Рекурсия и Стек
Числа и Строки

Занятие 4

GO IT

goit.com.ua

План занятия

- Глобальный объект
- Замыкания
- Scope для new Function
- Локальные переменные для объекта
- Модули через замыкания
- Практика

Глобальный объект

GO IT

goit.com.ua

global object

```
// объявление var создаёт свойство window.a
var a = 5;
console.log( window.a ); // 5
```

```
window.a = 5;
console.log( a ); // 5
```

Порядок инициализации

1. Инициализация, подготовка к запуску.
2. Выполнение

```
// На момент инициализации, до выполнения кода:
// window = { f: function, a: undefined, g: undefined }

var a = 5;
// window = { f: function, a: 5, g: undefined }

function f(arg) { /*...*/ }
// window = { f: function, a: 5, g: undefined }
// без изменений, f обработана ранее

var g = function(arg) { /*...*/ };
// window = { f: function, a: 5, g: function }
```

Порядок инициализации

```
console.log('a' in window); // true,  т.к. есть свойство window.a
console.log(a); // равно undefined,  присваивание будет выполнено далее
console.log(f); // function ...,  готовая к выполнению функция
console.log(g); // undefined, т.к. это переменная, а не Function Declaration

var a = 5;
function f() { /*...*/ }
var g = function() { /*...*/ };
```


```
a = 5;
console.log( a ); // 5
```

```
console.log( a ); // undefined
var a = 5;
```

```
console.log( a ); // error, a is not defined
a = 5;
```

var может быть несколько

```
var i = 10;

for (var i = 0; i < 20; i++) {
    //...
}

var i = 5;
```

Что выведет этот код?

```
if ('a' in window) {
    var a = 1;
}
console.log( a );
```

Замыкания, работа функций

GO IT

goit.com.ua

Лексическое окружение

LexicalEnvironment

```
function sayHi(name) {
    var phrase = 'Привет, ' + name;
    console.log( phrase );
}

sayHi('Вася');
```

Лексическое окружение

1. LexicalEnvironment
2. Функция выполняется
3. Очистка памяти

```
function sayHi(name) {
    // LexicalEnvironment = { name: 'Вася', phrase: undefined }
    var phrase = 'Привет, ' + name;

    // LexicalEnvironment = { name: 'Вася', phrase: 'Привет, Вася'}
    console.log( phrase );
}

sayHi('Вася');
```

Доступ ко внешним переменным

```
var userName = 'Вася';

function sayHi() {
    console.log( userName ); // 'Вася'
}
```

sayHi.[[Scope]] = window

Доступ ко внешним переменным

- Каждая функция при создании получает ссылку [[Scope]] на объект с переменными, в контексте которого была создана.
- При запуске функции создаётся новый объект с переменными LexicalEnvironment. Он получает ссылку на внешний объект переменных из [[Scope]].
- При поиске переменных он осуществляется сначала в текущем объекте переменных, а потом — по этой ссылке.

Всегда текущее значение

```
var phrase = 'Привет';

function say(name) {
    console.log(phrase + ', ' + name);
}

say('Вася');  // Привет, Вася (*)

phrase = 'Пока';

say('Вася'); // Пока, Вася (**)
```

Вложенные функции

```
function sayHiBye(firstName, lastName) {

    console.log( 'Привет, ' + getFullName() );
    console.log( 'Пока, ' + getFullName() );

    function getFullName() {
        return firstName + ' ' + lastName;
    }

}

sayHiBye('Вася', 'Пупкин');
// Привет, Вася Пупкин ; Пока, Вася Пупкин
```

Возврат функции

```
function makeCounter() {
    var currentCount = 1;

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

var counter = makeCounter(); // (*)

// каждый вызов увеличивает счётчик и возвращает результат
console.log( counter() ); // 1
console.log( counter() ); // 2
console.log( counter() ); // 3

// создать другой счётчик, он будет независим от первого
var counter2 = makeCounter();
console.log( counter2() ); // 1
```

Возврат функции

```
function() { // [[Scope]] -> {currentCount: 1}
    return currentCount++;
};
```

Возврат функции

```
var counter = makeCounter();

var counter2 = makeCounter();

console.log( counter() ); // 1
console.log( counter() ); // 2
console.log( counter() ); // 3

// счётчики независимы
console.log( counter2() ); // 1
```

Свойства функции

```
function f() {}

f.test = 5;
console.log( f.test );
```

Свойства функции

```
function makeCounter() {
    function counter() {
        return counter.currentCount++;
    }
    counter.currentCount = 1;

    return counter;
}

var counter = makeCounter();
console.log( counter() ); // 1
console.log( counter() ); // 2
```

Свойства функции

```
var counter = makeCounter();
console.log( counter() ); // 1

counter.currentCount = 5;

console.log( counter() ); // 5
```

Замыкание

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

Замыкание

1. Все переменные и параметры функций являются свойствами объекта переменных LexicalEnvironment. Каждый запуск функции создает новый такой объект. На верхнем уровне им является «глобальный объект», в браузере — window.
2. При создании функция получает системное свойство [[Scope]], которое ссылается на LexicalEnvironment, в котором она была создана.
3. При вызове функции, куда бы её ни передали в коде — она будет искать переменные сначала у себя, а затем во внешних LexicalEnvironment с места своего «рождения».

Задачка 7.1

Что будет, если вызов sayHi('Вася'); стоит в самом-самом начале, в первой строке кода?

```
say('Вася');

var phrase = 'Привет';

function say(name) {
    console.log( name + ", " + phrase );
}
```

Задачка 7.2

Каков будет результат выполнения этого кода?

```
var value = 0;

function f() {
    if (1) {
        value = true;
    } else {
        var value = false;
    }

    console.log( value );
}

f();
console.log(value);
```

Задачка 7.3

Что выведут эти вызовы?

```
var currentCount = 1;

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

var counter = makeCounter();
var counter2 = makeCounter();

console.log( counter() ); // ?
console.log( counter() ); // ?

console.log( counter2() ); // ?
console.log( counter2() ); // ?
```

[[Scope]] для new Function

GO IT

goit.com.ua

[[Scope]] для new Function

```
var a = 1;

function getFunc() {
    var a = 2;

    var func = new Function('', 'console.log(a)');

    return func;
}

getFunc()(); // 1, из window
```

Локальные переменные для объекта

GO IT

goit.com.ua

Счётчик-объект

```
var a = 1;

function getFunc() {
    var a = 2;

    var func = new Function('', 'console.log(a)');

    return func;
}

getFunc()(); // 1, из window
```

Модули

GO IT

goit.com.ua

Зачем нужен модуль?

hello.js

```
// глобальная переменная нашего скрипта
var message = "Привет";

// функция для вывода этой переменной
function showMessage() {
    console.log( message );
}

// выводим сообщение
showMessage();
```

Зачем нужен модуль?

```
<script>
    var message = 'Пожалуйста, нажмите на кнопку';
</script>
<script src="hello.js"></script>

<button>Кнопка</button>
<script>
    // ожидается сообщение из переменной выше...
    console.log( message );
    // но на самом деле будет введено "Привет"
</script>
```

Приём проектирования «Модуль»

```
(function() {

    // глобальная переменная нашего скрипта
    var message = "Привет";

    // функция для вывода этой переменной
    function showMessage() {
        console.log( message );
    }

    // выводим сообщение
    showMessage();

})();
```

Экспорт значения

```
(function() {

    // lodash - основная функция для библиотеки
    function lodash(value) {
        // ...
    }

    // вспомогательная переменная
    var version = '2.4.1';
    // ... другие вспомогательные переменные и функции

    // код функции size, пока что доступен только внутри
    function size(collection) {
        return Object.keys(collection).length;
    }

    // присвоим в lodash size и другие функции, которые нужно вынести из модуля
    lodash.size = size;
    // lodash.defaults = ...
    // lodash.cloneDeep = ...

    // "экспортировать" lodash наружу из модуля
    window._ = lodash; // в оригинальном коде здесь сложнее, но смысл тот же

}());
```

Экспорт значения

```
<script
src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/3.2.0/lodash.js">
</script>

<script>
    var user = {
        name: 'Вася'
    };

    _.defaults(user, {
        name: 'Не указано',
        employer: 'Не указан'
    });

    console.log( user.name ); // Вася
    console.log( user.employer ); // Не указан
    console.log( _.size(user) ); // 2
</script>
```

Экспорт через return

```
var lodash = (function() {

    var version;
    function assignDefaults() { /*...*/ }

    return {
        defaults: function() {  }
    }

})();
```

Практика

Задачка 7.4

```
function funky(o) {
    o = null;
}

var x = [];

funky(x);
console.log(x);

// A. null
// B. []
// C. undefined
// D. throw
```

Задачка 7.5

Напишите функцию которая принимает аргумент и возвращает этот аргумент.

```
identity(3); // 3
```

Задачка 7.5

решение

```
function identity(x) {
    return x;
}
```

Задачка 7.6

Напишите две бинарные функции add и mul которые складывают и умножают свои 2 аргумента.

```
add(3,4); //7

mul(3,4); //12
```

Задачка 7.6

решение

```
function add(x,y) {
    return x + y;
}

function mul(x,y) {
    return x * y;
}
```

Задачка 7.7

Напишите функцию которая принимает аргумент и возвращает функцию которая возвращает этот аргумент.

```
var idf = identityf(3);

console.log(idf()); // 3
```

Задачка 7.7

решение

```
function identityf(x) {
    return function() {
        return x;
    };
}
```

Задачка 7.8

Напишите функцию складывает числа из 2х вызовов

```
addf(3)(4); //7
```

Задачка 7.8

решение

```
function addf(x) {
    return function(y) {
        return x + y;
    };
}
```

Задачка 7.9

Напишите функцию которая принимает бинарную функцию и вызывает её с имя аргументами.

```
var addf2 = applyf(add);
console.log(addf2(3)(4)); //7
console.log(applyf(mul)(3)(4)); //12
```

Задачка 7.9

решение

```
function applyf(binary) {
    return function(x) {
        return function(y) {
            return binary(x)(y);
        }
    }
}
```

План занятия

- Глобальный объект
- Замыкания
- Scope для new Function
- Локальные переменные для объекта
- Модули через замыкания
- Практика

2015