Функции
this

Наталья Дружинина

Содержание

- Функции (Functions)
- Аргументы
- Область видимости
- Замыкание
- Всплытие
- this
- Контекст исполнения

Функции

3.1

Функции

- Не повторять код (DRY)
- Объединить группу действий
- Рекурсивный вызов
- Создать область видимости

Аргументы

```
function min(a, b) {
    return a < b ? a : b;
}


min(2, 7);    // 2

min(3, 4, 2); // 3

min(13);      // undefined
```

Аргументы

```
function min(a, b) {
    return a > b ? b : a;
}
```

```
min(2, 7);     // 2
min(13);       // 13
```

Аргументы

```
function min(a, b) {
    if (b === undefined) {
        return a;
    }

    return a < b ? a : b;
}


min(2, 7);     // 2
min(13);       // 13
```

Аргументы. Значения по умолчанию

```
function min(a, b = Infinity) {
    return a < b ? a : b;
}
```

```
min(2, 7);     // 2
min(13);       // 13
```

Аргументы. Значения по умолчанию

```
function log(a = 'hello') {
    console.log(a);
}

log(); // hello

log(0); // 0

log(undefined); // hello

log(null); // null
```

Именованные аргументы

```
function BMI(params) {
    const { weight, height } = params;

    return weight / (height * height);
}



BMI({ weight: 60, height: 1.7 }) // 20.7
```

Именованные аргументы. Деструктуризация

```
function BMI(params) {
    const { weight, height } = params;

    return weight / (height * height);
}



BMI({ weight: 60, height: 1.7 }) // 20.7
```

Именованные аргументы. Достоинства

- Удобно, если несколько необязательных аргументов
- Неважен порядок аргументов
- Неограниченное число аргументов
- Легко рефакторить код

Именованные аргументы. Недостатки

- Неявный интерфейс

arguments

Объект arguments - это подобный массиву объект, который содержит аргументы, переданные в функцию.

arguments MDN docs

arguments

```
function example() {
    arguments[1]; // 12
    arguments.length; // 2
}

example(3, 12);
```

arguments

```
function sum() {
    let sum = 0;

    for (let i = 0; i < arguments.length; i++) {
        sum += arguments[i];
    }

    return sum;
}

sum(); // 0
sum(2); // 2
sum(2, 4, 8);   // 14
```

arguments в стрелочных функциях

В браузере:

```
const func = () => console.log(arguments);

func(); // ReferenceError: arguments is not defined
```

arguments в стрелочных функциях

```
function argumentsTest(str) {
    console.log(arguments);

    const arrowFunc = (a, b) => {
        console.log(arguments);
    }

    arrowFunc(1, 2);
}
```

arguments в стрелочных функциях

```
function argumentsTest(str) {
    console.log(arguments); // [Arguments] { '0': 'test' }

    const arrowFunc = (a, b) => {
        console.log(arguments); // [Arguments] { '0': 'test' }
    }

    arrowFunc(1, 2);
}

argumentsTest('test');
```

Rest operator

```
function example(a, b, ...others) {
    console.log(a);
    console.log(b);
    console.log(others);
}



example(1, 2, 3, 4, 5);
// 1
// 2
// [3, 4, 5]
```

Rest operator

```
function sum(...numbers) {
    return numbers.reduce((sum, item) => {
        return sum + item;
    }, 0);
}



sum(1, 2, 3); // 6
```

Способы объявления функции

```
// function declaration
function add(a, b) {
    return a + b;
}


// function expression
const add = function (a, b) {
    return a + b;
}
```

function declaration

```
add(2, 3);


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

function declaration

```
add(2, 3); // 5


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

function expression

```
add(2, 3);


const add = function (a, b) {
    return a + b;
};
```

function expression

```
add(2, 3); // ReferenceError: add is not defined


const add = function (a, b) {
    return a + b;
};
```

Named function expression

```
const factorial = function inner(n) {
    return n === 1 ?
        1 : n * inner(n - 1);
}
```

```
typeof factorial; // 'function'
typeof inner; // undefined
factorial(3);    // 6
```

Области видимости

4.1

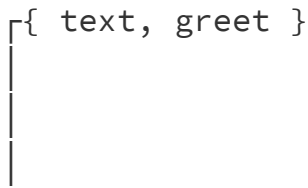
Область видимости

Код:

```
const text = 'Привет';

function greet() {}
```

Область видимости:

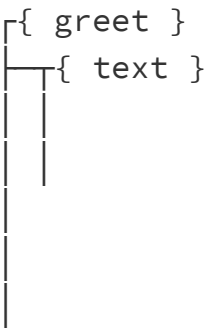


Создание области видимости

Код:

```
function greet() {
    const text = 'Привет';
    text; // 'Привет'
}
```

Область видимости:



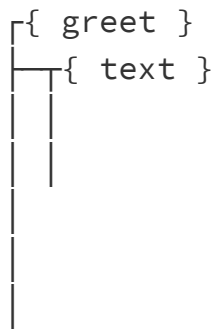
Создание области видимости

Код:

```
function greet() {
    const text = 'Привет';
    text; // 'Привет'
}

text;   // ReferenceError:
        //    text is not defined
```

Область видимости:



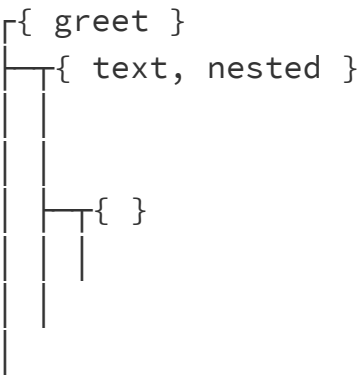
Вложенные функции

Код:

```
function greet() {
    let text = 'Привет';

    function nested() {
        text; // 'Привет'
    }
}
```

Область видимости:



Затенение

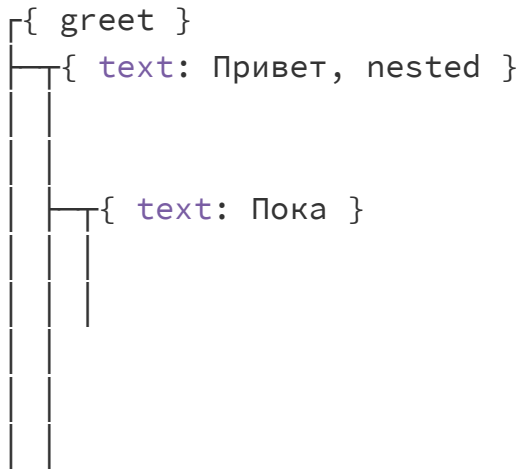
Код:

```
function greet() {
    let text = 'Привет';

    function nested() {
        let text = 'Пока';
        text; // 'Пока'
    }

    text; // 'Привет'
}
```

Область видимости:



Блочные области видимости

```
let x = 10;
var y = 10;
{
    let x = 5;

    var y = 5;
    {
        let x = 2;
        var y = 2;
        console.log(x);
        console.log(y);
    }
    console.log(x);
    console.log(y);
}
console.log(x);
console.log(y);
```

Блочные области видимости

```
let x = 10;
var y = 10;
{
    let x = 5;

    var y = 5;
    {
        let x = 2;
        var y = 2;
        console.log(x); // 2
        console.log(y); // 2
    }
    console.log(x);
    console.log(y);
}
console.log(x);
console.log(y);
```

Блочные области видимости

```
let x = 10;
var y = 10;
{
    let x = 5;

    var y = 5;
    {
        let x = 2;
        var y = 2;
        console.log(x); // 2
        console.log(y); // 2
    }
    console.log(x); // 5
    console.log(y); // 2
}
console.log(x);
console.log(y);
```

Блочные области видимости

```
let x = 10;
var y = 10;
{
    let x = 5;

    var y = 5;
    {
        let x = 2;
        var y = 2;
        console.log(x); // 2
        console.log(y); // 2
    }
    console.log(x); // 5
    console.log(y); // 2
}
console.log(x); // 10
console.log(y); // 2
```

Всплытие (Hoisting)

Выполнение кода

1. Инициализация
    1. function declaration
    2. var
2. Собственно выполнение

Всплытие функций

```
add(2, 3);  // 5


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

Всплытие переменных. var

Код:

```
add(2, 3);  // TypeError:
            // add is not a function

var add = function (a, b) {
    return a + b;
}
```

Интерпретатор:

```
var add;
add(2, 3);

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

~~Всплытие переменных~~. let/const

```
add(2, 3);  // ReferenceError: add is not defined

const add = function (a, b) {
    return a + b;
}
```

Замыкание

Замыкание

Замыкание – это функция вместе со всеми внешними переменными, которые ей доступны.

Ссылка на переменные

Код:

Счётчик ссылок:

```
function greet() {
    const text = 'Привет';
}

greet();
```

```
{ text: 1 }
```

Ссылка на переменные

Код:

```
function greet() {
    const text = 'Привет';
}

greet();
```

Счётчик ссылок:

```
{ text: 0 }
```

Ссылка на переменные

Код:

Счётчик ссылок:

```
function makeCounter() {
    let currentCount = 0;

    return function () {
        return currentCount++;
    };
}

const counter = makeCounter();
```

```
{ currentCount: 1 }
```

Ссылка на переменные

Код:

Счётчик ссылок:

```
function makeCounter() {
    let currentCount = 0;

    return function () {
        return currentCount++;
    };
}

const counter = makeCounter();
```

```
{ currentCount: 1 }
```

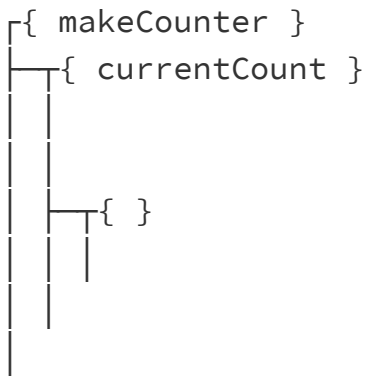
Замыкание

Код:

```
function makeCounter() {
    let currentCount = 0;

    return function () {
        return currentCount++;
    };
}
```

Область видимости:



Замыкание

```
const counter = makeCounter();
counter(); // 0

counter(); // 1
counter(); // 2

const yetAnother = makeCounter();
yetAnother(); // 0
```

Замыкание

```
function greet(name) {
    return function () {
        return `Привет, ${name}`;
    }
}

let helloWorld = greet('мир!');



helloWorld(); // "Привет, мир!"
```

Перерыв

this

this в Java

```
public class User {
    private int age = 30;

    public void showAge() {
        System.out.println(this.age);
    }
}

public static void main(String []args){
    User vasya = new User();

    vasya.showAge();
}
```

Свойства this

- ключевое слово
- указывает на текущий объект
- нельзя перезаписать

this в Javascript

```
class User () {
    constructor() {
        this.age = 24;
    }

    showAge() {
        return this.age;
    }
}

const mike = new User();

mike.showAge(); // 24
```

this за пределом объекта

В браузере

```
this.innerWidth; // 1440
```

В Node.js

```
this.process.version; // 'v10.8.0'
```

Контекст исполнения

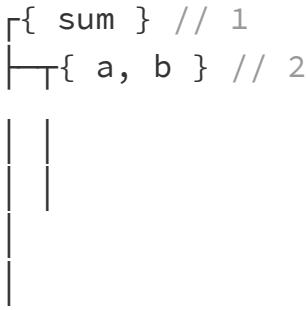
Код:

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

}

sum(1, 2);
```

Область видимости:



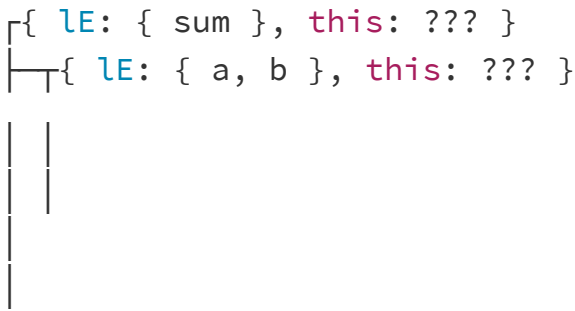
Код:

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

}

sum(1, 2);
```

Контекст исполнения:



Значение this зависит от:

1. Типа участка кода

2. Как мы попали на этот участок

3. Режима работы интерпретатора

Тип участка кода. Глобальный в браузере

```
this.innerWidth; // 1440

window.innerWidth; // 1440

innerWidth; // 1440
```

Тип участка кода. Глобальный в Node.js

```
this.process.version; // 'v10.8.0'

global.process.version; // 'v10.8.0'

process.version; // 'v10.8.0'
```

Тип участка кода. Глобальный

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

```
global.console.log('Hello!');
```

```
this.console.log('Hello!');
```

Тип участка кода. Глобальный

```
this === global; // true
```

Как попали на участок кода. Простой вызов

```
function getSelf() {
    return this;
}

getSelf(); // global
```

Как попали на участок кода. Метод объекта

Код:

Значение this:

```
const block = {
    innerHeight: 200,

    getHeight: function () {
        return this.innerHeight;
    }
}
```

???.innerHeight;

Как попали на участок кода. Метод объекта

Код:

Значение this:

```
const block = {
    innerHeight: 200,

    getHeight: function () {
        return this.innerHeight;
    }
}

block.getHeight(); // 200
```

block.innerHeight;

Как попали на участок кода. Метод объекта

Код:

Значение this:

```
const block = {
    innerHeight: 200,

    getHeight: function () {
        return this.innerHeight;
    }
}

const getHeight = block.getHeight;
```

```
???.innerHeight;
```

Как попали на участок кода. Метод объекта

Код:

Значение this:

```
const block = {
    innerHeight: 200,

    getHeight: function () {
        return this.innerHeight;
    }
}

const getHeight = block.getHeight;
getHeight();
```

```
window.innerHeight;
```

Заимствование метода

call

Метод call() вызывает функцию с указанным значением this и индивидуально предоставленными аргументами.

Function.prototype.call() - JavaScript | MDN

```
fun.call(thisArg, arg1, arg2, ...);
```

Как попали на участок кода. Заимствование метода

Код:

```
const mike = {
    age: 30,

    getAge: function () {
        return this.age;
    }
};
const jane = {
    age: 24
};
```

Значение this:

```
???.age;
```

Как попали на участок кода. Заимствование метода

Код:

```
const mike = {
    age: 30,

    getAge: function () {
        return this.age;
    }
};
const jane = {
    age: 24
};
mike.getAge.call(jane); // 24
```

Значение this:

jane.age;

apply

Метод apply() вызывает функцию с указанным значением this и аргументами, предоставленными в виде массива.

Function.prototype.apply() - JavaScript | MDN

```
fun.apply(thisArg, [arg1, arg2]);
```

apply

```
Math.min(4, 7, 2, 9); // 2
```

```
const numbers = [4, 7, 2, 9];
Math.min(arr); // NaN

Math.min.apply(Math, numbers); // 2

Math.min.apply(null, numbers); // 2
```

Spread operator

```
const numbers = [4, 7, 2, 9];

Math.min(...numbers); // 2
```

Как попали на участок кода. Callback

```
const person = {
    name: 'Jack',
    items: ['keys', 'phone', 'banana'],

    showItems: function () {
        return this.items.map(function (item) {     ???.items
            return `${this.name} has ${item}`;      ???.name
        });
    }
}
```

Как попали на участок кода. Callback

```
const person = {
    name: 'Jack',
    items: ['keys', 'phone', 'banana'],

    showItems: function () {
        return this.items.map(function (item) {
            return `${this.name} has ${item}`;
        });
    }
}

person.showItems();
```

person.items
???.name

Как попали на участок кода. Callback

```
const person = {
    name: 'Jack',
    items: ['keys', 'phone', 'banana'],

    showItems: function () {
        return this.items.map(function (item) {   person.items
            return `${this.name} has ${item}`;    global.name
        });
    }
}

person.showItems();
```

Результат

```
'undefined has keys'
'undefined has phone'
'undefined has banana'
```

Как попали на участок кода. Стрелочные функции

```
const person = {
    name: 'Jack',
    items: ['keys', 'phone', 'banana'],

    showItems: function () {
        return this.items.map(item => {
            return `${this.name} has ${item}`;
        });
    }
}

person.showItems();
```

```
person.items
person.name
```

Результат

```
'Jack has keys'
'Jack has phone'
'Jack has banana'
```

Как попали на участок кода. Callback

```
const person = {
    name: 'Jack',
    items: ['keys', 'phone', 'banana'],

    showItems: function () {
        return this.items.map(function (item) {
            return `${this.name} has ${item}`;
        }, this);
    }
}

person.showItems();
```

person.items
person.name

bind

Метод bind() создаёт новую функцию, которая при вызове устанавливает в качестве контекста выполнения this предоставленное значение. <...>

Function.prototype.bind() - JavaScript | MDN

```
fun.bind(thisArg, arg1, arg2, ...);
```

Как попали на участок кода. Callback

```
const person = {
    name: 'Jack',
    items: ['keys', 'phone', 'banana'],

    showItems: function () {
        return this.items.map(function (item) {     ???.items
            return this.name + ' has ' + item;      ???.name
        }.bind(this));
    }
}
```

Как попали на участок кода. Callback

```
const person = {
    name: 'Jack',
    items: ['keys', 'phone', 'banana'],

    showItems: function () {
        return this.items.map(function (item) {
            return this.name + ' has ' + item;
        }.bind(this));
    }
}

person.showItems();
```

person.items
person.name

Частичное применение

```
Math.pow(2, 3);  // 8
Math.pow(2, 10); // 1024


const binPow = Math.pow.bind(null, 2);

binPow(3);  // 8
binPow(10); // 1024
```

Режим работы интерпретатора. Режим совместимости

```
function getSelf() {
    return this;
}

getSelf(); // global
```

Режим работы интерпретатора. Строгий режим

```
function getSelf() {
    'use strict';

    return this;
}

getSelf(); // undefined
```

Вопросы?