Глава 1

JavaScript. Введение

1.1. Введение

Изначально JavaScript развивался для того, чтобы сделать интернет-страницы более отзывчивыми и интерактивными. Чтобы пользователь во время взаимодействия со страницей получал мгновенный feedback от своих действий.

До JavaScript для этих целей использовался язык Java. Создание динамической страницы требовало от Java-программиста много усилий. Написанный код нужно было сначала скомпилировать, затем упаковать результат компиляции в апплет и подключить его к странице. Такое положение вещей не соответствовало высоким темпам развития всемирной сети. Использование JavaScript позволяет исключить лишние действия, тем самым упрощая жизнь разработчику: достаточно написать код и подключить его на страницу.

Можно выделить следующие особенности языка JavaScript:

Синтаксис → Java, C, C++. Синтаксис языка был сделан похожим на синтаксис Java, C и C++. Это было сделано специально, чтобы разработчики, которые писали на этих языках, легко могли освоить и JavaScript.

Динамическая типизация Динамическая типизация была позаимствована из Perl, который был популярным в то время языком.

Ссылки на функции → Lips. Функции в JavaScript являются объектами первого класса, то есть их можно использовать и передавать по ссылке как аргументы.

Наследование через прототипы → Self. Наследование в JavaScript реализовано через прототипы. Эта идея была заимствована из языка Self.

JavaScript был разработан Брэндоном Айком для компании Netscape в 1995 году.

Создатель JavaScript, Брэндон Айк.

Вот как сам Брэндон пишет о том, как создавался язык:

> ... он должен был быть написан за 10 дней, а иначе мы бы имели что-то похуже JS...
>
> ... В то время мы должны были двигаться очень быстро, т.к. знали, что Microsoft идет за нами...
>
> ... JS был обязан «выглядеть как Java», только поменьше, быть эдаким младшим братом-простаком для Java...

Следующим важным этапом создания JavaScript было появление формата JSON (JavaScript Object Notation). JSON разработал Дуглас Крокфорд в 2001 году, чтобы заменить популярный в то время формат XML.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<coffee_shop>
    <name>Works</name>
    <cashlessPayment>true</cashlessPayment>
    <capacity>3</capacity>
    <barista>
        <persone>
            <name>Лёша</name>
            <favourite>cappuccino</favourite>
        </persone>
        <persone>
            <name>Лиза</name>
            <favourite>tea</favourite>
        </persone>
    </barista>
</coffee_shop>
```

Listing 1: Пример данных в формате XML.

Формат XML сам по себе является избыточным. Поэтому, чтобы ускорить передачу данных по сети, он был заменен на JSON.

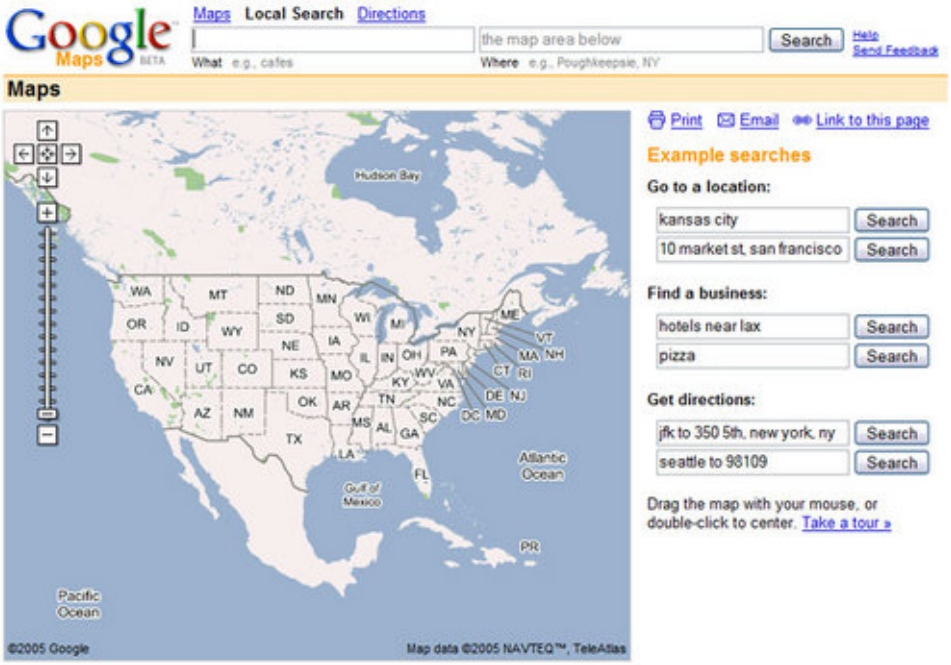
```json
{
    "name": "Works",
    "cashlessPayment": true,
    "capacity": 3,
    "barista": [
        {
            "name": "Лёша",
            "favourite": "cappuccino"
        },
        {
            "name": "Лиза",
            "favourite": "tea"
        }
    ]
}
```

Listing 2: Пример данных в формате JSON

JSON также является более выразительным, чем XML, в дополнение к тому, что он не является избыточным. Кроме того, с JSON было легко

работать из JavaScript, поскольку данные в формате JSON были представлениями объектов JavaScript.

В 2005 году Джеймс Гаррет разрабатывает AJAX (Asynchronous JavaScript and XML). AJAX позволяет подгружать данные без обновления веб-страницы.



Google maps, 2005 год

С использованием этого подхода были реализованы карты Google. В результате этого пользователь мог подгружать нужный фрагмент карты без необходимости обновлять страницу браузера. В то время это было прорывом.

JavaScript распространялся с бешеной скоростью и вскоре был в браузере почти у каждого пользователя. Но кроме этого JavaScript стал пригоден для разработки серверной части. Появился NodeJS, позволяющий запускать на сервере скрипты на JavaScript. Он набрал популярность за счет следующего:

Кроссплатформенность node.js работал на любой операционной системе, в том числе Linux, Windows, MacOS и так далее ...

Единая архитектура Код для сервера и клиента написаны на одном языке. Это позволяет избежать ряда проблем.

Один поток В NodeJS код исполняется в одном потоке, за счет чего он становится проще и его легче разрабатывать. Исчезают многие трудности, которые появляются при работе с многопоточным кодом: не нужно переключаться между потоками, входить в зону блокировок и так далее.

Поддержка сообщества Большинство повседневных задач, как правило, уже решены и доступны в качестве пакета в Node Package Manager (NPM). На данный момент NPM является самым быстроразвивающимся пакетным менеджером.

Таким образом, JavaScript на данный момент можно использовать не только в браузере, но и на сервере. Более того, можно даже делать запросы в базы данных с использованием JavaScript.

1.2. Типы данных

В JavaScript существуют 6 типов данных:

- Числа
- Строки
- Булевые величины
- Неопределённые величины
- Объекты и массивы
- Функции

1.2.1. Числовой тип

Чтобы определить число, в JavaScript используется запись, состоящая из цифр.

```
123     // 123
12.3    // 12.3
```

По умолчанию используется десятичная система счисления, но также можно использовать шестнадцатиричную (используется префикс $0x$), восьмеричную (используется префикс $0o$) и двоичную (используется префикс $0b$) системы.

```
0x11    // 17
0b11    // 3
0o11    // 9
```

В JavaScript на хранение числа отводится 64 бита. Казалось бы, максимальное целое число, представимое в JavaScript равно

$$2^{64} = 18446744073709552000.$$

На самом деле это не совсем так. Во-первых, 1 бит используется для хранения знака числа. Несколько бит используются для хранения позиции десятичной точки, поэтому максимальное целое число, представимое в JavaScript, равно:

```
Number.MAX_SAFE_INTEGER
// 9007199254740991
```

Это число всегда доступно как свойство `MAX_SAFE_INTEGER` объекта Number.

Для работы с большими значениями можно использовать экспоненциальную запись числа. Слева от e расположено основание экспоненты (любое десятичное число), а справа — степень экспоненты.

```
2.998e8
// 2.998 × 10^8
// 299800000
```

Следует помнить, что вычисления не с целыми числами будут неточными.

Также в JavaScript есть два особенных числа:

```
Infinity
-Infinity
```

Если вычесть из бесконечности единицу, получится по-прежнему бесконечность:

```
Infinity - 1 === Infinity // true
```

`Infinity` ($+\infty$) больше любого другого представимого в JavaScript числа. Аналогичным образом ведет себя `-Infinity`

Кроме того, в JavaScript есть особенное число `NaN` (Not a Number)

```
NaN
```

Это число возникает как результат недопустимых арифметических операций:

```
0/0 // NaN
Infinity - Infinity // NaN
'один' / 'два' // NaN
```

1.2.2. Строковый тип

Следующий тип данных — строки. Для того, чтобы представить строку в JavaScript, произвольный текст обрамляется одинарными или двойными кавычками:

```
'строка текста'
"строка текста"
' español русский \' '
```

Оба варианта допустимы, однако рекомендуется заранее для себя выбрать предпочтительный тип представления строки и придерживаться ему. Внутри строки можно использовать Unicode-символы, а также символы одинарной или двойной кавычки (для этого их нужно экранировать с помощью обратного слеша).

1.2.3. Логический тип

Логический тип представляет булевы величины (принимают значения true и false).

```
true
false
```

Самый простой способ получить булеву величину — выполнить операцию сравнения.

1.2.4. Неопределённые величины

Неопределённые величины — самый необычный тип данных в JavaScript.

```
undefined
null
```

При определении переменной ее значение по умолчанию не определено (undefined):

```
var a;
console.log(a); // undefined
```

1.2.5. Оператор typeof

Оператор typeof позволяет узнавать тип значения. Результат возвращается в виде строки.

```
typeof 0;   // 'number'
typeof '0'; // 'string'
```

1.2.6. Преобразование к числу

Для того, чтобы преобразовать явно строку к числу, можно использовать ParseInt и ParseFloat. ParseInt принимает два аргумента.

```
ParseInt(string, radix);
```

Первый из них — это строка (если первый аргумент не строка, он неявно приводится к строке при помощи метода toString), которую нужно привести к числу, а второй — система счисления.

```
ParseInt('17', 10); // 17
ParseInt('123');    // 123
ParseInt('11', 2);  // 3
```

Если второй аргумент не указан, ParseInt самостоятельно подбирает систему счисления. Однако, в зависимости от реализации, он может это делать по-разному, поэтому настоятельно рекомендуется указывать систему счисления всегда.

Если функция ParseInt не может разобрать первый символ строки, то он возвращает NaN.

```
ParseInt('a1', 10); // NaN
ParseInt('2b', 10); // 2
```

Если же строка начинается с числа, за которым идет не число, то ParseInt возвращает число, которое он смог прочитать.

Функция ParseFloat является двойственной к функции ParseInt и позволяет приводить строки к вещественным числам.

```
ParseFloat(string);
```

В отличие от ParseInt, ParseFloat принимает один аргумент, строку, и работает всегда в десятичной системе счисления:

```
ParseFloat('3.14');     // 3.14
ParseFloat('314e-2');  // 3.14
ParseFloat('a1');   // NaN
```

ParseFloat может принимать как строки, представляющие числа с использованием точки, так строки, представляющие числа в экспоненциальной записи. Если передать в ParseFloat строку, которая начинается не с числа, результат будет NaN.

1.3. Переменные

1.3.1. Определение переменной

Для того, чтобы определить новую переменную, используется ключевое слово var и вслед за ним название переменной.

```
var studentsCount;
studentsCount = 98;
```

После определения переменной, ее можно использовать, например присвоить значение, как в примере выше.

Можно использовать более короткую запись и присваивать значение в момент объявления переменной:

```
var studentsCount = 98;
```

Кроме того, с помощью оператора «запятая» можно объявить несколько переменных после одного ключевого слова var:

```
var studentsCount = 98,
    language = 'JavaScript';
```

1.3.2. Допустимое имя переменной

В качестве первой буквы переменной можно использовать буквы, символы нижнего подчеркивания и доллара:

```
a-z _ $
```

Однако рекомендуется для первого символа использовать буквы нижнего регистра латинского алфавита. Имя переменной не может начинаться с цифры. Остальные символы в имени переменной могут быть буквами, цифрами, символами подчеркивания и доллара:

```
a-z 0-9 _ $
```

Использовать знак минус (-) в имени переменной нельзя.

В качестве названия переменной нельзя использовать зарезервированные слова:

```
break    do       try        while
case     else     new        with
catch    finally  return
continue for      switch
debugger function this
default  if       throw
delete   in       instanceof
typeof   var      void
```

Кроме того, с выходом новой спецификации этот список пополнился:

```
class enum   extends super
const export import
```

1.3.3. Именование констант

Иногда возникает необходимость в константах, то есть в переменных, значения которых не будут меняться по ходу программы. Чтобы отличать в коде константы от переменных, значение которых не предполагается постоянным, рекомендуется именовать константы буквами верхнего регистра, отделяя слова с помощью символа нижнего подчеркивания:

```
// Переменная
var currentTime;

// Константа
var MILLISECONDS_IN_DAY;
```

1.3.4. Именование переменных

Рекомендуется придерживаться следующих правил при выборе имени переменной, чтобы сделать код более понятным и читаемым:

- Следует избегать транслита в именах переменных.

  ```
  spisokDruzey; // X
  tsena;        // X
  ```

  Обычно такие переменные читаются хуже, а также при использовании внешних библиотек в коде будут присутствовать как переменные, имя которых написано транслитом, так и переменные, имя которых написано английскими словами:

  ```
  friends;      // OK
  price;        // OK
  ```

- Следует избегать слишком коротких и слишком длинных названий переменных:

  ```
  h, w;                         // X
  friendsListWithNameAndAge;    // X
  height, width;  // OK
  myFriends;      // OK
  ```

- Следует использовать для именования переменных так называемый camelCase:

```
my_friends;     // X
myFriends;      // OK
```

  Если название переменной состоит из нескольких слов, начинайте каждое слово, кроме первого, с заглавной буквы.

- Следует учитывать тип данных, который будет у значения этой переменной, при выборе ее имени. Переменные, которые хранят булевое значение, могут начинаться на is, а имена переменных, хранящих массивы, — оканчиваться на s.

```
isCorrect = true;
totalCount = 47;
friends = [];
```

1.4. Комментарии

Чтобы пояснить некоторые участки кода, можно использовать комментарии: строки, которые игнорируются интерпретатором. В JavaScript доступен строчный вид комментариев:

```
// это короткий комментарий
```

А также его блочный аналог:

```
/* а это длинный комментарий
написаный в несколько строк */
```

Комментарий может располагаться на отдельной строке, а также заканчивать строку с кодом:

```
/* ах этот длинный комментарий ... */
var weather = 'cold';

console.log(weather); // cold
```

Рекомендуется использовать только строчные комментарии, потому как использование блочных комментариев может привести к следующей ситуации.

```
var weather = 'sunny';

/*/* ах этот длинный комментарий ... */
```

```
var weather = 'cold';*/

console.log(weather);
// SyntaxError: Unexpected token *
```

1.5. Операторы

Все операторы имеют приоритет. Это означает, что если несколько операторов используются в одной записи, то сначала выполняются операторы с высшим приоритетом. Далее операторы перечислены в порядке уменьшения приоритета.

1.5.1. Унарные операторы

Унарные операторы, то есть операторы, которые применяются к одному операнду, имеют наивысший приоритет.

```
++ (инкремент)
-- (декремент)
+  (унарный плюс)
-  (унарный минус)
!  (логическое НЕ)
```

Унарный минус меняет значение числовой переменной на противоположенное. Логическое отрицание меняет булево значение true на false и наоборот.

Инкремент (декремент) бывает двух видов:

- Постфиксный инкремент:

  ```
  var a = 1;
  var b = a++;  // b === 1, a === 2
  ```

  Сначала производится присваивание, а после этого увеличивается значение переменной на 1.

- Префиксный инкремент:

  ```
  var a = 1;
  var b = ++a;  // b === 2, a === 2
  ```

  Сначала увеличивается значение переменной на 1, а после этого производится присваивание.

1.5.2. Бинарные

Бинарные операторы работают с двумя операндами. Наибольший приоритет среди них имеют бинарные арифметические операторы:

```
* (умножение)
/ (деление)
% (остаток от деления)
+ (сложение)
- (вычитание)
+ (сложение строк)
```

Сложение числовых значений:

```
2 + 3 = 5
```

Сложение (конкатенация) строк:

```
'«JavaScript - это простой, но ' +
'изящный язык, который является ' +
'невероятно мощным для решения ' +
'многих задач» © Джон Резиг'
```

Дальше, по уменьшению приоритета, идут (бинарные) операторы сравнения:

```
<  (меньше)
<= (меньше или равно)
>  (больше)
>= (больше или равно)
==  (проверка на равенство)
!=  (проверка на неравенство)
=== (проверка на идентичность)
!== (проверка на неидентичность)
```

Операторы сравнения возвращают булевы величины. Разница между сравнением на равенство и сравнением на идентичность будет обсуждаться позже в рамках курса.

Далее идут логические операторы:

```
&& (И)
|| (ИЛИ)
```

Оператор логического И имеет больший приоритет среди логических операторов.

Наименьший приоритет среди бинарных операторов имеют операторы присваивания и присваивания с операцией:

`= (присваивание)`

`*=, /=, +=, -=, &=, ^=, |= (присваивание с операцией)`

Следующий код демонстрирует смысл присваивания с операцией:

```
var a = 1;
a += 1;
a = a + 1;
```

1.5.3. Условные операторы

Самый простой способ записать условный оператор — использовать ключевое слово if, после которого в круглых скобках идет логическое выражение.

```
if (language === 'JavaScript') {
    likes = likes + 1;
} else {
    likes = likes - 1;
}
```

Если это логическое выражение истинно, выполняется код в первых фигурных скобках, иначе будет выполнен код, который записан в фигурных скобках после ключевого слова else. При этом else и последующая часть являются необязательными.

Другой вид условного оператора — тернарный оператор. Записывается он следующим образом: логическое выражение, после которого идет знак вопроса. Если это выражение истинно, то выполняется код до двоеточия, иначе выполняется выражение, которое записано после двоеточия.

```
likes = language === 'JavaScript' ?
    likes + 1 :
    likes - 1;
```

Оператор switch-case также является условным оператором:

```
switch (language) {
    case 'JavaScript':
        likes++;
        break;
    case 'C++':
    case 'Java':
        break;
    default:
        likes--;
}
```

После ключевого слова switch в круглых скобках следует выражение, вычисляя которое можно получить некоторое значение. Если это значение совпадает с одним из значений, записанных после ключевого слова case, выполняется код этого case. Иначе выполняется код, записанный в default. Если в case встречается ключевое слово break, то работа case'а прекращается, иначе — происходит проваливание в следующий case.

1.6. Точка с запятой

В примерах выше каждая строчка заканчивается символом «точка с запятой». Это очень важный символ, который обязан присутствовать в конце каждой строки.

Однако, если его пропустить, ошибки не будет. Это связано с тем, что интерпретатор неявным образом поставит точку запятой. Такое неявное добавление точки с запятой может привести к ошибкам в работе программы:

```
function getTrue() {
    return true;
}

getTrue();    // true
```

Теперь, если добавить перенос строки после ключевого слова return, казалось бы функция должна работать также.

```
function getTrue() {
    return
    true;
}

getTrue();    // undefined
```

Интерпретатор неявным образом поставил символ «точка с запятой» после слова return. В результате функция возвращает undefined.

1.7. Строгий режим

Строгий режим появился вместе со спецификацией 5.1. Дело в том, что кроме расширения возможностей языка, в этой спецификации были внесены коррективы, которые нарушают обратную совместимость с написанным ранее кодом.

Чтобы такого не произошло, по умолчанию интерпретатор работает в режиме совместимости с предыдущими спецификациями. Для того, чтобы писать код в соответствии с последней спецификацией, нужно включить строгий режим. Для этого в начале файла или функции следует добавить директиву:

```
'use strict';

// этот код будет работать
// по современному стандарту ES5
```

После включения строгого режима код начинает вести себя иначе. Без строгого режима можно объявить переменную без использования ключевого слова var:

```
text = 'hello';

text; // 'hello'
```

В строгом режиме это приведет к ошибке:

```
'use strict';

text = 'hello'; // ReferenceError:
                // text is not defined
```

Полный список изменений, которые включаются вместе со строгим режимом, доступен по ссылке.

Следует отметить, что строгий режим нельзя выключить. Поэтому, если строгий режим был включен в начале файла и в коде используются внешние библиотеки, которые не готовы к работе в строгом режиме, это может привести к ошибкам.

Рекомендуется в таких случаях не включать строгий режим глобально, а вместо этого включить его в рамках своих функций.

1.8. Пример запуска

Для начала приведем пример кода для запуска в браузере:

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

Данный код выводит в консоль браузера приветственное сообщение. Чтобы запустить программу, нужно перейти в консоль браузера (Ctrl+Alt+J) и скопировать строчку кода в строку ввода.

В результате в консоль выведено сообщение, а после этого (так как код ничего не возвращает) — undefined.

Для того, чтобы запустить код на сервере, нужно сперва установить NodeJS. После это с использованием любого текстового редактора или IDE нужно создать js-файл со следующим содержимым:

```
// index.js
console.log('Hello, world!');
```

Чтобы запустить этот файл, в консоли терминала используем команду node:

```
$ node index.js
```