Functions часть 2

- Стрелочные функции
- Функции обратного вызова
- Немедленно вызываемые функции
- Функции конструкторы
- Замыкания
- Рекурсия
- Области видимости

TeachMeSkills

Урок

Стрелочная функция

стрелочная функция

```
const greet = () => 'Hello students!'


const greet = () => {
    return 'Hello students'
}
```

Особенности стрелочной функции

- **Стрелочные функции не содержат собственный контекст this, а используют значение this окружающего контекста.**
- **Имеют сокращенный синтаксис**
- **Не имеют собственного объекта arguments**
- **Выражение стрелочных функций не позволяют задавать имя, поэтому стрелочные функции анонимны, если их ни к чему не присвоить.**
- **Тело стрелочной функции может иметь краткую (concise body) или блочную (block body) форму. Блочная форма не возвращает значение, необходимо явно вернуть значение.**
- **Стрелочные функции не могут быть использованы как конструктор и вызовут ошибку при использовании с new**

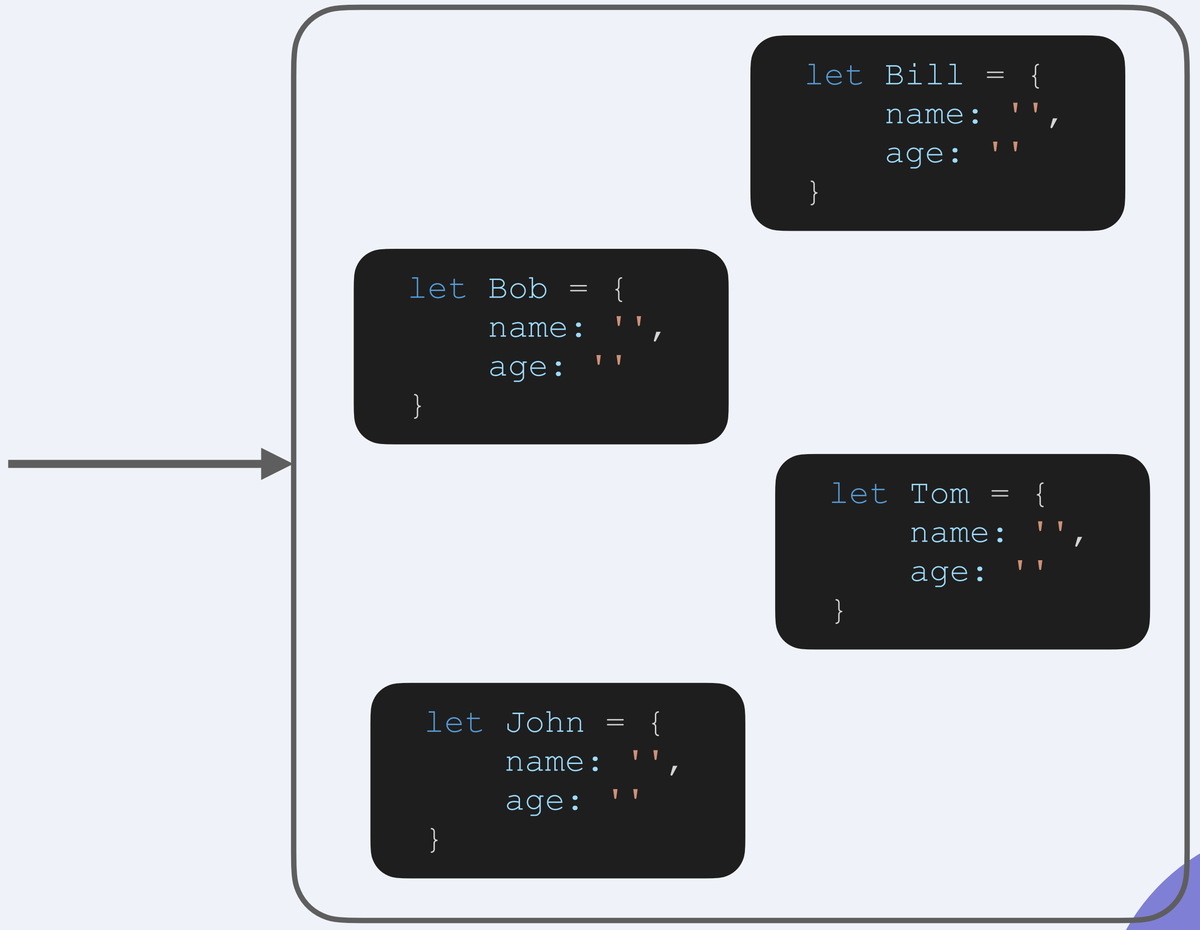
Callback функция


```
function foo(bar) {
    let name = 'Lizzy';
    bar(name);
}

foo(bar);

function bar(name) {
    console.log('Hello ' + name);
}
```

- **Функция обратного вызова - это функция, переданная в другую функцию в качестве аргумента, которая затем вызывается по завершению какого-либо действия.**

Функции конструкторы

Если нам нужно создать множество однотипных объектов мы можем воспользоваться функциями конструкторами



Функции конструкторы

Функции-конструкторы являются обычными функциями. Но есть два соглашения:

- **Имя функции-конструктора должно начинаться с большой буквы.**
- **Функция-конструктор должна вызываться при помощи оператора `"new"`**

Синтаксис

Присваиваем в объект переданные параметры

```
function Animal(name, voice){
    this.name = name;
    this.voice = voice;
}

let cat = new Animal('Cleo', 'Meaw');
let dog = new Animal('Sharif', 'Gav')
```

Оператор new создает экземпляр объекта

Под капотом движка

Создается пустой объект и присваиваем его в this

Возвращаем объект this

```
function Animal(name, voice){
    // this = {}

    this.name = name;
    this.voice = voice;

    // return this
}

let cat = new Animal('Cleo', 'Meaw');
let dog = new Animal('Sharik', 'Gav')
```

- Задача функции конструктора создать экземпляр объекта, а не возвращать явно какой-то результат

Методы

В this мы можем добавлять не только свойства, но и методы

```
function Animal(name, voice){
    this.name = name;
    this.voice = voice;

    this.getVoice = function(){
        console.log(this.voice);
    }
}

let cat = new Animal('Cleo', 'Meaw');
let dog = new Animal('Sharik', 'Gav')

cat.getVoice() // Meaw
```

Рекурсия

Рекурсия на первый взгляд сложно

Главная сложность рекурсии что ее сложно представить в голове без практики и знаний

Рекурсии помогают нам писать элегантные решения задач

Рекурсия

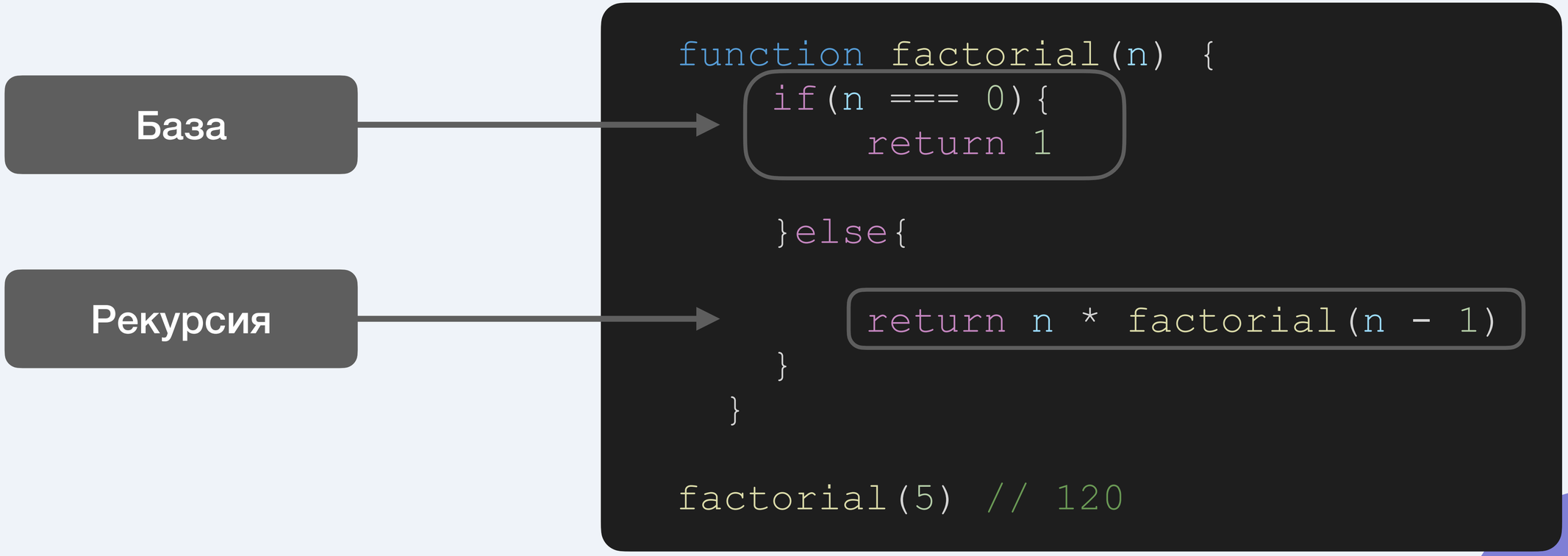
```
const factorialOf = integer => {
    let factorial = 1;

    for(let i = 1; i <= integer; i++) {
      factorial *= i;
    }

    return factorial;
  }

  factorialOf(5)
```

VS

```
const factorial = num => {
    return num ? num * factorial(num - 1) : 1;
}

factorial(5)
```

Рекурсия

В программировании рекурсия это вызов функции из неё же самой



Замыкание

Замыкание это функция у которой есть доступ к своей внешней функции по области видимости, даже после того, как внешняя функция прекратилась.

Это говорит о том, что замыкание может запоминать и получать доступ к переменным, и аргументам своей внешней функции, даже после того, как та прекратит выполнение

Замыкание это способ получения доступа и управления внешними переменными из функции.

```
function foo (){
    let name = 'Bob'
    console.log('name ' + name);
}

function boo (){
    console.log('name ' + name);
}

foo() // name Bob
boo() // name
```

```
function foo (){
    let name = 'Bob'
    console.log('name ' + name);

    function boo (){
        console.log('name ' + name);
    }

    boo() // name Bob
}

foo() // name Bob
```

Замыкание позволяет экономить на вычислениях

```
function calcTax(n) {
    let tax = n / 100
    return function(cost) {
      return cost - (cost * tax)
    };
  }

const calcPurshace = calcTax(13);

calcPurshace(100) // 87
calcPurshace(250) // 217.5
calcPurshace(843) // 733.41
```

Для самостоятельного чтения на learn.javascript.ru

Язык Javascript часть 1 → **Главы. 4.5 6.1 6.3**