Объекты

Самый интересный тип в JS

Кирилл Талецкий

TeachMeSkills
7 августа 2023

Что будет

- Синтаксис и особенности
- Доступ к свойствам
- Методы объекта
- Ключевое слово this
- Конструкторы
- Символы
- Преобразование к примитивам

Повторим: сколько типов данных есть в JS?

Объект

- Единственный составной (не примитивный) тип в JS
- Используется для
  - хранения коллекций
  - создания сложных структур данных и методов

Литерал объекта

- Синтаксис “литерал”

```
let obj = {}; // создаём пустой объект, присваиваем его в переменную `obj`

let shape = {
  type: 'circle',
  radius: 5,
}
```

Доступ к значениям

- Через оператор `.`

```
let shape1 = {
  type: 'circle',
  radius: 5,
};

// чтение
console.log(shape.type); // circle
console.log(shape.radius); // 5

// переприсваивание
shape.radius = 25;
console.log(shape.radius); // 25

// удаление
delete shape.radius;
console.log(shape.radius) // undefined
```

Ключи

```
let profile = {
  first_name: 'John', // упрощённый синтаксис без кавычек
  last_name: 'Doe',
  "0#%&*+-!^@internal_property": "some value", // можно использовать спецсимволы, но нужны кавычки
}
```

- Синтаксис без кавычек можно использовать, если имя подчиняется правилам наименования переменных
  - Имя не начинается с цифры и не содержит спецсимволы кроме `$` и `_`
- Спецсимволы можно использовать, если взять название переменной в кавычки

Квадратные скобки

```
let profile = {
  first_name: 'John',
  last_name: 'Doe',
  "0#%&*+-!^@internal_property": "some value",
}

console.log(profile.first_name) // упрощённый синтаксис для ключей без спецсимволов, значение можно достать через оператор `.`

console.log(profile["0#%&*+-!^@internal_property"]) // чтобы достать значение со спецсимволами, нужны квадратные скобки
console.log(profile["first_name"]) // к обычным ключам тоже можно обратиться через квадратные скобки
```

Квадратные скобки

- С помощью квадратных скобок можно “на лету” составлять ключи

```
let profile = {
  first_name: 'John',
  last_name: 'Doe',
  "0#%&*+-!^@internal_property": "some value",
}

const prefix = 'last'
const postfix = 'name'
const DELIMITER = '_'

console.log(profile[prefix + DELIMITER + postfix]) // Doe
```

Квадратные скобки
Вычисляемые свойства

- Можно динамически составить ключ, и записать по нему значение в объект

```
const keyPrefix = 'first'
const DELIMITER = '_'

let profile = {
  [keyPrefix + DELIMITER + 'name']: "John"
}

console.log(profile) // { first_name: "John" }
```

Квадратные скобки
Вычисляемые свойства

- Ещё один способ положить что-то в объект через динамический ключ

```
const keyPrefix = 'first'
const DELIMITER = '_'

let profile = {}

console.log(profile) // {}

profile['first' + '_' + 'name'] = "Венцеслав"

console.log(profile) // { first_name: "Венцеслав" }
```

Значения из переменных

- В значения объекта можно записывать значения из переменных

```
const userResponse = +prompt("What is your age?");

let profile = {
  age: userResponse
}

console.log(profile);
```

Значения из переменных

- Если название ключа и переменной совпадают, то можно воспользоваться короткой записью

```
const userResponse = +prompt("What is your age?");

let profile = {
  age: userResponse
}

console.log(profile);
```

```
const age = +prompt("What is your age?");

let profile = {
  age
}

console.log(profile);
```

Проверка существования ключа

- Оператор `in`

```
let profile = {
  firstName: "John"
}

console.log("firstName" in profile); // true

console.log("lastName" in profile); // false
console.log(profile.lastName !== undefined); // false
```

Проверка существования ключа

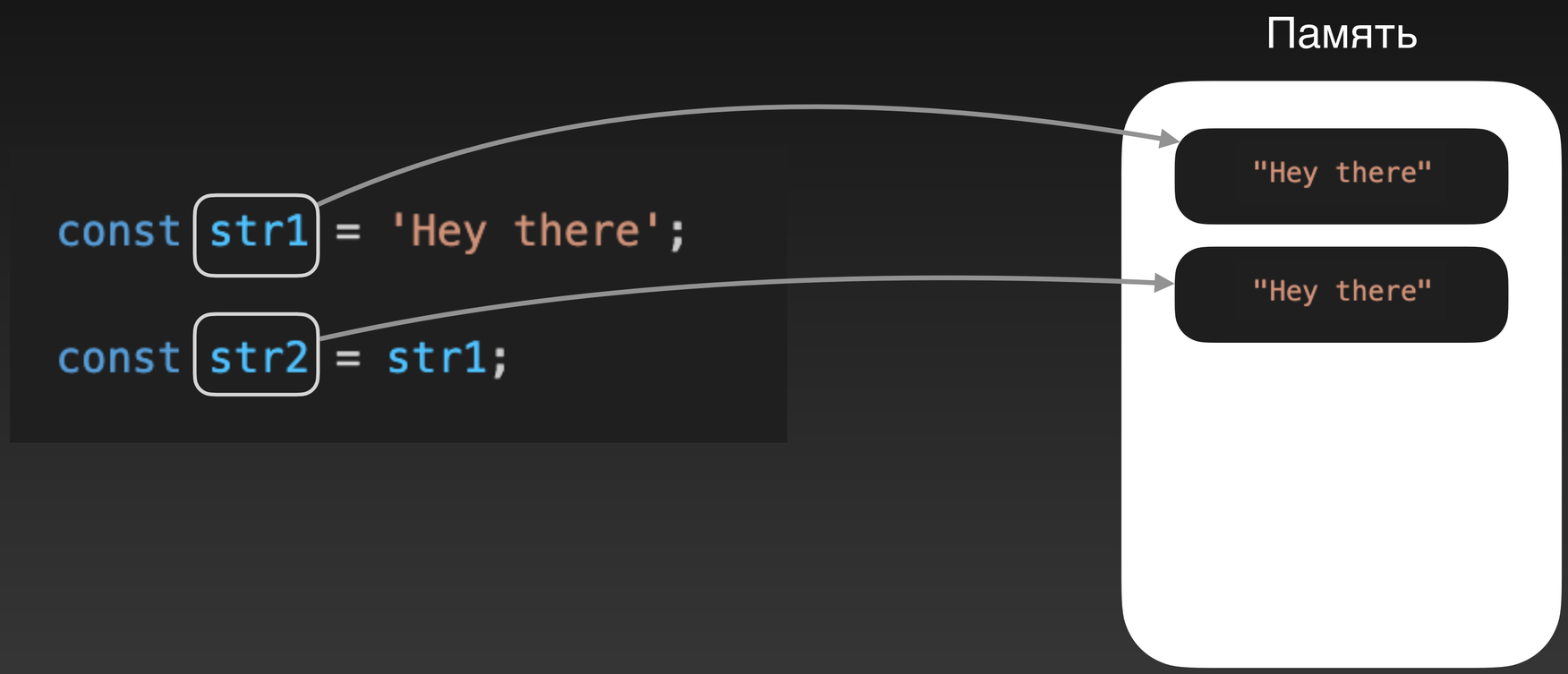
- Оператор `in` вернёт true, если свойство есть, даже если оно равно `undefined`

```
let profile = {
  firstName: "John",
  lastName: undefined,
}

console.log("lastName" in profile); // false
console.log(profile.lastName !== undefined); // true (!)
```

Копирование по ссылке

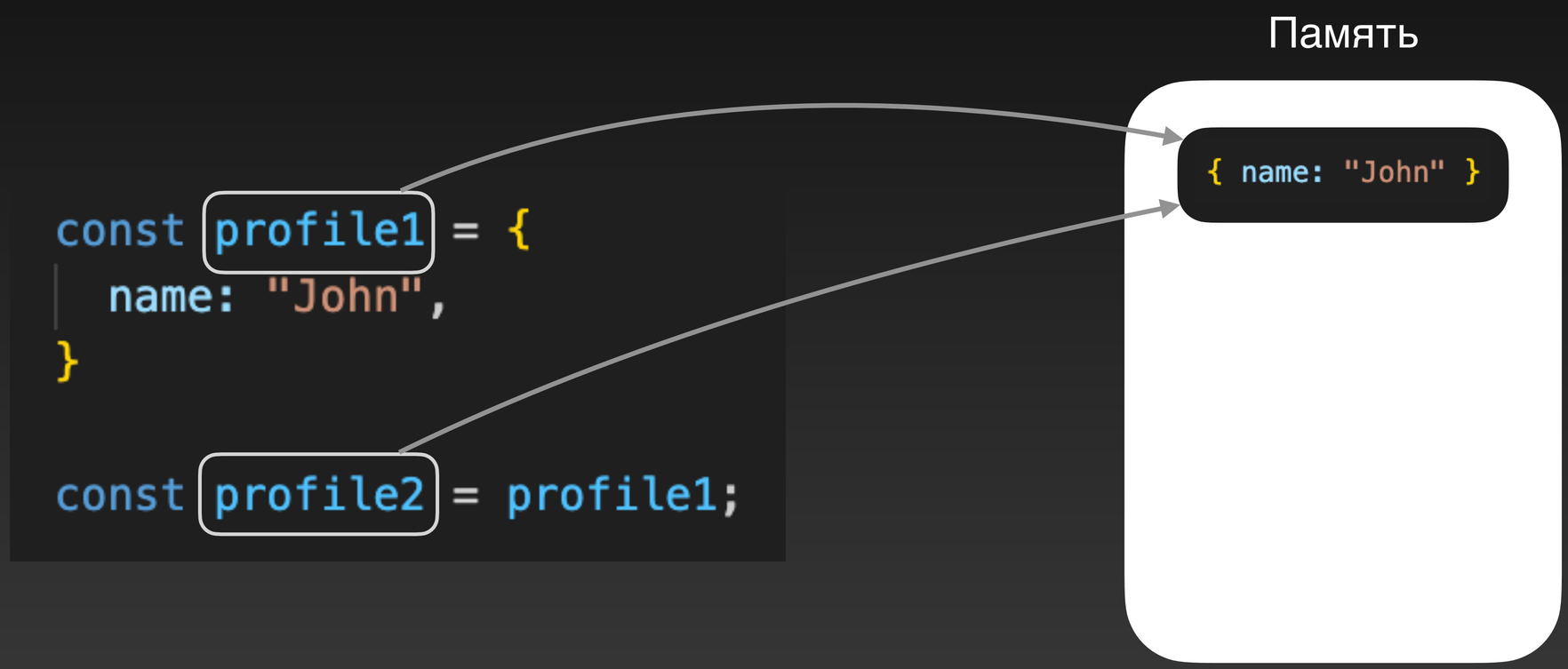
Копирование примитивов
По значению

- В JavaScript примитивы копируются “по значению”



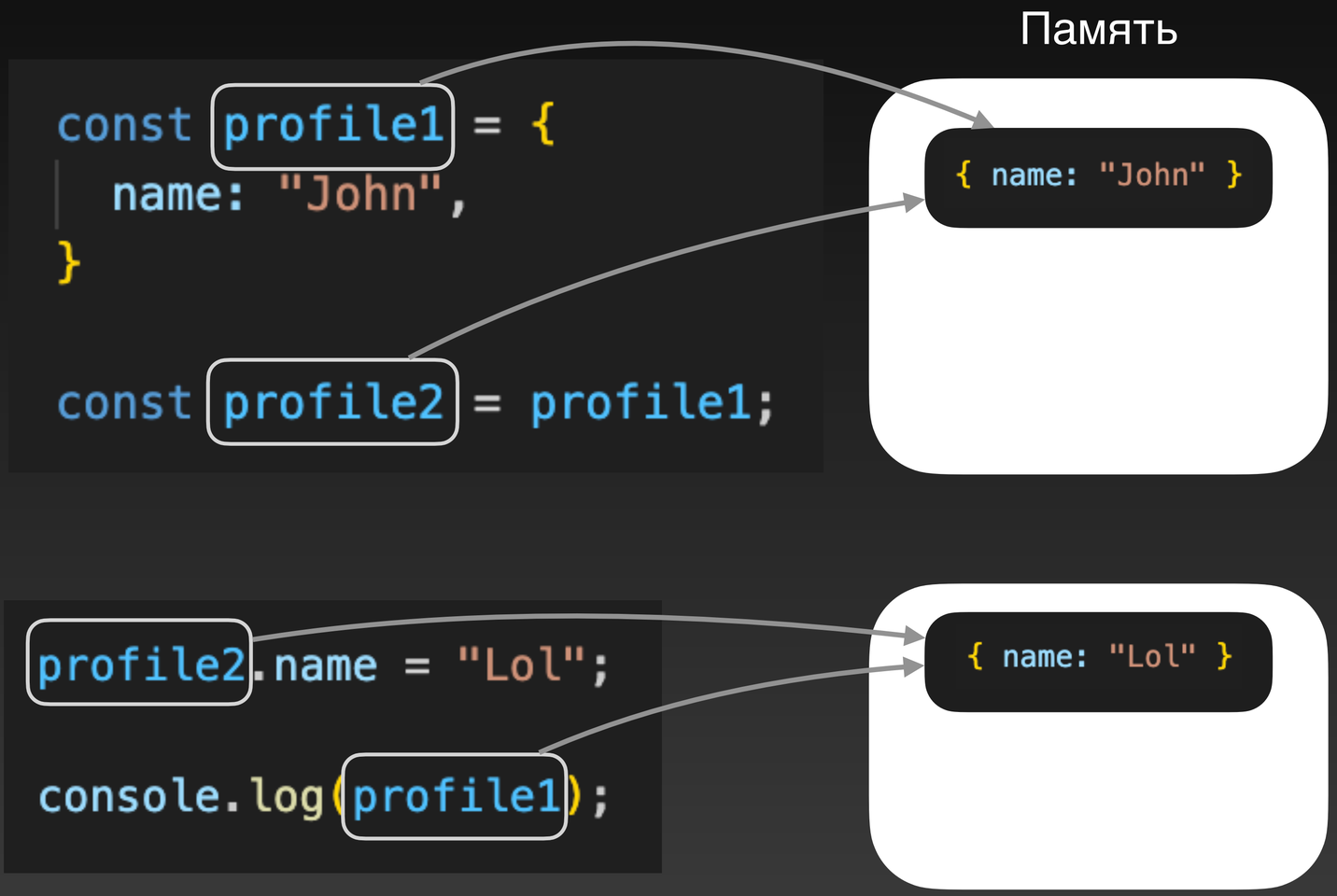
Копирование объектов
По ссылке

- Объекты копируются “по ссылке”



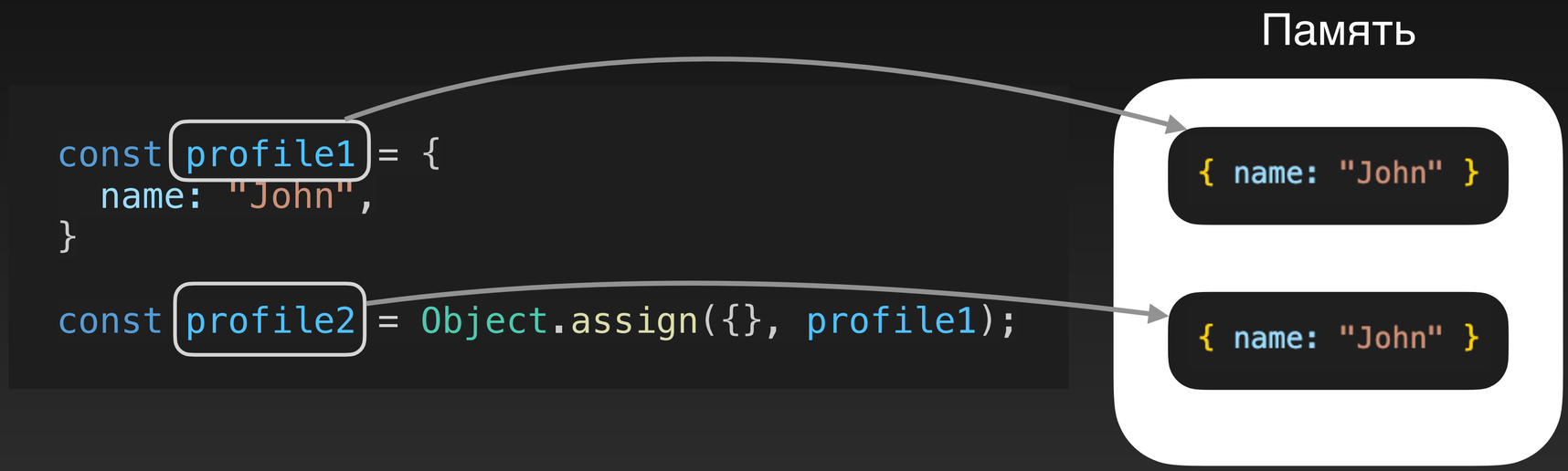
Копирование объектов

По ссылке



Копирование объектов

По значению



Присваивание через const

```
const profile = {
  firstName: "John",
  lastName: "Doe",
};

profile.firstName = "Pete"; // допустимо, ссылка на profile осталась прежней
console.log(profile); // { firstName: "Pete", lastName: "Doe" }

profile = {} // ошибка – нельзя присвоить другую ссылку в `profile`
```

- const гарантирует неизменность ссылки
- значение объекта при этом можно свободно изменять
- в JS есть возможность сделать объект полностью неизменяемым, но об этом позже

Циклы

for ... in

- Идёт по ключам объекта

```
const profile = {
  firstName: 'Lol',
  lastName: 'Kek',
  age: 25,
  bio: 'Test profile for demo',
}

for (key in profile) {
  console.log(key);
}

// firstName lastName age bio
```

Методы

Методы

- По ключу в объект можно записать любое значение, в том числе и функцию

```
const profile = {
  firstName: 'Иван',
}

profile.greet = function () {
  console.log('Привет!')
};


profile.greet() // Привет!
```

Сокращённая запись

```
const profile = {
  firstName: 'Иван',
  greet: function () {
    console.log('Привет!')
  }
}
```

```
const profile = {
  firstName: 'Иван',
  greet() {
    console.log('Привет!')
  }
}
```

This

Ключевое слово this

- В методах объекта нам хотелось бы иметь доступ к самому объекту
- В JavaScript это достигается с помощью ключевого слова `this`

```
const profile = {
  name: 'Иван',
  greet() {
    console.log(`${this.name} говорит: привет!`)
  }
}

profile.greet() // Иван говорит: привет!
```

Динамическая природа this

- this — динамический контекст выполнения функции

Динамическая природа this

- this — динамический контекст выполнения функции

```
const profile = { name: 'Иван' };
const animal = { name: 'Айдар', type: 'собакен' };

const greet = function () {
  console.log(`${this.name} говорит: привет!`);
}

profile.sayHi = greet;
animal.sayHi = greet;

profile.sayHi(); // Иван говорит: привет!
animal.sayHi(); // Айдар говорит: привет!
```

Динамическая природа this

```
const greet = function () {
  console.log(`${this.name} говорит: привет!`);
}
```

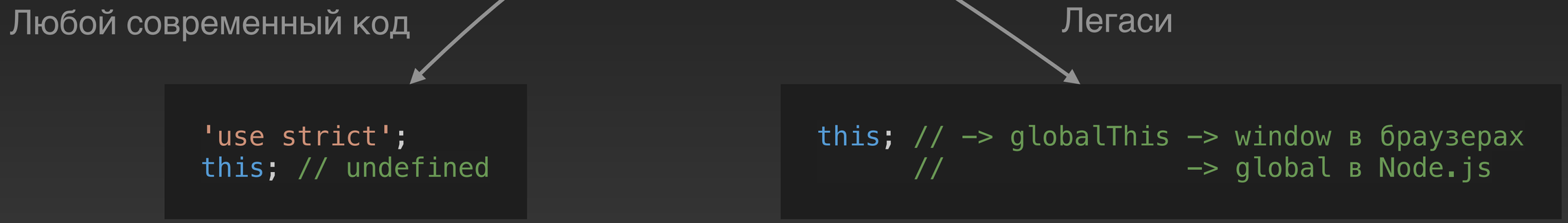
```
const profile = { name: 'Иван' };
const animal = { name: 'Айдар', type: 'собакен' };

const greet = function () {
  console.log(`${this.name} говорит: привет!`);
}

profile.sayHi = greet;
animal.sayHi = greet;

profile.sayHi(); // Иван говорит: привет!
animal.sayHi(); // Айдар говорит: привет!
```

- this — это всегда объект до `.`

Динамическая природа this

- Если никакого объекта до точки нет, то на что тогда будет ссылаться `this`?

```
const greet = function () {
  console.log(`${this.name} говорит: привет!`);
}
```



Стрелочные функции и this

- У стрелочных функций нет `this`
- Мнемоника: в стрелочной функции `this` — это просто какая-то переменная снаружи
  - Это может быть this внешней функции
  - Или это будет undefined / globalThis

```
const profile = {
  name: 'Иван',
  greet() {
    const arrow = () => console.log(${this.name});
    arrow();
  }
}

profile.greet(); // Иван
```

Вопрос с собеседования: что выведется в консоль?

```
const profile = {
  name: 'Иван',
  greet: () => {
    console.log(this.name)
  },
  greet2: function () {
    console.log(this.name)
  }
}

profile.greet(); // ?
profile.greet2(); // ?
```

Разбор для стрелочной функции

```
const profile = {
  name: 'Иван',
  greet: () => {
    console.log(this.name)
  }
}

profile.greet();
```

```
const greet = () => {
  console.log(this.name)
};

const profile = {
  name: 'Иван',
  greet,
}

profile.greet();
```

Конструкторы

Функции конструкторы

- Позволяют создавать много однотипных объектов
- Конструкторы — обычные функции, но
  - Их принято называть с большой буквы
  - Их принято вызывать только с помощью оператора new

Функции конструкторы
Пример

```
function Profile(name, status) {
  this.name = name;
  this.status = status || 'active';
}

const profile = new Profile('Иван');

console.log(profile); // { name: "Иван", status: "active" }
```

Вызов через new

1. Создаётся новый пустой объект, присваивается в `this`
2. Выполняется тело функции
3. Возвращается значение `this`

```
function Profile(name, status) {
  // (1) this = {}; (неявно)

  // (2) явно выполняется тело функции
  this.name = name;
  this.status = status || 'active';

  // (3) return this; (неявно)
}

const profile = new Profile('Иван');

console.log(profile);
// { name: "Иван", status: "active" }
```

Оператор `?.`

Проблема

- Иногда нам необходимо проверять наличие свойства перед тем, как обратиться к нему

```
let profile = {
  name: "Иван",
  avatar: {
    large: 'https://...',
    small: 'https://...',
  }
}

console.log(profile.avatar.large);

profile = null;

console.log(profile.avatar.large);
// ошибка! – нельзя прочитать свойство `avatar` у `null`
```

Решение

- Можно последовательно проверять существование каждого значения через `&&`

```
console.log(profile && profile.photo && profile.photo.large);
```

Синтаксис `?.`

- Позволяет проверить наличие значения перед обращением к ключу

```
console.log(profile?.photo?.large);
```

Синтаксис `?.`

```
console.log(profile?.photo?.large);
```

```
if (profile !== null || profile !== undefined) {
  return profile.photo
} else {
  return undefined
}
```

```
if (photo !== null || photo !== undefined) {
  return photo.large
} else {
  return undefined
}
```

Синтаксис `?.`

- Позволяет проверить наличие метода перед его вызовом

```
profile?.actions?.sayHi?.();
```

- Можно использовать с квадратными скобками

```
profile?.lol?.["--kek?"];
```

Символы

Создание символа

```
const hidden = Symbol('описание символа’);

// описание ни на что не влияет
// используется во основном для отладки
```

```
const hidden1 = Symbol('hidden');
const hidden2 = Symbol('hidden');

// два символа с одинаковым описанием – это два разных символа
console.log(hidden1 == hidden2); // false
```

- Невозможно создать два одинаковых символа

Скрытые свойства

```
const hidden = Symbol('hidden');

const profile = {
  name: 'Ivan',
  [hidden]: 'Some hidden value',
}

console.log(profile); // { name: 'Ivan' }

for (key in profile) {
  console.log(key);
}
// name

// у нас есть секретный "ключ" — символ,
// по которому можно прочитать наше секретное значение
console.log(profile[hidden]); // Some hidden value

// если мы не будем экспортировать `hidden`, то пользователи
// не будет доступа к нашему спрятанному значению
```

Системные символы

- В JavaScript есть встроенные символы
- Чаще всего, они используются для свойств, которые позволяют настраивать поведение объектов

```
Symbol.iterator
Symbol.toPrimitive
Symbol.hasInstance
```

Преобразование в примитивы

Когда значение приводится к числу?
Когда значение приводится к строке?
Когда значение приводится к булевому типу?

Правила

- В сочетании с операторами, объект всегда будет приведён к примитиву
- Так же, как и любое другое значение
  - Объект будет приведён к *числу* в математических выражениях
  - Объект будет приведён к *строке* в сложении со строкой, или, например, при выводе на экран
  - Объект будет приведён к *булевому* типу в логических операциях.
    Тут всё просто: объект — это всегда true.

Хинты

- Немного деталей: JS выделяет следующие контексты приведения к примитиву
  - `string` — объект нужно приводить к строке (вывод на экран)
  - `number` — нужно приводить к числу (мат. операции)
  - `default` — неясно, к какому типу приводить (бинарный плюс, `==`)

Алгоритм

- Чтобы выполнить преобразование, JavaScript пытается найти и вызвать три следующих метода объекта:
  - Вызвать **obj[Symbol.toPrimitive](hint)** – метод с символьным ключом Symbol.toPrimitive (системный символ), если такой метод существует,
  - Иначе, если хинт равен *"string"* попробовать вызвать **obj.toString()** или **obj.valueOf()**, смотря какой из них существует.
  - Иначе, если хинт равен *"number"* или *"default"* попробовать вызвать **obj.valueOf()** или **obj.toString()**, смотря какой из них существует.

Алгоритм

```
/**
 * Это не настоящая функция!
 * Иллюстрирует, как JS приводит объект к примитиву.
 *
 * @param {{}} obj – объект, который надо привести
 * @param {"string" | "number" | "default"} hint – JS подсказка контекста выполнения
 */
function convertToPrimitive(obj, hint) {
  if (obj[Symbol.toPrimitive]) return obj[Symbol.toPrimitive](hint);

  switch(hint) {
    case 'string':
      return obj.toString?.() || obj.valueOf?.();
    case 'number':
    case 'default':
      return obj.valueOf?.() || obj.toString?.();
  }
}
```

Приведение объекта — практика

Функции как объект