

Функции и контекст

Тошлин Сергей

План лекции

- Функции и их особенности
- Аргументы и параметры
- Область видимости
- this

Объявления функции

Объявления функции

1. Function Declaration

```
function greet(name) {  
    return `Hello ${name}`;  
}  
  
greet('Sergey'); // Hello Sergey
```

Обычное объявление, можно вызвать до объявления

Объявления функции

1. Function Declaration

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

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

Всплытие (Hoisting)

Обычное объявление, можно вызвать до объявления

Объявления функции

2. Function expression

```
const greet = function (name) {  
    return `Hello ${name}`;  
};
```

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

Функция присваивается переменной, описывается без имени, нельзя вызвать до объявления

Объявления функции

3. Named function expression

```
const count = function step(number) {  
    if (number > 0) {  
        step(number - 1);  
    }  
};  
  
count(5);
```

Имя функции доступно только изнутри, имя переменной доступно везде

Объявления функции

4. Стрелочные функции

```
const greet = (name) => {  
    return `Hello ${name}`;  
};
```

```
const greet = (name) => `Hello ${name}`;
```

```
const greet = name => `Hello ${name}`;
```

Особенности функций

Особенности функций

1. Может возвращать или не возвращать значение
(будет неявно возвращать `undefined`)

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

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

Особенности функций

1. Может возвращать или не возвращать значение
(будет неявно возвращать `undefined`)

```
function calcSum(a, b) {  
    const s = a + b;  
  
    return;  
}  
  
calcSum(2, 3); // undefined
```

Особенности функций

1. Может возвращать или не возвращать значение
(будет неявно возвращать `undefined`)

```
function calcSum(a, b) {  
    const s = a + b;  
}  
  
calcSum(2, 3); // undefined
```

Особенности функций

2. Являются объектами высшего порядка

Могут быть переданы в другие функции в качестве аргумента, могут бытьозвращены из других функций в качестве результата выполнения, а также могут иметь личные свойства, как и другие объекты

Особенности функций

2. Являются объектами высшего порядка

```
// Принимаем функцию в функции, как параметр  
  
function modifyItems(items, modify) {  
    return items.map(item => modify(item));  
}  
  
const numbers = [1, 2, 3, 4];  
  
  
const multiplyBy2 = item => item * 2;  
const doubleValues = modifyItems(numbers, multiplyBy2);  
// [2, 4, 6, 8]  
  
  
const squared = item => item ** 2;  
const squaredValues = modifyItems(numbers, squared);
```

Особенности функций

2. Являются объектами высшего порядка

```
// Принимаем функцию в функции, как параметр  
  
function modifyItems(items, modify) {  
    return items.map(item => modify(item));  
}  
  
const numbers = [1, 2, 3, 4];  
  
  
const multiplyBy2 = item => item * 2;  
const doubleValues = modifyItems(numbers, multiplyBy2);  
// [2, 4, 6, 8]  
  
  
const squared = item => item ** 2;  
const squaredValues = modifyItems(numbers, squared);
```

Особенности функций

2. Являются объектами высшего порядка

```
// Принимаем функцию в функции, как параметр  
  
function modifyItems(items, modify) {  
    return items.map(item => modify(item));  
}  
  
const numbers = [1, 2, 3, 4];  
  
const multiplyBy2 = item => item * 2;  
const doubleValues = modifyItems(numbers, multiplyBy2);  
// [2, 4, 6, 8]  
  
const squared = item => item ** 2;  
const squaredValues = modifyItems(numbers, squared);
```

Особенности функций

2. Являются объектами высшего порядка

```
// Принимаем функцию в функции, как параметр  
  
function modifyItems(items, modify) {  
    return items.map(modify); // .map(item => modify(item))  
}  
  
const numbers = [1, 2, 3, 4];  
  
  
const multiplyBy2 = item => item * 2;  
const doubleValues = modifyItems(numbers, multiplyBy2);  
// [2, 4, 6, 8]  
  
  
const squared = item => item ** 2;  
const squaredValues = modifyItems(numbers, squared);
```

Особенности функций

2. Являются объектами высшего порядка

```
function getRaiser(pow) {  
    return function (num) {  
        return num ** pow;  
    }  
}
```

```
const squared = getRaiser(2);  
squared(2); // 4  
squared(3); // 9
```

```
const cube = getRaiser(3);  
cube(2); // 8
```

Особенности функций

2. Являются объектами высшего порядка

```
function getRaiser(pow) {  
    return function (num) {  
        return num ** pow;  
    }  
}
```

```
const squared = getRaiser(2);  
squared(2); // 4  
squared(3); // 9
```

```
const cube = getRaiser(3);  
cube(2); // 8
```

Особенности функций

2. Являются объектами высшего порядка

```
function getRaiser(pow) {  
    return function (num) {  
        return num ** pow;  
    }  
}
```

```
const squared = getRaiser(2);  
squared(2); // 4  
squared(3); // 9
```

```
const cube = getRaiser(3);  
cube(2); // 8
```

Особенности функций

2. Являются объектами высшего порядка

```
function getRaiser(pow) {  
    return function (num) {  
        return num ** pow;  
    }  
}
```

```
getRaiser(2)(3); // 9
```

```
getRaiser(3)(2); // 8
```

Особенности функций

2. Являются объектами высшего порядка

```
function createTimeExecutor() {  
    createTimeExecutor.ts = Date.now();  
  
    return function () {  
        return Date.now - createTimeExecutor.ts;  
    }  
}  
  
const executeTime = createTimeExecutor();  
  
executeTime(); // 0  
...
```

Особенности функций

2. Являются объектами высшего порядка

```
function createTimeExecutor() {  
    createTimeExecutor.ts = Date.now();  
  
    return function () {  
        return Date.now - createTimeExecutor.ts;  
    }  
}  
  
const executeTime = createTimeExecutor();  
  
executeTime(); // 0  
...
```

Не делайте так

Аргументы и параметры

Аргументы и параметры

*Аргументы – то, что передаем в функцию при вызове,
параметры – то, что описано в объявлении функции*

```
function sum(a, b) { // Параметры  
    return a + b;  
}  
  
sum(2, 3); // Аргументы
```

Аргументы – передаем
Параметры – принимаем

Аргументы и параметры

Аргументы

1. В функцию можно передавать или не передавать аргументы

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

```
sum(2, 3); // 5  
sum(); // NaN
```

Аргументы и параметры

Аргументы

2. Недостающие аргументы получают значение undefined

```
function sum(a, b) {  
    return a + b;  
}  
  
sum(1); // NaN (1 + undefined)
```

Аргументы и параметры

Аргументы

3. Лишние аргументы игнорируются

```
function sumAll(a, b, c, d, e, f) {  
    return a + b + c + d + e + f;  
}
```

```
sumAll(1, 2, 3, 4, 5, 6, 7, 8, 9); // 21
```

Аргументы и параметры

Аргументы

3. Лишние аргументы игнорируются, но можно передать произвольное количество

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

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

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

```
// Spread operator
```

```
Math.max(...numbers); // 5
```

Аргументы и параметры

Параметры

1. Функция может принимать или не принимать параметры

```
function getFive() {  
    return 5;  
}
```

```
getFive(); // 5
```

Аргументы и параметры

Параметры

1. Функция может принимать или не принимать параметры

```
function getHex(dec) {  
    return dec.toString(16);  
}
```

```
getHex(431); // '1af'
```

Аргументы и параметры

Параметры

2. Можно указать значение параметра по умолчанию

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

```
sum(1); // 3
```

```
sum(1, undefined); // 3
```

```
sum(1, null); // ???
```

Аргументы и параметры

Параметры

2. Можно указать значение параметра по умолчанию

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

```
sum(1); // 3
```

```
sum(1, undefined); // 3
```

```
sum(1, null); // 1
```

Только `undefined`,
переданный в явном
или неявном виде,
заставляет функцию
подставить значение по
умолчанию

Аргументы и параметры

Параметры

3. Можно принимать произвольное количество параметров: **Rest Operator**, arguments

```
function sumAll(...numbers) {  
    let total = 0;  
    for (let number of numbers) {  
        total += number;  
    }  
    return total;  
}  
  
sumAll(1, 2, 3, 4, 5, 6, 7, 8, 9); // 45
```

Аргументы и параметры

Параметры

3. Можно принимать произвольное количество параметров: **Rest Operator**, arguments

```
function sumAll(first, second, ...numbers) {  
    let total = first + second;  
    for (let number of numbers) {  
        total += number;  
    }  
    return total;  
}  
  
sumAll(1, 2, 3, 4, 5, 6, 7, 8, 9); // 45
```

Аргументы и параметры

Параметры

3. Можно принимать произвольное количество параметров: Rest Operator, **arguments**

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

```
sumAll(1, 2, 3, 4, 5, 6, 7, 8, 9); // 45
```

arguments
использовался в ES5,
сейчас не нужен

Аргументы и параметры

Аргументы передаются по значению или по ссылке

- **По значению** передаются примитивы: число, строка, etc. В этом случае все, что происходило с ним внутри функции, остается внутри функции
- **По ссылке** передаются сложные типы данных: массив, объект, функция, etc. В этом случае все изменения с этими данными изменяют

Аргументы и параметры

Аргументы передаются по значению или по ссылке

// По значению

```
function increment(num) {  
    num = num + 1;
```

```
    return num;  
}
```

```
let a = 5;
```

```
increment(a); // 6
```

```
console.log(a); // 5, исходное значение НЕ изменилось
```

Аргументы и параметры

Аргументы передаются по значению или по ссылке

// По ссылке

```
function incrementAll(array) {  
    for (let i = 0; i < array.length; ++i) {  
        array[i] = array[i] + 1;  
    }  
  
    return array;  
}  
  
let b = [1, 2, 3, 4];  
  
incrementAll(b); // [2, 3, 4, 5]
```

Аргументы и параметры

Аргументы передаются по значению или по ссылке

```
// По ссылке
```

```
function incrementAll(array) {  
    array = array.map(num => num + 1);
```

```
    return array;  
}
```

```
let b = [1, 2, 3, 4];
```

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

Исходное значение не
изменится, если
полностью
перезаписать объект

Аргументы и параметры

Именованные аргументы (ES5): объявление

```
function formatString(params) {  
    if (params.needCapitalized) {  
        ...  
    }  
  
    if (params.hasMarkdown) {  
        ...  
    }  
};
```

Аргументы и параметры

Именованные аргументы (ES5): объявление

```
function formatString(params) {  
    if (params.needCapitalized) {  
        ...  
    }  
  
    if (params.hasMarkdown) {  
        ...  
    }  
};
```

Паттерн, который предполагает заворачивание аргументов в объект

Аргументы и параметры

Именованные аргументы (ES5): вызов функции

```
formatString({  
    needCapitalized: false,  
    hasMarkdown: true,  
    text: 'A _роза_ упала **на лапу** Азора'  
});  
formatString({  
    text: 'A _роза_ упала **на лапу** Азора',  
    hasMarkdown: true,  
    needCapitalized: false  
});  
formatString({  
    text: 'A _роза_ упала **на лапу** Азора',  
    hasMarkdown: true
```

Все три вызова предполгают одинаковый результат

Именованные аргументы (ES5): Достоинства

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

Именованные аргументы (ES5): Недостатки

- Неявный интерфейс
- Неудобно работать с аргументами внутри функции

О нас уже позаботились!

Destructuring Operator

```
const obj = {  
    a: 5,  
    b: 8  
};  
  
const { a, b } = obj;  
  
console.log(a, b) // 5 8
```

Destructuring Operator

```
const obj = {  
    a: 5,  
    b: 8  
};
```

```
const { a, b } = obj;
```

```
console.log(a, b) // 5 8
```

Деструктуризация

Аргументы и параметры

Именованные аргументы (ES6)

```
// Destructuring Operator

function formatString1({ needCapitalized, hasMarkdown }) {
    return needCapitalized && hasMarkdown;
};

// Default Named Arguments
function formatString2({
    needCapitalized = false,
    hasMarkdown = false
}) {
    return needCapitalized && hasMarkdown;
};
```

Аргументы и параметры

Именованные аргументы (ES6)

```
// Destructuring Operator
```

```
function formatString1({ needCapitalized, hasMarkdown }) {  
    return needCapitalized && hasMarkdown;  
};
```

```
// Default Named Arguments  
function formatString2({  
    needCapitalized = false,  
    hasMarkdown = false  
) {  
    return needCapitalized && hasMarkdown;  
};
```

Аргументы и параметры

Именованные аргументы (ES6)

```
// Destructuring Operator

function formatString1({ needCapitalized, hasMarkdown }) {
    return needCapitalized && hasMarkdown;
};

// Default Named Arguments
function formatString2({
    needCapitalized = false,
    hasMarkdown = false
}) {
    return needCapitalized && hasMarkdown;
};
```

Аргументы и параметры

Именованные аргументы (ES6)

```
function formatString2({  
    needCapitalized = false,  
    hasMarkdown = false  
) {  
    return needCapitalized && hasMarkdown;  
};  
  
formatString2();  
// Uncaught TypeError:  
// Cannot destructure property of 'undefined' or 'null'.
```

Аргументы и параметры

Именованные аргументы (ES6)

```
function formatString2({  
    needCapitalized = false,  
    hasMarkdown = false  
) {  
    return needCapitalized && hasMarkdown;  
};  
  
formatString2(undefined);  
// Uncaught TypeError:  
// Cannot destructure property of 'undefined' or 'null'.
```

Аргументы и параметры

Именованные аргументы (ES6)

```
function formatString3({  
    needCapitalized = false,  
    hasMarkdown = false  
} = {  
    needCapitalized: false,  
    hasMarkdown: false  
) {  
    return needCapitalized && hasMarkdown;  
};  
  
formatString3(); // false
```

Аргументы и параметры

Именованные аргументы (ES6)

```
function formatString3({  
    needCapitalized = false,  
    hasMarkdown = false  
} = {  
    needCapitalized: false,  
    hasMarkdown: false  
) {  
    return needCapitalized && hasMarkdown;  
};  
  
formatString3(); // false
```

Уже не так все плохо!

Аргументы и параметры

Именованные аргументы (ES6)

```
function formatString4({  
    needCapitalized = false,  
    hasMarkdown = false  
} = {}) {  
    return needCapitalized && hasMarkdown;  
};  
  
formatString4(); // false
```

Аргументы и параметры

Именованные аргументы (ES6)

```
function formatString4({  
    needCapitalized = false,  
    hasMarkdown = false  
} = {}) {  
    return needCapitalized && hasMarkdown;  
};  
  
formatString4(); // false
```

Область видимости

Область видимости

*Область видимости – для конкретной
переменной, участок кода, на котором она
является корректной*

Область видимости

Пример 1

```
const phone = '(902)890-18-92';

function getPhoneString() {
    return `+7${phone}`;
}

getPhoneString(); // '+7(902)890-18-92'
```

Область видимости

Пример 2

```
function getPhoneString() {  
    return `+7${phone}`;  
}  
  
const phone = '(902)890-18-92';  
  
getPhoneString(); // '+7(902)890-18-92'
```

Область видимости

Пример 3

```
function getPhoneString() {  
    return `+7${phone}`;  
}
```

```
getPhoneString();
```

```
const phone = '(902)890-18-92';  
// Uncaught ReferenceError:  
// Cannot access 'phone' before initialization
```

Почему так?

Какие бывают области видимости?

- Статическая (лексическая) (определяется в момент парсинга кода, и о ней пойдет речь сейчас)
- Контекст исполнения (поговорим позже)

Бывает трех видов:

- Локальная (в нее помещаются все переменные, объявленные в функции, и аргументы, переданные в функцию)
- Глобальная (переменная или функция объявлена в глобальной области видимости, если она объявлена не в теле другой функции)
- Блочная (ES6)

Локальная область видимости

В нее помещаются все переменные, объявленные в функции, и аргументы, переданные в функцию

```
function getPhoneString(countryCode) {  
    const phone = '(902)890-18-92';  
  
    return `${countryCode}${phone}`;  
}
```

```
getPhoneString('+7'); // '+7(902)890-18-92'
```

Локальная область видимости

В нее помещаются все переменные, объявленные в функции, и **аргументы**, переданные в функцию

```
function getPhoneString(countryCode) {  
    const phone = '(902)890-18-92';  
  
    return `${countryCode}${phone}`;  
}  
  
getPhoneString('+7'); // '+7(902)890-18-92'
```

Локальная область видимости

В нее помещаются все **переменные**, объявленные в функции, и аргументы, переданные в функцию

```
function getPhoneString(countryCode) {  
    const phone = '(902)890-18-92';  
  
    return `${countryCode}${phone}`;  
}  
  
getPhoneString('+7'); // '+7(902)890-18-92'
```

Локальная область видимости

Внутри функции создается область видимости, которая имеет доступ к более широким областям видимости.

```
const phone = '(902)890-18-92';

function getPhoneString(countryCode) {
    return `${countryCode}${phone}`;
}

getPhoneString('+7'); // '+7(902)890-18-92'
```

Локальная область видимости

Внутри функции создается область видимости, которая имеет доступ к более широким областям видимости.

```
const phone = '(902)890-18-92';
```

```
function getPhoneString(countryCode) {  
    return `${countryCode}${phone}`;  
}
```

```
getPhoneString('+7'); // '+7(902)890-18-92'
```

Локальная область видимости

Внутри функции создается область видимости, которая имеет доступ к более широким областям видимости.

```
const phone = '(902)890-18-92';

function getPhoneString(countryCode) {
    return `${countryCode}${phone}`;
}

getPhoneString('+7'); // '+7(902)890-18-92'
```

Локальная область видимости

Если попытаться обратиться на чтение к переменной, которая задекларирована в области видимости, к которой мы не имеем доступа в данный момент

```
function getPhoneString() {  
    const phone = '(902)890-18-92';  
  
    return `+7${phone}`;  
}  
  
console.log(phone);  
// Uncaught ReferenceError: phone is not defined
```

Локальная область видимости

Если попытаться обратиться на чтение к переменной, которая задекларирована в области видимости, к которой мы не имеем доступа в данный момент

```
function getPhoneString() {  
    const phone = '(902)890-18-92';  
  
    return `+7${phone}`;  
}  
  
console.log(phone);  
// Uncaught ReferenceError: phone is not defined
```

Локальная область видимости

Где задекларирована функция, к таким переменным она имеет доступ

```
function createPhoneGetter() {  
    const phone = '(902)890-18-92';  
  
    return function () {  
        return `+7${phone}`; // '+7(902)890-18-92'  
    }  
}  
  
const getPhoneString = createPhoneGetter();
```

Локальная область видимости

Где задекларирована функция, к таким переменным она имеет доступ

```
function createPhoneGetter() {
    const phone = '(902)890-18-92';

    return function () {
        return `+7${phone}`; // '+7(902)890-18-92'
    }
}

const getPhoneString = createPhoneGetter();
```

Локальная область видимости

Где задекларирована функция, к таким переменным она имеет доступ

```
function formatter() {  
    return `+7${phone}`;  
    // Uncaught ReferenceError: phone is not defined  
}  
  
function createPhoneGetter() {  
    const phone = '(902)890-18-92';  
  
    return formatter;  
}  
  
const getPhoneString = createPhoneGetter();
```

Локальная область видимости

Где задекларирована функция, к таким переменным она имеет доступ

```
function formatter() {  
    return `+7${phone}`;  
    // Uncaught ReferenceError: phone is not defined  
}  
  
function createPhoneGetter() {  
    const phone = '(902)890-18-92';  
  
    return formatter;  
}  
  
const getPhoneString = createPhoneGetter();
```

Локальная область видимости

The screenshot shows a browser window with the URL <https://translate.google.ru>. The page content is the Google Translate interface, with the language set to "RUSSKIY". The developer tools are open, specifically the "Console" tab.

```
> const pi = Math.PI;  
function getCircleArea(r) {  
    return pi * Math.pow(r, 2);  
}  
  
function getCircleAreaInArmy(r) {  
    const pi = 3;  
    return pi * Math.pow(r, 2);  
}  
  
getCircleArea(5); // 78.53981633974483  
getCircleAreaInArmy(5); // 75  
✖ > Uncaught SyntaxError: Identifier 'pi' has already been declared at <anonymous>:1:1 VM154:1  
> pi  
< f (a){new ji(function(b,c){var d=a.length,e=[];if(d)for(var f=function(m,q){d-e[m]=q;0==d&&b(e)},g=function(m){c(m)},k=0,l;k<a.length;k++)l=a[k],oi(l,Sa(f,k),g);else b(e)})}
```

A red box highlights the error message: **Uncaught SyntaxError: Identifier 'pi' has already been declared at <anonymous>:1:1**. The file reference is VM154:1.

Локальная область видимости

При использовании паттерна "модуль" создается область видимости, и переменные внутри имеют уникальные имена

```
((() => {
    const cherry = 2;
    const raspberry = 3;

    const berryCount = cherry + raspberry; // berryCount = 5
})());

const berryCount = cherry + raspberry; // berryCount = NaN
```

Локальная область видимости

При использовании **паттерна "модуль"** создается область видимости, и переменные внутри имеют уникальные имена

```
((() => {
    const cherry = 2;
    const raspberry = 3;
    const berryCount = cherry + raspberry; // berryCount = 5
})());
```

IIFE (immediately-invoked function expression)

```
const berryCount = cherry + raspberry; // berryCount = NaN
```

Локальная область видимости

The screenshot shows a browser window with the URL <https://translate.google.ru>. The developer tools are open, specifically the 'Console' tab. The code in the console is as follows:

```
> ((() => {
    const pi = Math.PI;

    function getCircleArea(r) {
        return pi * Math.pow(r, 2);
    }

    function getCircleAreaInArmy(r) {
        const pi = 3;
        return pi * Math.pow(r, 2);
    }

    getCircleArea(5); // 78.53981633974483
    getCircleAreaInArmy(5); // 75
})()
```

The output of the code is shown below the console input:

```
< undefined
```

The browser interface includes a sidebar with navigation icons (History, Save, Communities), a top bar with tabs (Elements, Console, Redux), and a status bar at the bottom right with the number 79.

Глобальная область видимости

Переменная или функция объявлена в глобальной области видимости, если она объявлена не в теле другой функции

```
const phone = '(902)890-18-92';

function getPhoneString(countryCode) {
    const contact = 'Иваныч';

    return `${countryCode}${phone} (${contact})`;
}

const result = getPhoneString('+7');
```

Глобальная область видимости

Переменная или функция объявлена в глобальной области видимости, если она объявлена не в теле другой функции

```
const phone = '(902)890-18-92';

function getPhoneString(countryCode) {
    const contact = 'Иваныч';

    return `${countryCode}${phone} (${contact})`;
}

const result = getPhoneString('+7');
```

Блочная область видимости

Появилась в ES6

```
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);
```

Блочная область видимости

Появилась в ES6

```
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);
```

Блочная область видимости

Появилась в ES6

```
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);
```

Блочная область видимости

Появилась в ES6

```
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)

Затенение

Затенение

Если внутри одной функции объявляется переменная с идентификатором, который уже существует в родительской, то на время работы дочерней функции эта переменная принимает значение из ближайшей области видимости

```
const pi = Math.PI;

function getCircleArea(r) {
    return pi * Math.pow(r, 2);
}

function getCircleAreaInArmy(r) {
    const pi = 3;
    return pi * Math.pow(r, 2);
}

getCircleArea(5); // 78.53981633974483

getCircleAreaInArmy(5); // 75
```

Затенение

Если внутри одной функции объявляется переменная с идентификатором, который уже существует в родительской, то на время работы дочерней функции эта переменная принимает значение из ближайшей области видимости

```
const pi = Math.PI;

function getCircleArea(r) {
    return pi * Math.pow(r, 2); // pi = Math.PI
}

function getCircleAreaInArmy(r) {
    const pi = 3; // Shadowed-name
    return pi * Math.pow(r, 2); // pi = 3
}

getCircleArea(5); // 78.53981633974483

getCircleAreaInArmy(5); // 75
```

Благодаря этому механизму испортить переменные внутри функции нельзя

Замыкание

Замыкание – функция со всеми
внешними переменными, к которым
она имеет доступ

Замыкание позволяет скрывать ваши данные от доступа снаружи

Замыкание

Механизм "счетчик ссылок"

Код:

```
function greet() {  
    const text = 'Привет';  
    { text: 1 }  
}  
  
greet();
```

Счётчик ссылок:

Замыкание

Механизм "счетчик ссылок"

Код:

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

```
greet();
```

Счётчик ссылок:

```
{ text: 0 }
```

Замыкание

Механизм "счетчик ссылок"

Код:

```
function makeCounter() {  
    let currentCount = 0;  
  
    return function () {  
        return currentCount++;  
    }  
  
}  
  
const counter = makeCounter();
```

Счётчик ссылок:

Замыкание

Механизм "счетчик ссылок"

Код:

```
function makeCounter() {  
    let currentCount = 0;  
  
    return function () {  
        return currentCount++;  
    }  
}  
  
const counter = makeCounter();  
{ currentCount: 1 }
```

Счётчик ссылок:

Замыкание

Замыкание

Код:

```
function makeCounter() {  
    let currentCount = 0;  
  
    return function () {  
        return currentCount++;  
    }  
}
```

Область видимости:

Внутри любой функции
можно объявлять
другую функцию,
которая образовывает
свою область
видимости

Паттерн "фабрика"
(Factory)

Замыкание

Замыкание

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

```
counter(); // 0
```

```
counter(); // 1
```

```
counter(); // 2
```

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

```
yetAnother(); // 0
```

Функция вызывается в
другом месте кода, но
имеет доступ к
currentCount, которая
задекларирована в
области видимости

Замыкание

Замыкание

```
function greet(name) {  
    return function () {  
        return `Привет, ${name}`;  
    }  
  
}  
  
let helloWorld = greet('мир!');  
  
helloWorld(); // "Привет, мир!"
```

На этапе парсинга мы знаем, что factory возвращает ссылку на функцию

Создается функция, создается область видимости внутри и привязывается область видимости, которая находится в месте декларации

Замыкание

Функция имеет доступ к переменным,
которые были доступны ей **в месте ее
декларации, а не вызова**

Перерыв

this

~~this~~

Контекст исполнения

Контекст исполнения

Контекст исполнения тесно связан с областью видимости и содержит ее.

Кроме нее, содержит еще и ключевое слово **this**

Внутри любой функции мы имеем доступ к:

- набору аргументов, которые передали
- arguments
- this

Контекст исполнения
формируется на этапе
выполнения кода

this

this

Ключевое слово

Особенности:

- нельзя перезаписать
- указывает на текущий объект
- можно использовать за пределами объекта
- определяется в момент интерпретации участка кода (в функции в момент вызова)

this

Значение this зависит от:

1. Типа участка кода
2. Как мы попали на этот участок
3. Режима работы интерпретатора

this

Значение this зависит от:

1. Типа участка кода
2. Как мы попали на этот участок
3. Режима работы интерпретатора

this

1. Тип участка кода

1. Глобальный
2. Node.js модуль

this

1. Тип участка кода

1. Глобальный
2. ~~Node.js модуль~~

this. Тип участка кода.

1.1. Глобальный в браузере

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

```
window.innerWidth; // 1440
```

```
innerWidth; // 1440
```

В браузере в глобальной
области видимости this
ссылается на window

this. Тип участка кода.

1.1. Глобальный в Node.js

```
this.process.version; // 'v10.16.0'
```

```
global.process.version; // 'v10.16.0'
```

```
process.version; // 'v10.16.0'
```

В Node.js в глобальной области видимости this ссылается на global

this. Тип участка кода.

1.1. Глобальный

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

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

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

console.log(); также
вызывается у
глобального объекта,
работает аналогично:
this.console.log();

this

Значение this зависит от:

1. Типа участка кода
2. Как мы попали на этот участок
3. Режима работы интерпретатора

this

2. Как мы попали на этот участок кода

1. Функция объявлена в глобальной области видимости
2. Функция описана в модуле Node.js
3. Функция вызвана в качестве значения поля объекта
4. Функция вызвана в качестве значения поля прототипа объекта
5. Функция кладется в память в качестве значения некоторой переменной

this

2. Как мы попали на этот участок кода

1. ~~Функция объявлена в глобальной области видимости~~
2. ~~Функция описана в модуле Node.js~~
3. Функция вызвана в качестве значения поля объекта
4. Функция вызвана в качестве значения поля прототипа объекта
5. Функция кладется в память в качестве значения некоторой переменной

this

2. Как мы попали на этот участок кода

1. ~~Функция объявлена в глобальной области видимости~~
2. ~~Функция описана в модуле Node.js~~
3. Функция вызвана в качестве значения поля объекта
4. ~~Функция вызвана в качестве значения поля прототипа объекта~~
5. Функция кладется в память в качестве значения некоторой переменной

this. Как мы попали на этот участок кода.

2.3. Функция вызвана в качестве значения поля объекта

```
const block = {
    innerHeight: 200,
    getHeight: function () {
        return this.innerHeight;          // ??.innerHeight;
    }
}
```

this. Как мы попали на этот участок кода.

2.3. Функция вызвана в качестве значения поля объекта

```
const block = {  
    innerHeight: 200,  
  
    getHeight: function () {  
        return this.innerHeight;      // block.innerHeight;  
    }  
}  
  
block.getHeight(); // 200
```

this. Как мы попали на этот участок кода.

2.5. Функция кладется в память в качестве значения некоторой переменной

```
const block = {  
    innerHeight: 200,  
  
    getHeight: function () {  
        return this.innerHeight;           // ??.innerHeight;  
    }  
  
}  
  
const getHeight = block.getHeight;
```

this. Как мы попали на этот участок кода.

2.5. Функция кладется в память в качестве значения некоторой переменной

```
const block = {  
    innerHeight: 200,  
  
    getHeight: function () {  
        return this.innerHeight;  
    }  
  
}  
  
const getHeight = block.getHeight;  
getHeight();
```

this будет ссылаться на глобальный объект, при том не важно, где была определена функция

```
// window.innerHeight;
```

Такое поведение называется "займствование метода"

this. Как мы попали на этот участок кода.

2.6. Функция была передана в качестве callback

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

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

this. Как мы попали на этот участок кода.

2.6. Функция была передана в качестве callback

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

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

person.showItems();
```

this. Как мы попали на этот участок кода.

2.6. Функция была передана в качестве 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();
```

this. Как мы попали на этот участок кода.

2.6. Функция была передана в качестве callback

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

this. Как мы попали на этот участок кода.

2.6. Функция была передана в качестве callback

Проблема: в callback контекст как бы явно не задан и в качестве this подставляется global

this. Как мы попали на этот участок кода.

2.6. Функция была передана в качестве callback

Решение:

- В языке предусмотрена передача контекста исполнения callback'а вторым аргументом

this. Как мы попали на этот участок кода.

2.6. Функция была передана в качестве callback

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

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

person.showItems();
```

this. Как мы попали на этот участок кода.

2.6. Функция была передана в качестве callback

Проблема: Не все функции, работающие с callback, принимают контекст выполнения вторым аргументом

this. Как мы попали на этот участок кода.

2.6. Функция была передана в качестве callback

Решения:

- В языке предусмотрена передача контекста исполнения callback'а вторым аргументом
- Сохранить контекст в переменную: const self = this;
- Стрелочные функции

this. Как мы попали на этот участок кода.

2.6. Функция была передана в качестве callback

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

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

const self = person;
person.showItems();
```

this. Как мы попали на этот участок кода.

2.6. Функция была передана в качестве callback

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

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

person.showItems();
```

this

Методы привязки контекста

1. .call()

2. .apply()

3. .bind()

this. Методы привязки контекста

.call()

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

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

this. Методы привязки контекста

.apply()

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

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

this. Методы привязки контекста

.bind()

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

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

this. Методы привязки контекста

.bind()

Также .bind() позволяет использовать "частичное применение"

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

```
const binPow = Math.pow.bind(Math, 2);  
  
binPow(3); // 8  
binPow(10); // 1024
```

this. Методы привязки контекста

.bind()

Также .bind() позволяет использовать "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
```

Каррирование (Currying)

this

Значение this зависит от:

1. Типа участка кода
2. Как мы попали на этот участок
3. Режима работы интерпретатора

this

3. Режим работы интерпретатора

1. Режим обратной совместимости
2. Строгий режим

this. Режим работы интерпретатора.

3.1. Режим обратной совместимости

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

this совпадает с
контекстом, в котором
была вызвана функция

this. Режим работы интерпретатора.

3.2. Строгий режим

```
function getSelf() {  
    'use strict';  
  
    return this;  
}  
  
getSelf(); // undefined
```

Для функции,
вызванной в global
значение this –
undefined

Ссылки

- Функции на learn.javascript.ru
- Функции в JavaScript (MDN)
- Function Expression и функции-стрелки

Вопросы?

Спасибо!