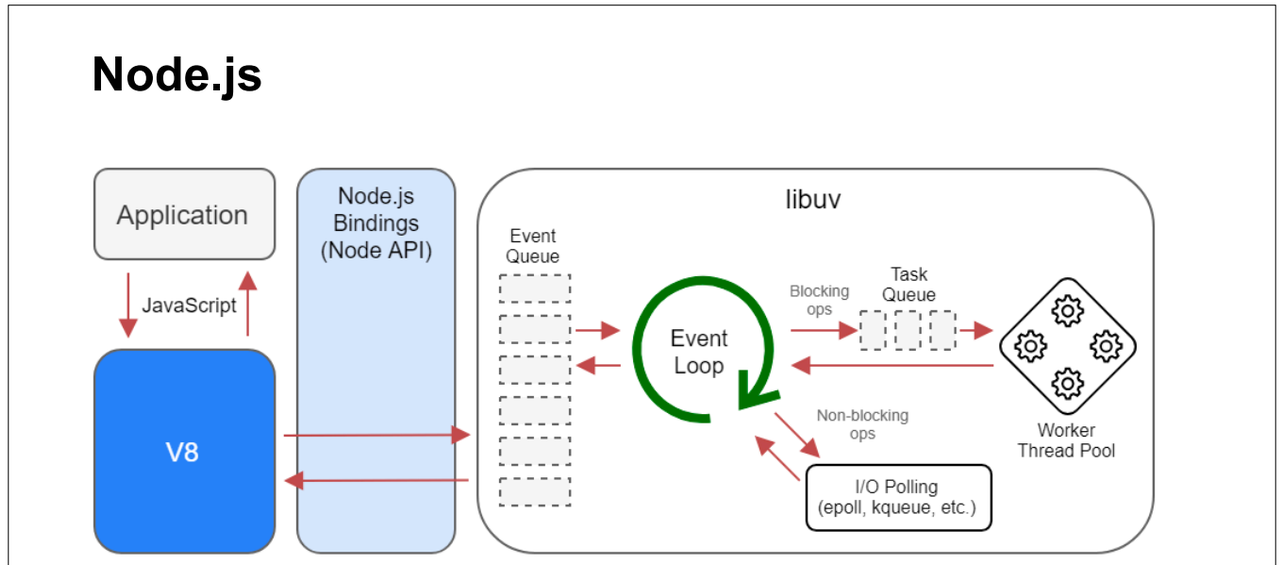
JavaScriptCore

Пока что мы говорили о сферическом JavaScript в вакууме, но вот уже приближаемся к реальному миру. JavaScript код может запускаться где угодно, куда смогли встроить JavaScript движок (Даже в какой-нибудь умный чайник).
Node.js – среда исполнения JavaScript на сервере, она использует движок V8 и позволяет написать, при желании, бэкенд или заскриптовать рутинные действия, но больше всего Node.js помогает при решении типичных фронтендерских задач – скомпилировать код, оптимизировать картинки, сжать стили, прогнать тесты, задеплоить в продакшн.
Далее – браузеры. Всё что основано на браузере Chromium (Chrome, Opera, Yandex Browser, теперь уже и Edge) использует V8, в Firefox уже больше 20 лет живёт SpiderMonkey, а в Safari – JavaScriptCore.

Если раньше JavaScript продвигали как язык для создания анимашек для дизайнеров, то сегодня можно создать систему умного дома используя только JS, написать высоконагруженный бэкенд прикрутив к нему машинное обучение на TensorFlow.js, сделать мобильное приложение и веб приложение или 3D игру с крутой графикой.



На этот счёт есть даже мем. Зачем учить какой-то другой язык программирования если всё можно написать на JavaScript.



Высокоуровнево архитектура Node.js выглядит следующим образом:

В основе лежит V8, он исполняет JS код. Node.js предоставляет программисту набор библиотек, функций и глобальных объектов (Node API). А сам этот программный интерфейс Node написан по большей степени на C++ и взаимодействует с операционной системой.
С этого слайда стоит запомнить следующее – JavaScript в большинстве сред исполнения однопоточный. Вся мультизадачность перекладывается на Node API и тот C++ код, который стоит за ним. Читаете файл? Создаётся поток в OS, читает его, а Node.js продолжает исполняться. К вам пришёл сетевой запрос? Аналогично. Про внутреннее устройство Node.js у нас будет лекция в начале следующего семестра.

А что в браузере? На самом деле схема точно такая же. Только вместо Node API будет Browser API. API ноды и браузера сильно отличаются. Что и логично. Про браузер тоже будет отдельная лекция.

Декларация переменных

```
const year = 1999;
year = 2000;  // TypeError: Assignment to constant variable.

let year;
year = 1999;
```


Закончим сегодня о высоком, вернёмся к нашим hello world'ам. Посмотрим как объявлять переменные. Для этого у нас есть 2 ключевых слова – const и let. Есть ещё, конечно, var, но современные программисты вспоминают его с содроганием и лучше не тревожить это древнее зло.

const позволяет объявить переменную, значение которой не может быть впоследствии переприсвоено. Если вы попытаетесь это сделать, то произойдёт ошибка.

На случай если вам нужно переприсваивать значение, то используйте let, но ваш выбор по умолчанию – const.

Акцентирую ваше внимание на точках с запятой. Хоть язык и позволяет опускать их, как и Kotlin, но общепринятая практика – выставлять их. Эта практика спасает от некоторых глупых ошибок при рефакторинге.

Именование переменных

```
const year = 1999;       // OK
const $year = 1999;      // OK
const _year = 1999;      // OK
const year1999 = 1999;   // OK
const 1year = 1999;      // Not OK!
```

Первым символом может быть буква, $ или _.
Последующие символы могут содержать всё вышеперечисленное + цифры.

Название переменной, функции, класса или метода может начинаться с буквы, знака доллара или _. После первого символа можно использовать всё тоже самое, но ещё и цифры.

Именование переменных

```
const myBirthday = '07.02.1996';

const COLOR_RED = '#f00';
```

camelCase – по дефолту

SCREAMING_SNAKE_CASE – для трудно запоминаемых значений, которые известны до начала исполнения программы

Если допустимые символы – ограничение интерпретатора, то использование camelCase это уже соглашение, которое принято в JS комьюнити. Постарайтесь не использовать snake_case в коде, даже если вы днём пишете на JS, а ночью на Python.

Есть исключение – если какое-то значение известно до начала исполнения программы и оно трудно запоминается, то вынесите его в константу и назовите её в кричащем snake case.

Комментарии и JSDoc

```
// Однострочный комментарий
console.log('Привет');

/*
 * Многострочный
 * Комментарий
 */

/**
 * Складывает два целых числа
 * @param {Number} a Первое целое
 * @param {Number} b Второе целое
 * @throws {TypeError} Когда в аргументы переданы не числа
 * @returns {Number} Сумма аргументов
 */
```


Говорят, надо писать код настолько лаконичный, что ему не нужны комментарии. Это правда. Но не всегда такое возможно.
Если вам всё таки нужно написать комментарий, то вы можете сделать это в одну строку, в несколько строк, или вовсе развернуться и написать целый кусок документации на специальном синтаксисе называемом JSDoc. JSDoc стандартизован, помогает описывать принимаемые аргументы, то как ведёт себя функция, что возвращает и зачем вообще нужна. JSDoc также позволяет указывать типы данных на входе и на выходе, чем пользуются современные редакторы кода и показывают вам подсказки.

Типы данных. Number

```
const i = 123;
const d = 12.345;

const n = NaN;
const pi = Infinity;
const ni = -Infinity;

1 / 0; // Infinity
Infinity/Infinity; // NaN
```

Числовой тип данных представляет как целочисленные значения, так и значения с плавающей точкой.
Все числа в JavaScript хранятся в формате IEEE754, и, если говорить в терминах Java, то все числа в JS имеют тип данных Double.
Кроме обычных чисел, существуют специальные числовые значения – NaN, Infinity, -Infinity.
Infinity – математическая бесконечность, она больше любого числа
-Infinity – тоже самое, но меньше любого числа
NaN – Not a Number, это значение означает арифметическую ошибку. Оно вернётся, если вдруг вы попробуете, к примеру, поделить строку на число.

Типы данных. String

```
const str = "Привет";
const str2 = 'Одинарные кавычки тоже подойдут';
const phrase = `Обратные кавычки позволяют встраивать переменные ${str}`;
```

〉 Между ' ' и " " разницы нет
〉 Обратные кавычки (` `, backticks) позволяют выполнять интерполяцию
〉 Интерполяция быстрее и более читаема чем конкатенация 'str1' + 'str2'

В JavaScript существует три типа кавычек.
- Двойные кавычки: "Привет".
- Одинарные кавычки: 'Привет'.
- Обратные кавычки: `Привет`.

Между одинарными и двойными нет никакой разницы. Нет в JavaScript типа данных char, как в C/C++/Java.
Обратные кавычки имеют расширенную функциональность. Они позволяют встраивать в них выражения. Это называется **интерполяция**.

Типы данных. Null и Undefined

```
let age = null;
```

- null – отдельный тип данных
- Означает «Ничего», «Пусто», «Значение неизвестно»
- Предназначен для явного присвоения

```
let x; // undefined
```

- undefined – отдельный тип данных
- Означает «Значение не было присвоено»
- Явно присваивать не рекомендуется

Типы данных. Boolean

```
const isTestFinished = true;

const isStudentEnrolled = false;

const areYouBeautiful = 2 > 1;
```

- Булевый тип принимает значения true и false
- Булевые значения могут быть результатом сравнений
- К false преобразуются: 0, пустая строка, null, undefined, NaN
- Остальные значения преобразуются к true

Условные операторы: if, ?

```
if (year === 2019) {
  console.log('Yeah!');
} else if (year === 2020) {
  console.log('May be…');
} else {
  console.log('Nooo!!!');
}
```

```
const isAccessAllowed = (age > 18) ? true : false;
```

- if (...) вычисляет условие в скобках и выполняет соответствующий блок
- Фигурные скобки для блоков рекомендуется использовать всегда
- Вложенные тернарные операторы очень плохо читаются

== против ===

```
0 == false; // true

0 === false; // false, так как сравниваются разные типы
```

- При == происходит неявное приведение типов
- Используйте всегда ===

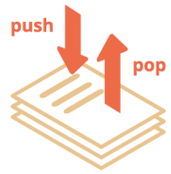
Типы данных. Массивы

```
const arr = new Array();
const arr = [];
```

```
const assortment = [
  'action',
  'fantasy',
  'sci-fi',
  null,
  undefined,
  true,
  1
];
```

- Короткий вариант объявления ([ ]) предпочтителен
- Могут хранить элементы любого типа
- Можно задавать значения при инициализации

Типы данных. Методы массивов



- Массивы в JS – двусторонняя очередь
- push – добавляет в конец
- pop – извлекает из конца
- shift – извлекает из начала
- unshift – добавляет в начало

Циклы

```
while (condition) {
  // "тело цикла"
}

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

for (начало; условие; шаг) {
  // Тело цикла
}
```

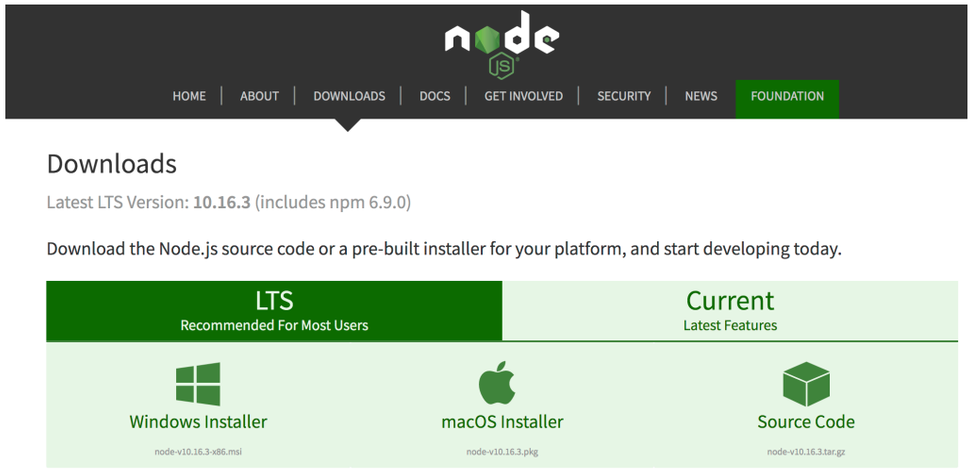
〉 Всё как в C

Функции

```
function имя(параметры) {
  // Тело функции
}

function checkAge(age) {
    return age >= 18;
}

const age = 17;

if (checkAge(age)) {
  console.log('Доступ получен');
} else {
  console.log('Доступ закрыт');
}
```

- Названия в camelCase
- В названии функции должен быть глагол
- Если нет return, то возвращается undefined

Установка Node.js



Посмотрим, какие инструменты нужны для современной разработки фронтенда.

В первую очередь нужно установить Node.js. Есть 2 версии: LTS и Current. LTS – версия с долгим сроком поддержки, Current – та, которая сейчас активно развивается. Рекомендую всегда использовать LTS, сэкономите себе нервы и время. Если хочется экспериментов – ставьте Current. Платформа доступна под все широкоизвестные операционные системы.

Выбор браузера


Chromium


Chrome


Yandex Browser


Firefox

В любом браузере, даже в Internet Explorer есть инструменты разработчика, но они крайне неудобны. В Edge и Safari вы тоже будете страдать. Для целей разработки лучше завести отдельный браузер. Вам будет комфортно в любом, который основан на Chromium, например Chrome или Yandex Browser, либо в Firefox. Почему стоит отделить тот в котором вы разрабатываете, от того которым пользуетесь? Постепенно ваш девелоперский браузер обрастёт расширениями, всякими тестовыми логинами и вряд ли они нужны вам в повседневной жизни.

Текстовый редактор

Visual Studio Code

WebStorm

Есть 2 наиболее удобных и прокачанных редактора кода для JavaScript, TypeScript и других околоWebных технологий – Visual Studio Code от Microsoft и WebStorm от JetBrains. VSC бесплатный, опенсорсный и развивается семимильными шагами. WebStorm это платное ПО с закрытым исходным кодом. Вам, как студентам или сотрудникам JetBrains (я предполагаю что такие есть), можно получить эту IDE бесплатно.
Пишу код в обоих редакторах, по настроению :) Но большее предпочтение отдаю Visual Studio Code. Он проще в освоении, местами лучше работает автодополнение. Выберите сами для себя что удобнее, попробуйте и то и другое.

Плагины для VSCode

EditorConfig

ESLint

Prettier

Stylelint

Live Server

В WebStorm всё необходимое уже есть, а вот тем кто выберет VSCode рекомендую поставить следующие плагины:

- EditorConfig – поможет вашему редактору кода ставить правильное количество пробелов, пробелы вместо табов и так далее.
- ESLint – линтер для JS, это утилита, которая проверяет ваш код на соответствие стилю, подсказывает участки кода, в которых могут быть потенциальные ошибки
- Prettier – автоформаттер, делает код красивым по щелчку пальца

Заранее подскажу, для UX курса желательно поставить Stylelint плагин для проверки CSS и Live Server чтобы HTML который вы пишете автоматически обновлялся в браузере и не приходилось перезагружать страницу.

Git

https://git-scm.com

Вам понадобится git – система контроля версий. Она нужна для версионирования кода и для совместной разработки, ну и конечно для сдачи домашних заданий. Очень надеюсь, что к 3 курсу вам уже удалось с ней поработать. Если нет, то изучайте, советуйтесь с однокурсниками, спрашивайте нас.

Демо с отправкой домашнего задания

Домашнее задание (01-warmup)

- В утро воскресенья будет выдан доступ в GitLab
- Разработка в отдельной ветке (название ветки - ваш логин)
- Название Merge Request – Имя Фамилия
- После прохождения автотестов будет назначен ментор

- Дата и время старта: 22.09 11:40 (воскресенье)
- Дедлайн для прохождения автотестов: 29.09 11:40 (+ неделя)
- Дедлайн для ручной проверки ментором: 06.10 11:40 (+ ещё неделя)

Что читать?

- MDN
- learn.javascript.ru

Яндекс

Спасибо

Евгений Марков

Разработчик интерфейсов

evgenymarkov@yandex-team.ru