язык программирования

# JavaScript

# Сегодня в программе

- глобальный объект
- функции
- массивы и другие коллекции
- даты и регулярные выражения

# Глобальный объект

*Some of my biggest regrets!*

Brendan Eich

# Глобальная область видимости

```
<script>
  let name = "Roman";
</script>

<script>
  console.log(name); // "Roman"
</script>
```

# Локальная область видимости

```
<script>
  (function () {
      let name = "Roman";
  }());
</script>
```

Избегайте глобальных переменных.

## Модули ECMAScript

```
<script type="module">
  let name = "Roman";
</script>
```

# Глобальный объект

```
<script>
  Number();

  window.Number();
</script>
```

Браузер

```
window
```

Node.js

```
global
```

## Свойства глобального объекта

- Конструкторы
- Ошибки
- Экранирование
- Работа с числами
- Коллекции

## Функции-конструкторы

```
Array                    Number
Boolean                  Object
Date                     RegExp
Function                 String
```

## 3 роли конструкторов

- Создание объекта
- Приведение типа
- Пространство имён

# Создание объекта

```
// массив заданного размера
let predefinedSize = new Array(31);

// выражение на основе переменных
let expr = new RegExp(`${prefix}`);
```

Однако, в большинстве случаев лучше использовать литерал, а не конструктор.

# Приведение типа

```
String(5); // '5'

Number('30'); // 30

Boolean([]); // true
```

# Пространства имён

```
Array.isArray([]);

String.fromCharCode('48');

Number.MAX_SAFE_INTEGER;
```

Используйте REPL, чтобы изучить свойства того или иного пространства имён.

# Конструкторы ошибок

```
Error                    SyntaxError
EvalError                TypeError
RangeError               URIError
ReferenceError
```

# Обработка конкретных видов ошибок

```
try {
  JSON.parse('Invalid JSON');
} catch (err) {
  if (err instanceof SyntaxError) {
    // handle error
  }
  throw err;
}
```

# Функции для экранирования

```
decodeURI
decodeURIComponent
encodeURI
encodeURIComponent
```

Эти функции пригодятся, если вы будете самостоятельно формировать URL.

```
> encodeURI('Hello world')
"Hello%20world"
```

# Функции для работы с числами

```
isFinite
isNaN
parseFloat
parseInt
```

# Коллекции

```
Map
Set
WeakMap
WeakSet
```

# Типизированные массивы

```
ArrayBuffer
DataView
Float32Array
Float64Array
Int16Array
Int32Array
Int8Array
Uint16Array
Uint32Array
Uint8Array
Uint8ClampedArray
```

# Консоль

```
console.log
console.error
console.time
console.timeEnd
```

# Измерение времени

```
> console.time('timer')
undefined
> console.timeEnd('timer')
timer: 6384.525ms
```

# Функции

# Function Expression

```
let greet = function (whom) {
    return `Hello ${whom}`;
};

greet('world'); // Hello world
```

# Named Function Expression

```
let count = function step(number) {
    console.log(number);

    if (number > 0) {
        step(number - 1);
    }
};
```

# Function Declaration

```
function greet(whom) {
    return `Hello ${whom}`;
}

greet('world');
```

# Hoisting (Всплытие)

```
greet('world'); // Hello world

function greet(whom) {
    return `Hello ${whom}`;
}
```

Всплытие характерно только для Function Declaration. Для Function Expression вызывать функцию можно только после объявления.

# Arrow Functions

```
let greet = (whom) => {
    return `Hello ${whom}`;
};

let greet = whom => `Hello ${whom}`;
```

# Применение стрелочных функций

```
let us = ['me', 'you']

us.map(whom => `Hello ${whom}`);
// ['Hello me', 'Hello you']
```

# Возврат объекта

```
let createPerson = () => ({
    name: 'Roman',
    age: 28
});
```

Если стрелочная функция возвращает объект, фигурные скобки нужно обернуть в круглые.

# Аргументы и параметры

```
function sum(a, b) { // parameters
    return a + b;
}

sum(2, 3); // arguments
```

## Неявный возврат значения

```
function sum(a, b) {

}

sum(1, 2); // undefined
```

# Недостающие аргументы

```
function sum(a, b) {
    return a + b;
}

sum(1); // NaN (1 + undefined)
```

# Значение параметра по умолчанию

```
function sum(a, b = 2) {
    return a + b;
}

sum(1); // 3
```

# Значение по умолчанию (ES5)

```
function sum(a, b) {
    if (b === undefined) {
        b = 2;
    }
    return a + b;
}

sum(1); // 3
```

# Значение по умолчанию (ES5)

```
function sum(a, b) {
    b = b || 2;

    return a + b;
}

sum(1); // 3
```

# Значение по умолчанию (ES5)

```
function sum(a, b) {
    b = b || 2;

    return a + b;
}

sum(1, 0); // 3 😱
```

# Произвольное количество параметров

```
function sumAll(a, b, c, d, e, f) {
    ...
}

sumAll(1, 2, 3, 4, 5, 6, 7, 8, 9);
```

# Rest Operator

```
function sumAll(...numbers) {
    let total = 0;
    for (let number of numbers) {
        total += number;
    }
    return total;
}
```

# Arguments (ES5)

```
function sumAll() {
    let total = 0;

    for (let number of arguments) {
        total += number;
    }
```

# Произвольное количество аргументов

```
Math.max(1, 2, 3, 4, 5); // 5

let numbers = [1, 2, 3, 4, 5];
Math.max(numbers); // NaN
```

## Spread Operator

```
let numbers = [1, 2, 3, 4, 5];

Math.max(...numbers);
```

## Apply (ES5)

```
let numbers = [1, 2, 3, 4, 5];

Math.max.apply(null, numbers);
```

# Проблема позиционных аргументов

```
let showButton =
  function (round, animated) {
    if (round)    { ... }
    if (animated) { ... }
  };

showButton(true, false);
```

# Именованные аргументы (ES5)

```
let showButton =
  function (params) {
    if (params.round)    { ... }
    if (params.animated) { ... }
  };

showButton({ round: true,
             animated: false });
```

# Destructuring Operator

```
let showButton =
  function ({ round, animated }) {
    if (round)    { ... }
    if (animated) { ... }
  };

showButton({ round: true,
             animated: false });
```

# Default Named Arguments

```
let showButton =
  function ({ round    = false
            , animated = false }) {
    ...
  };

showButton({ round: true });
```

# Массивы

# Holes (Дырки)

```
let chars = [];

chars[0] = 'a';
chars[2] = 'c';

console.log(chars); // ['a', , 'c']
```

Такой массив называется разрежённым.

# Конструктор

```
let chars = new Array(3);
console.log(chars); // [ , , ]

chars.fill(0);
console.log(chars); // [0, 0, 0]
```

Конструктор создаёт массив с дырками. Их нужно заполнить самостоятельно.

# Удаление элемента

```
let chars = ['a', 'b', 'c'];

delete chars[2];

console.log(chars); // ['a', 'b', ]
console.log(chars.length); // 3
```

# Вырезание элемента

```
let chars = ['a', 'b', 'c'];

chars.splice(1, 2); // ['b', 'c']

console.log(chars); // ['a']
console.log(chars.length); // 1
```

# Длина массива

```
> [ 'a', 'b' ].length
2
> [ 'a', , 'b' ].length
3
```

Свойство `length` отслеживает наибольший индекс элемента массива, а не его размер.

# Изменение длины массива

```
let chars = ['a', 'b', 'c'];

chars.length = 5;
console.log(chars); // ['a', 'b', 'c'

chars.length = 1;
console.log(chars); // ['a']
```

## Методы массива

- Destructive – изменяют исходный массив
- Nondestructive – создают новый массив

# Добавление и удаление элементов

```
shift       pop       splice
unshift     push
```

Эти методы изменяют исходный массив.

# Упорядочивание элементов

```
reverse
sort
```

Эти методы изменяют исходный массив.

## Сортировка чисел

```
[1, 2, 10]
    .sort(); // [1, 10, 2]
```

# Сортировка чисел

```
let asNumbers = (a, b) => {
    if (a > b) return 1;
    if (a < b) return -1;
    return 0;
}

[1, 2, 10]
    .sort(asNumbers); // [1, 2, 10]
```

# Сортировка строк

```
['уж', 'ёж']
    .sort(); // ['уж', 'ёж']
```

# Сортировка строк

```
let asStrings = (a, b) => {
    return a.localeCompare(b);
}

['уж', 'ёж']
    .sort(asStrings); // ['ёж', 'уж']
```

# Операции над массивами

```
concat
slice
join
```

# Slice

```
let chars = ['a', 'b', 'c'];

chars.slice(0, 1); // ['a']
chars.slice(1, -1); // ['b']
chars.slice(-2); // ['b', 'c']
chars.slice(); // ['a', 'b', 'c']
```

slice удобно использовать перед вызовом мутирующего метода, чтобы не затрагивать исходный массив.

# Итерирование и трансформация

```
arr.<method>(callback);

function callback(value, index) {
    ...
}
```

Все рассматриваемые далее методы создают новый массив, а не изменяют исходный.

## ForEach

```
let fruits = ['apple', 'pear'];

fruits.forEach((elem, index) => {
    console.log(elem, index);
});
```

# Some, Every

```
let numbers = [1, 10, 2];
let result = numbers.some(x => {
    return x > 5;
});

console.log(result); // true
```

some и every – ленивые методы. Они не будут обходить все элементы, если условие уже выполнено (или уже не выполнено).

# Filter

```
let numbers = [1, -10, 2];
let positive = numbers.filter(x => {
    return x > 0;
});

console.log(positive); // [1, 2]
console.log(numbers); // [1, -10, 2]
```

# Удаление элемента по индексу

```
let numbers = [1, -10, 2];

numbers = numbers.filter((x, i) => {
    return i !== 1;
});

console.log(numbers); // [1, 2]
```

# Удаление falsy элементов

```
let numbers = [1, undefined, 2];

numbers = numbers.filter(Boolean);

console.log(numbers); // [1, 2]
```

Данный подход не сработает для удаления дырок в массиве – filter, как и остальные методы из этой группы, пропускает дырки при обходе массива.

# Map

```
let numbers = [1, 2];
let doubles = numbers.map(x => {
    return x * 2;
});

console.log(doubes); // [2, 4]
```

## Reduce

```
let xs = [1, 2, 3];
let sum = xs.reduce((prev, x) => {
    return prev + x;
});
```

# Цепочки вызовов

```
let numbers = [1, -1, 3, -3];

numbers
    .filter(x => x > 0)
    .map(x => x * 2)
    .reduce((x, y) => x + y);
```

# Коллекции

## Set

```
let set = new Set(['hello', 'world',

set.size; // 2
set.has('hello'); // true
```

# Сравнение элементов

```
let set = new Set();

set.add({ name: "Roman" });
set.add({ name: "Roman" });

set.size; // 2
```

Сравнение осуществляется эквивалентно оператору ===, за исключением NaN – он считается равным самому себе. Также считаются равными +0 и -0

# Преобразование в массив

```
let set = new Set(['hello', 'world',

let arr = [...set];
console.log(arr); // ['hello', 'world
```

# Map

```
let map = new Map();
let key = {};

map.set(key, 'Hello');
map.get(key); // Hello
```

## WeakMap, WeakSet

- Ключи могут быть удалены сборщиком мусора
- Ключами могут быть только объекты
- Подходят для реализации кэшей и обработчиков событий

# Date

## Date Time String

Строка в формате ISO 8601 Extended Format.

```
new Date('2017-10-17T19:48:21.684Z')
```

Не рекомендуется – результат зависит от реализации.

Node.js

```
> new Date('2017-10-17T00:00:00')
2017-10-17T00:00:00.000Z
```

Браузер

```
❯ new Date('2017-10-17T00:00:00')
2017-10-16T19:00:00.000Z
```

## Timestamp

Количество миллисекунд с 1970-01-01.

```
new Date(timestamp)
```

```
> new Date(1508270783253)
2017-10-17T20:06:23.253Z
```

## Timestamp текущего момента

```
let currentTimestamp = Date.now();

new Date(currentTimestamp);
```

## Без аргументов

Создаёт объект для текущей даты и времени.

```
new Date()
```

```
> new Date()
2017-10-17T19:59:20.068Z
```

## Отдельные значения

Создаёт объект с учётом временной зоны.

```
new Date(year, month, date?,
    hours?, minutes?, seconds?,
    milliseconds?
)
```

Месяц — в пределах от 0 (январь) до 11 (декабрь).

```
> new Date(2012, 10, 10)
2012-11-09T19:00:00.000Z
```

## Значения в UTC

```
let ts = Date.UTC(2012, 10, 10);
let utcDate = new Date(ts);
```

```
> new Date(Date.UTC(2012, 10, 10))
2012-11-10T00:00:00.000Z
```

## Методы для чтения и изменения

### Локальное время

- `get<unit>`
- `set<unit>`

### UTC

- `getUTC<unit>`
- `setUTC<unit>`

## <unit>

- `FullYear`
- `Month`
- `Date`
- `Day`
- `Hours`
- `Minutes`
- `Seconds`
- `Milliseconds`

```
> let d = new Date(2010, 10, 10)
'Wed Nov 10 2010 00:00:00 GMT+0500'
> d.getMonth()
10
> d.setUTCHours(20)
1289332800000
```

## Используйте `getFullYear`

```
> d.getYear()
110
> d.getFullYear()
2010
```

`getYear`, `setYear`, `getUTCYear`, `setUTCYear` – deprecated.

## Приведение к строке

Для компьютеров

```
> d.toISOString()
'2010-11-09T20:00:00.000Z'
```

Для людей

```
> `${d.getHours()}:${d.getMinutes()}`
'1:40'
```

## Прочие методы объекта даты

Timestamp.

```
new Date().getTime()
```

Смещение в минутах относительно UTC.

```
new Date().getTimezoneOffset()
```

# RegExp

*If it's more than two inches, rethink it. You probably want to use a different tool.*

Douglas Crockford

```
/^\d{2}:\d{2}:\d{2}$/
```

```
/^((?:(?:[^?+*{}()[\]\\|]+|\\.|\
[(?:\^?\\.|\^[^\\]|[^\\^])(?:
[^\]\\]+|\\.)*\]|\((?:\?[:=!]|\?
<[=!]|\?>)?(?1)??\)|\(\?(?:R|[+-]?
\d+)\))(?:(?:[?+*]|\{\d+(?:,\d*)?
\})[?+]?)?|\|)*)$/
```

## Конструктор

```
let name = 'Roman';
let regex =
  new RegExp(`Hello ${name}`);

let str = 'Hello Roman!'
str.replace(regex, 'Hi Pyotr');
```

## Буква «ё»

```
/\w/.test('ё');      // false
/[а-я]/.test('ё');   // false

/[а-яё]/.test('ё'); // true
```

# Множественная замена

```
'aa'.replace('a', 'A'); // 'Aa'

'aa'.replace(/a/g, 'A'); // 'AA'
```

g – глобальный флаг регулярного выражения

# Материалы

## Глобальный объект

- Global Variables
- The Global Object
- 23. Standard Global Variables
- 9.7 The global object
- ECMAScript modules in browsers

## Функции

- 15. Functions
- 4.4 From function expressions to arrow functions
- 4.5 Handling multiple return values
- 4.7 Handling parameter default values
- 4.8 Handling named parameters
- 4.9 From arguments to rest parameters

## Функции

- 4.10 From apply() to the spread operator (...)
- 9.6 Parameters as variables
- 9.8 Function declarations and class declarations
- 11. Parameter handling
- 27. Tail call optimization

## Массивы

- 18. Arrays
- 4.6 From for to forEach() to for-of
- 4.11 From concat() to the spread operator (...)
- 4.17 New Array methods
- 10. Destructuring
- 18. New Array features

## Прочее

- 19. Regular Expressions
- 20. Dates
- 19. Maps and Sets
- 20. Typed Arrays

# Конец