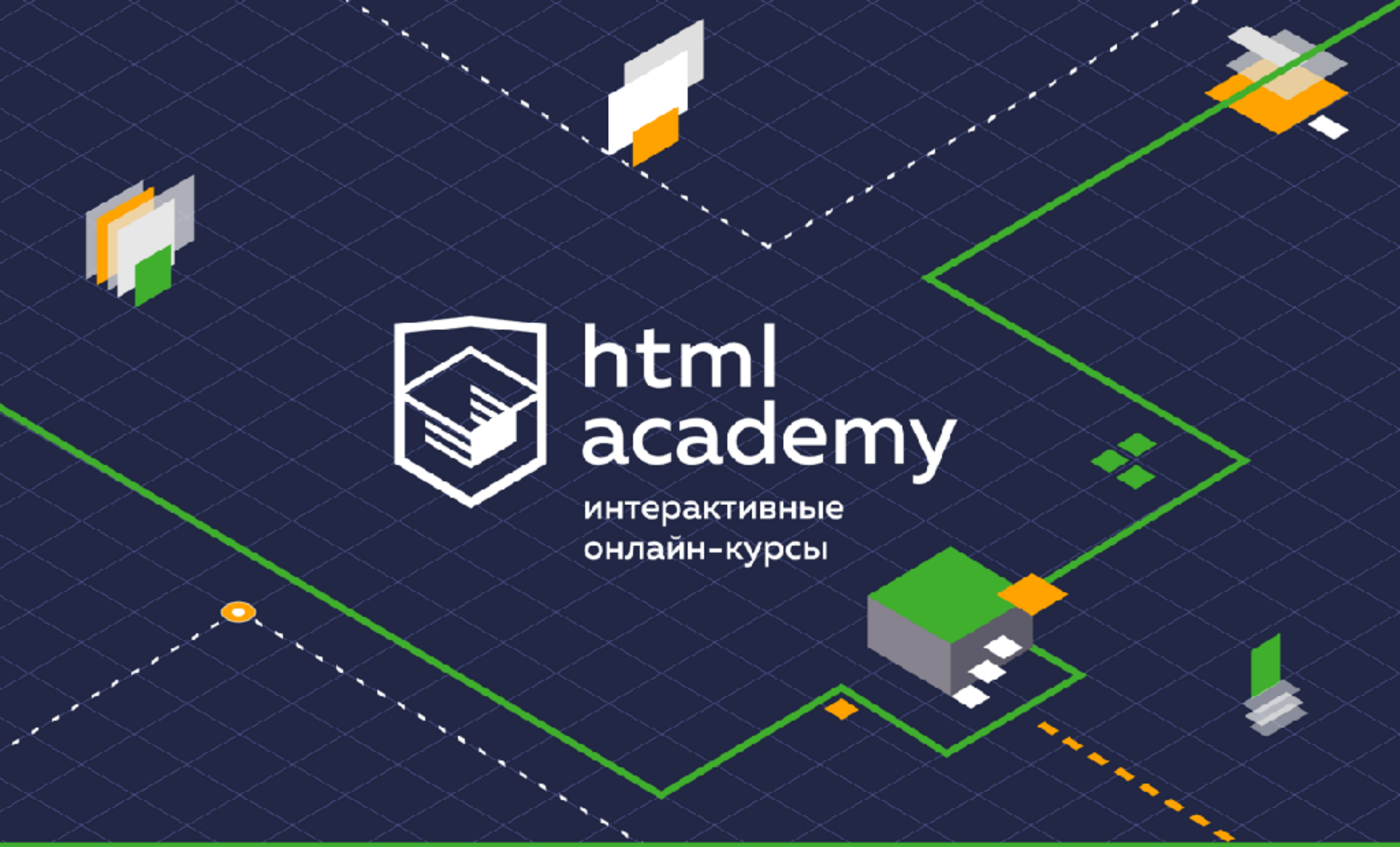


Раздел 0×08: Метапрограммирование

Unicode

Универсальная и интернациональная таблица кодировки символов

en.wikipedia.org/wiki/Unicode

ECMAScript 5 Unicode

```
const symbol = 'Й';

console.info(symbol.length); // 1. ?

const emoji = '😣';

console.info(emoji.length);  // 2. ?
```

mathiasbynens.be/notes/javascript-encoding

ECMAScript 5 Unicode

```
let symbol = 'Й';
// UCS-2 (65 535 символов)
symbol = '\u0439';
console.info(symbol.length); // 1

let emoji = '😣';
// UTF-16 (1 112 064 символов)
emoji = '\uD83D\uDE23';       // DEC 128 547
console.info(emoji.length);  // 2
```

mathiasbynens.be/notes/javascript-encoding

ECMAScript 2015 Unicode

```
let emoji = '😄';

// New UTF notation
emoji = '\u{1F604}';
console.info(emoji.codePointAt(0)); // 128 516
console.info(String.fromCodePoint(128516) === emoji); // true

console.info(emoji);          // 😄
console.info(emoji.length); // 2
```

mathiasbynens.be/notes/javascript-unicode

Регулярные выражения

RegExp

Флаг «U» (Unicode)

```
assert.notOk = (value) => assert.ok(!value);

// поиск по unicode символу
assert.ok(/\u0075/.test('u'));

// поиск по всем символам
assert.equal('≣'.match(/./g).length, 2);
assert.equal('\ud834\udf06', '≣');
assert.equal('\u{1d306}', '≣');

// поиск по количеству символов
assert.notOk(/\u{75}/.test('u'));
assert.notOk(/\u{1d306}/.test('≣'));
assert.notOk(/\u{a}/.test('uuuuuuuuuu'));

assert.ok(/\u{2}/.test('uu'));
assert.notOk(/\u{2}/.test('u'));
```

https://mathiasbynens.be/notes/es6-unicode-regex

Флаг «U» (Unicode)

```
assert.notOk = (value) => assert.ok(!value);

// поиск по всем юникод символам
assert.equal('≣'.match(/./ug).length, 1);

// поиск по юникоду в новом формате
assert.ok(/\u{75}/u.test('u'));
assert.ok(/\u{1d306}/u.test('≣'));

assert.notOk(/\u{2}/u.test('uu'));
```

Флаг «Y» (Sticky)

```
assert.notOk = (value) => assert.ok(!value);

// поиск по всем юникод символам
assert.equal('≣'.match(/./ug).length, 1);

// поиск по юникоду в новом формате
assert.ok(/\u{75}/u.test('u'));
assert.ok(/\u{1d306}/u.test('≣'));

assert.notOk(/\u{2}/u.test('uu'));
```

Где нужно

- Поиск и анализ текста
- Извлечение эмоций 😎🤗

Новые встроенные методы

Строковые (String)

```
assert.equal('.'.repeat(3), '...');

assert.ok('hello'.startsWith('hell'));
assert.ok('hello'.startsWith("ello", 1));

assert.ok('hello'.endsWith('ello'));
assert.ok('hello'.endsWith('hell', 4));
```

```
assert.ok('hello'.includes('ll'));
assert.ok(!('hello'.includes('ell', 2)));

for(const char of 'Hi, ☃!') { console.log(char); }

// H
// i
// ,
//
// ☃
// !
```

Число (Number)

```
assert(Number.isNaN(42) === false);
assert(Number.isNaN(NaN) === true);

assert(Number.isNaN('abcd') === false);
assert(isNaN('abcd') === true);

assert(Number.isFinite(Infinity) === false);
assert(Number.isFinite(42) === true);

assert(Number.isFinite('0') === false);
assert(isFinite('0') === true);

assert(Number.isSafeInteger(42) === true);
assert(Number.isSafeInteger(9007199254740992) === false);

assert(0.1 + 0.2 !== 0.3);
assert(Math.abs((0.1 + 0.2) - 0.3) < Number.EPSILON);

console.log(Math.trunc(42.7)); // 42
console.log(Math.trunc( 0.1)); // 0

console.log(Math.sign(7));   // 1
console.log(Math.sign(-7));  // -1
console.log(Math.sign('a')); // NaN
```

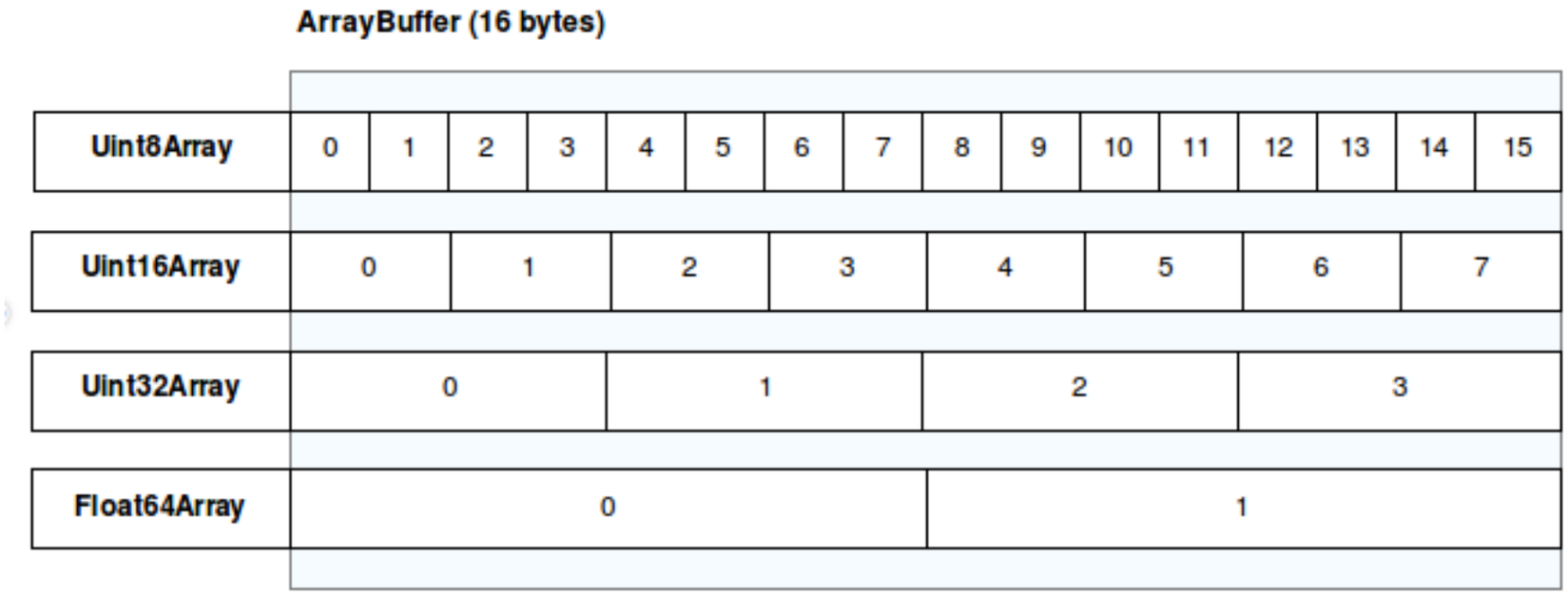
Типизированный массив

Типизированный массив (TypedArray)

```
//  ES6 class equivalent to the following C structure:
//  struct Example { unsigned long id; char username[16]; float amountDue }
class Example {
  constructor (buffer = new ArrayBuffer(24)) {
    this.buffer = buffer
  }
  set buffer (buffer) {
    this._buffer    = buffer;
    this._id        = new Uint32Array (this._buffer,  0,  1);
    this._username  = new Uint8Array  (this._buffer,  4, 16);
    this._amountDue = new Float32Array(this._buffer, 20,  1);
  }
  get buffer ()     { return this._buffer       }
  set id (v)        { this._id[0] = v           }
  get id ()         { return this._id[0]        }
  set username (v)  { this._username[0] = v     }
  get username ()   { return this._username[0]  }
  set amountDue (v) { this._amountDue[0] = v    }
  get amountDue ()  { return this._amountDue[0] }
}

let example = new Example();
example.id = 7;
example.username = 'John Doe';
example.amountDue = 42.0;
```

Типизированный массив (TypedArray)



Поддержка

- File API

Поддержка

- File API
- XMLHttpRequest

Поддержка

- File API
- XMLHttpRequest
- Fetch API

Поддержка

- File API
- XMLHttpRequest
- Fetch API
- Canvas

Поддержка

- File API
- XMLHttpRequest
- Fetch API
- Canvas
- WebSocket

Поддержка

- File API
- XMLHttpRequest
- Fetch API
- Canvas
- WebSocket
- Audio/Video, WebGL

Область применения

- Обработка аудио/видео поточных данных

Область применения

- Обработка аудио/видео поточных данных
- Объемные структуры данных

Область применения

- Обработка аудио/видео поточных данных
- Объемные структуры данных
- Эффективные структуры данных

Область применения

- Обработка аудио/видео поточных данных
- Объемные структуры данных
- Эффективные структуры данных
- Бинарный протокол общения с сервером

Internationalization (i18n) & Localization (l10n)

Date/Time Formatting

```
const l10nEN = new Intl.DateTimeFormat('en-US');
const l10nDE = new Intl.DateTimeFormat('de-DE');
const l10nRU = new Intl.DateTimeFormat('ru-RU');

const someday = new Date('1-2-2016'); // ISO 8601

console.log(l10nEN.format(someday)); // 1/2/2016
console.log(l10nDE.format(someday)); // 2.1.2016
console.log(l10nRU.format(someday)); // 02.01.2016
```

Collator

```
// in German,  "ä" sorts with "a"
// in Swedish, "ä" sorts after "z"
const list = ['ä', 'a', 'z'];
const l10nDE = new Intl.Collator('de');
const l10nSV = new Intl.Collator('sv');

console.log(l10nDE.compare('ä', 'z') === -1);
console.log(l10nSV.compare('ä', 'z') === +1);

console.log(list.sort(l10nDE.compare)); // [ 'a', 'ä', 'z' ]
console.log(list.sort(l10nSV.compare)); // [ 'a', 'z', 'ä' ]
```

Number formatting

```
const l10nEN = new Intl.NumberFormat('en-US');
const l10nDE = new Intl.NumberFormat('de-DE');
const l10nRU = new Intl.NumberFormat('ru-RU');

console.log(l10nEN.format(1234567.89)); // 1,234,567.89
console.log(l10nDE.format(1234567.89)); // 1.234.567,89
console.log(l10nRU.format(1234567.89)); // 1 234 567,89
```

Currency formatter

```
const l10nUSD = new Intl.NumberFormat('en-US',
  {style: 'currency', currency: 'USD'});
const l10nRUB = new Intl.NumberFormat('ru-RU',
  {style: 'currency', currency: 'RUB'});
const l10nEUR = new Intl.NumberFormat('de-DE',
  {style: 'currency', currency: 'EUR'});


console.log(l10nUSD.format(100200300.40)); // $100,200,300.40
console.log(l10nRUB.format(100200300.40)); // 100 200 300,40 ₽
console.log(l10nEUR.format(100200300.40)); // 100.200.300,40 €
```

Метапрограммирование

Программирование программ

Метапрограммирование

- Кодогенерация (*eval*)

Метапрограммирование

- Кодогенерация (*eval*)
- Анализ (*Reflect*, *typeof)*

Метапрограммирование

- Кодогенерация (*eval*)
- Анализ (*Reflect, typeof)*
- Модификация (*Reflect.defineProperty*)

Метапрограммирование

- Кодогенерация (*eval*)
- Анализ (*Reflect, typeof)*
- Модификация (*Reflect.defineProperty*)
- Трансформация (*get/set, Proxy*)

Метапрограммирование

- Кодогенерация (*eval*)
- Анализ (*Reflect, typeof)*
- Модификация (*Reflect.defineProperty*)
- Трансформация (*get/set, Proxy*)
- Расширение (*Symbol*)

Тип Symbol

developer.mozilla.org/en-US/docs/Glossary/Symbol

Новый тип

```
console.log(Symbol('foo') !== Symbol('foo'));

const foo = Symbol();
const bar = Symbol();

console.log(Symbol('mySymbol') == Symbol('mySymbol'));
console.log(typeof foo === 'symbol');
console.log(typeof bar === 'symbol');
```

```
const obj = {};
obj[foo] = 'foo';
obj[bar] = 'bar';
console.log(JSON.stringify(obj)); // {}
obj.bar = 'baz';
console.log(JSON.stringify(obj)); // {}
console.log(obj.bar);
console.log(obj[bar]);

Object.keys(obj) ;// []
Object.getOwnPropertyNames(obj); // []
Object.getOwnPropertySymbols(obj); // [ foo, bar ]
```

Глобальные символы

```
console.log(Symbol.for('app.foo') === Symbol.for('app.foo'));

const foo = Symbol.for('app.foo');
const bar = Symbol('app.bar');

console.log(Symbol.keyFor(foo)); // "app.foo"
console.log(Symbol.keyFor(bar)); // undefined

console.log(typeof foo === 'symbol');
console.log(typeof bar === 'symbol');

const obj = {};
obj[foo] = 'foo';
obj[bar] = 'bar';

JSON.stringify(obj); // {}

Object.keys(obj); // []
Object.getOwnPropertyNames(obj); // []
console.log(Object.getOwnPropertySymbols(obj)); // [ foo, bar ]
```

Для чего нужны символы

- Способ скрыть (инкапсулировать) данные

Для чего нужны символы

- Способ скрыть (инкапсулировать) данные
- По-настоящему уникальный набор значений (enum)

Для чего нужны символы

- Способ скрыть (инкапсулировать) данные
- По-настоящему уникальный набор значений (enum)
- Переопределять встроенное поведение

Встроенные символы

```
Symbol.hasInstance
Symbol.isConcatSpreadable
Symbol.iterator
Symbol.match
Symbol.prototype
Symbol.replace
Symbol.search
Symbol.species
Symbol.split
Symbol.toPrimitive
Symbol.toStringTag
Symbol.unscopables
```

Наследование от встроенных классов

```
class MyArray extends Array {
  constructor(...args) {
    super(...args);
  }

}
const arr = new MyArray(1, 2, 3);
const mapped = arr.map(x => x * x);

console.log(arr instanceof MyArray); // true
console.log(mapped instanceof MyArray);   // true
```

Наследование от встроенных классов

```
class MyArray extends Array {
  constructor(...args) {
    console.log('Constructor call');
    super(...args);
  }

  // Overwrite species to the parent Array constructor
  static get [Symbol.species]() {
    console.log('Symbol.species');
    return Array;
  }
}
const a = new MyArray(1, 2, 3);
const mapped = a.map(x => x * x);

console.log(mapped instanceof MyArray); // false
console.log(mapped instanceof Array);   // true
```

Паттерн Итератор (Iterator)

Iterable

То по чему можно итерироваться

Iterator

тот кто знает кто следующий

Demo

Symbol.iterator

```
const fibonacci = {
  [Symbol.iterator]() {
    let pre = 0, cur = 1;
    return {
      next () {
        [pre, cur] = [cur, pre + cur];
        return {done: false, value: cur}
      }
    }
  }
};

for (const n of fibonacci) {
  if (n > 1000) break;
  console.log(n);
}
```

Generator

Generator iterator

```
const fibonacci = {
  *[Symbol.iterator]() {
    let pre = 0, cur = 1;
    for (;;) {
      [ pre, cur ] = [ cur, pre + cur ];
      yield cur
    }
  }
};

for (const n of fibonacci) {
  if (n > 1000) break;
  console.log(n);
}
```

Generator сам по себе

```
function* range(start, end, step = 1) {
  while (start < end) {
    yield start;
    start += step;
  }
}

for (const i of range(0, 10, 2)) {
  console.log(i); // 0, 2, 4, 6, 8
}

console.log([...range(0, 5)]); // [ 0, 1, 2, 3, 4 ]
```

Generator использование

```
const fibonacci = function*(numbers) {
  let pre = 0, cur = 1;
  while (numbers-- > 0) {
    [pre, cur] = [cur, pre + cur];
    yield cur;
  }
};
```

```
for (const n of fibonacci(1000)) {
  console.log(n);
}

const numbers = [...fibonacci(1000)];

const [ n1, n2, n3, ...others ] = fibonacci(1000);
```

Использование генераторов

Использование генераторов

- Ленивая загрузка с сервера

Использование генераторов

- Ленивая загрузка с сервера
- Генерация данных

Использование генераторов

- Ленивая загрузка с сервера
- Генерация данных
- Выстраивание последовательности операций

Proxy and Reflect

Reflect

Reflect.apply() === Function.prototype.apply()

Reflect.construct() === new target(...args)

Reflect.defineProperty() === Object.defineProperty()

Reflect.deleteProperty() === delete target[name]

Reflect.get() === target[propertyKey]

Reflect.set() === target[propertyKey] = value

Reflect.has() === The in operator as function

Reflect.getOwnPropertyDescriptor() === Object.getOwnPropertyDescriptor()

Reflect.isExtensible() === Object.isExtensible()

Reflect.ownKeys() === Object.getOwnPropertyNames(target).concat(Object.getOwnPropertySymbols(target))

Reflect.preventExtensions() === Object.preventExtensions()

Reflect.setPrototypeOf() === Object.setPrototypeOf()

Reflect.getPrototypeOf() === Object.getPrototypeOf()

Reflect

- Единое пространство имён для работы с метаданными
- Расширяет старые возможности
- Добавляет новые возможности

Proxy

```
this.proxy = new Proxy(user, {
  get (receiver, name) {
    if (typeof name === 'symbol') return undefined;

    return name in receiver ? receiver[name] : `Unknown field: ${name}`;
  },
  set (target, key, value) {
    if (target[key]) {
      change(key, value);
    }
  }
});
```

Assert polyfill через *chai.assert*

```
window.assert = new Proxy(chai.assert, {
  get(target, property) {
    const value = target[property];
    if (typeof value === `function`) {
      return (...args) => {
        try {
          target[property](...args);
          console.log(`<i>Assertion: – passed</i>`);
        } catch (e) {
          console.log(e);
        }
      };
    }
    return value;
  }
});
```

Proxy хорошо подойдет

- Паттерн Адаптер
- Паттерн Заместитель
- Кэширование
- Экранирование
- Подсчет вызовов
- Создание оберток


```
PROGRAM BYEBYE
WRITE(UNIT=*, FMT=*) 'Näkemiin!'
END
```