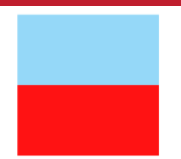
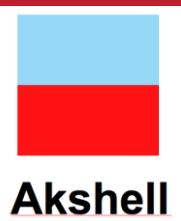
ECMAScript > Будущее

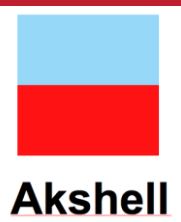
Дмитрий Сошников
http://dmitrysoshnikov.com



ECMAScript > ES6



let : Блочная область видимости



let : Блочная область видимости

```
// ES3


if (false) {
    var logLevel = 10;
}

alert(logLevel); // ?
```

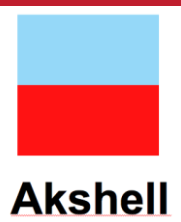
let : Блочная область видимости

```
// ES3
var logLevel; // = undefined;

if (false) {
    logLevel = 10;
}

alert(logLevel); // undefined
```

Все переменные создаются *до* запуска кода – при входе в контекст.

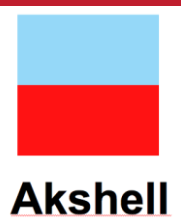
Так называемое «*поднятие*» (*hoisting*) переменных.

See: http://dmitrysoshnikov.com/notes/note-4-two-words-about-hoisting/



let : Блочная область видимости

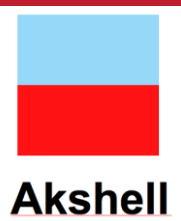
```
// ES6 Harmony


if (false) {
    let logLevel = 10;
}

alert(logLevel); // ReferenceError
```



let : Блочная область видимости

```
// ES3, ES5

var handlers = [];

for (var k = 0; k < 3; k++) {
    handlers[k] = function () {
        alert(k);
    };
}
```

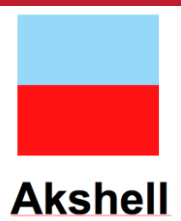
```
handlers[0](); // ?
handlers[1](); // ?
handlers[2](); // ?
```



let : Блочная область видимости

```
// ES3, ES5

var handlers = [];
var k;
for (k = 0; k < 3; k++) {
    handlers[k] = function () {
        alert(k);
    };
}
```

```
handlers[0](); // 3
handlers[1](); // 3
handlers[2](); // 3
```



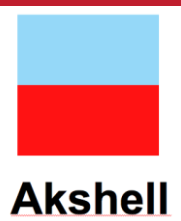
let : Блочная область видимости

ES3, ES5

```
for (var k = 0; k < 3; k++) {
    (function (x) {
        handlers[x] = function () {
            alert(x);
        };
    })(k);
}

handlers[0](); // 0
```

ES6

```
for (let k = 0; k < 3; k++) {
    let x = k;
    handlers[x] = function () {
        alert(x);
    };
}


handlers[0](); // 0
```



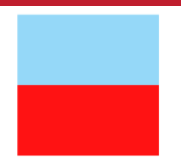
let : Блочная область видимости

```
let x = 10;
let y = 20;

let (x = x * 2, y = 30) {
    console.log(x + y); // 50
}

console.log(x + y); // 30

console.log(let (x = 100) x); // 100
console.log(x); // 10
```

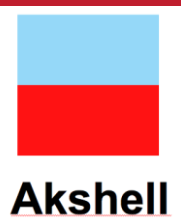
// let-объявление (definition)

// let-инструкции (statement)

// let-выражения (expression)

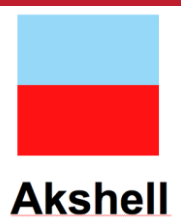


const : константы



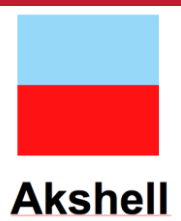
const : константы

```
const MAX_SIZE = 100;

// нельзя перезаписать (let, var, etc.)
let MAX_SIZE = 100; // error
```

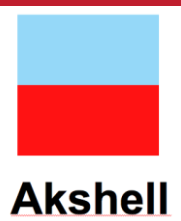


const : константные функции

```
const registerUser() {
  // реализация
}

// ошибка повторного объявления (function, let, var)
function registerUser() { ... }
```

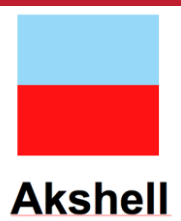


Параметры по умолчанию



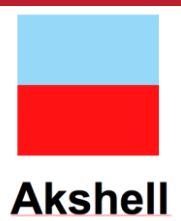
Параметры по умолчанию

```
function handleRequest(data, method) {
    method = method || “GET”;
    …
}
```

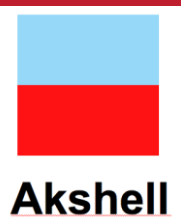


Параметры по умолчанию

```
function handleRequest(data, method) {
    method = method || “GET”;
    …
}

function handleRequest(data, method = “GET”) {
    …
}
```

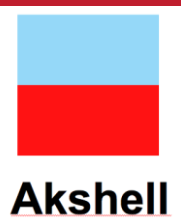


Деструктуризация или «нестрогий pattern-matching»



Деструктуризация: массивы

```
// для массивов
let [x, y] = [10, 20, 30]; // нестрогий matching

console.log(x, y); // 10, 20
```



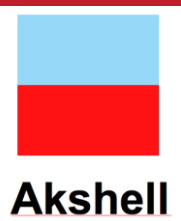
Деструктуризация: объекты

```
// for objects
let user = {name: “Ann”, location: {x: 10, y: 20}};

let {name: userName, location: {x: x, y: y}} = user;

console.log(userName, x, y); // “Ann”, 10, 20
```

Деструктуризация параметров функций

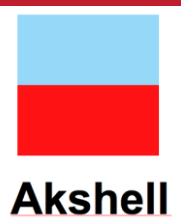
```
function Panel(config) {
    var title = config.title;
    var x = config.pos[0];
    var y = config.pos[1];
    return title + x + y;
}

new Panel({title: “Users”, pos: [10, 15]});
```

Слишком «шумно»



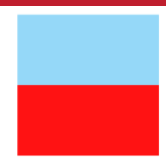
Деструктуризация параметров функций

```
function Panel({title: title, pos: [x, y]}) {
    return title + x + y;
}

let config = {title: “Users”, pos: [10, 15]};

new Panel(config);
```

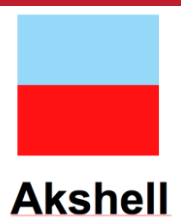


Деструктуризация: обмен переменных

```
// обмен двух переменных без третьей?

let x = 10;
let y = 20;

[x, y] = [y, x];  // легко
```

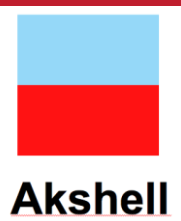


Замена arguments: “rest” и “spread”



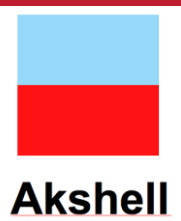
Объект arguments

```
// ES3, ES5

function format(pattern /*, rest */) {
    var rest = [].slice.call(arguments, 1);
    var items = rest.filter(function (x) { return x > 1});
    return pattern.replace("%v", items);
}

format("scores: %v", 1, 5, 3); // scores: 5, 3
```



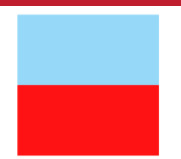
Прощай, arguments

```
// ES3, ES5

function format(pattern /*, rest */) {
    var rest = [].slice.call(arguments, 1); // сложно
    var items = rest.filter(function (x) { return x > 1});
    return pattern.replace("%v", items);
}

format("scores: %v", 1, 5, 3); // scores: 5, 3
```



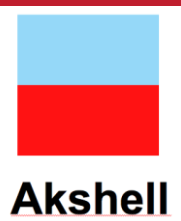
Привет, “rest”

```
// ES6 aka Harmony

function format(pattern, ...rest) {     // настоящий массив
    var items = rest.filter(function (x) { return x > 1});
    return pattern.replace(“%v”, items);
}

format(“scores: %v”, 1, 5, 3); // scores: 5, 3
```



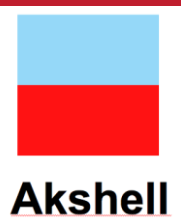
А также “spread”

```
// ES6 aka Harmony

function showUser(name, age, weight) {
    return name + “:” + age + weight;
}

let user = [“Alex”, 28, 130];

showUser(...user); // ok
showUser.apply(null, user); // desugared
```

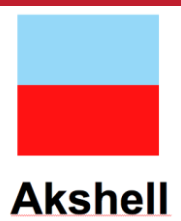


“rest” массивов при деструктуризации

```
// ES6 aka Harmony

let userInfo = [“John”, 14, 21, 3];
let [name, ...scores] = userInfo;

console.log(name); // “John”
console.log(scores); // [14, 21, 3]
```

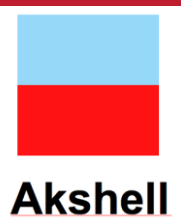


Сокращенные нотации



Сокращения в деструктуризации

```
let 3DPoint = {x: 20, y: 15, z: 1};

// полная нотация
let {x: x, y: y, z: z} = 3DPoint;

// сокращенная нотация
let {x, y, z} = 3DPoint;
```

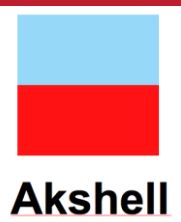


Короткий синтаксис функций -> функции

```
// обычная функция
[1, 2, 3].map(function (x) { return x * x; }); // [1, 4, 9]

// -> функция
[1, 2, 3].map((x) -> x * x); // [1, 4, 9]
```

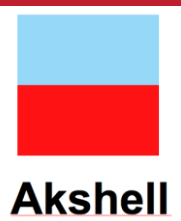
Синтаксически:

- опциональный return;
- -> вместо function
- Блочная скобки не обязательны



-> функции: примеры

```
// Пустая функция
let empty = ->;

// Скобки тела функции необязательны
let square= (x) -> x * x;

// Функция без параметров
let getUser = -> users[current];

// Для сложных выражений скобки нужны
let users = [{name: “Mark”, age: 28}, {name: “Sarah”, age: 26}];
users.forEach((user, k) -> { if (k > 2) console.log(user.name, k) });
```

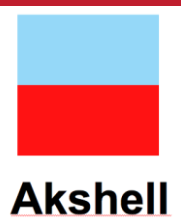


Обычные функции: динамический this

```
function Account(customer, cart) {
    this.customer = customer;
    this.cart = cart;
    $('#shopping-cart').on('click',  function (event) {
        this.customer.purchase(this.cart); // error on click
    });
}
```

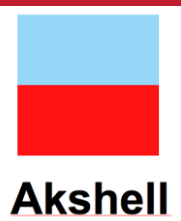
Решения:

```
var that = this;
.bind(this)
```

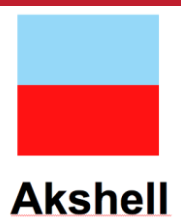


=> функции: лексический this

```
function Account(customer, cart) {
    this.customer = customer;
    this.cart = cart;
    $('#shopping-cart').on('click', (event) =>
        this.customer.purchase(this.cart); // OK
    );
}
```



Но… Сейчас также функции-блоки на повестке дня!

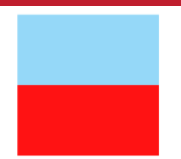


Короткий синтаксис функций функции-блоки

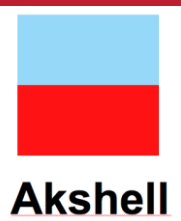
```
// обычная функция
[1, 2, 3].map(function (x) { return x * x; }); // [1, 4, 9]

// функция-блок
[1, 2, 3].map {|x| x * x}; // [1, 4, 9]
```

Синтаксически:

- опциональный **return**;
- **|x|** вместо **function**
- Необязательные скобки вызова



Proxy объекты : мета уровень



Proxy-объекты

```
/* handler – обработчик мета-уровня
  * proto – прототип прокси-объекта */
Proxy.create(handler, [proto])

/* handler – мета-обработчик
  * call – проксирование вызова
  * construct – проксирование конструирования */
Proxy.createFunction(handler, [call, [construct]])
```

See: http://wiki.ecmascript.org/doku.php?id=harmony:proxies

Proxy-объекты

```
// original object
let point = {
    x: 10,
    y: 20
};
```

```
// proxied object
let loggedPoint = Proxy.create({
    get: function (rcvr, name) {
        console.log("get: ", name);
        return point[name];
    },
    set: function (rcvr, name, value) {
        console.log("set: ", name, value);
        point[name] = value;
    }
}, Object.getPrototypeOf(point));
```

Перехват чтения свойства

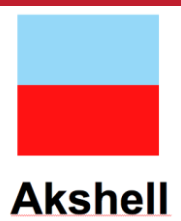
Перехват записи свойства

Proxy-объекты

Мета-обработчик

```
// перехват чтения
loggedPoint.x; // get:  x, 10

// перехват записи
loggedPoint.x = 20; // set:  x, 20

// отражается на оригинале
point.x; // 20
```

```
// proxied object
let loggedPoint = Proxy.create({
  get: function (rcvr, name) {
    console.log("get: ", name);
    return point[name];
  },
  set: function (rcvr, name, value) {
    console.log("set: ", name, value);
    point[name] = value;
  }
}, Object.getPrototypeOf(point));
```



Callable Proxy-объекты

```
// original object
let point = {x: 10, y: 20};
```

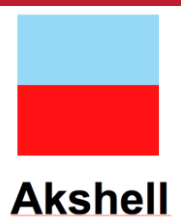
```
function callTrap() {
    console.log("call");
}

function constructTrap() {
    console.log("construct");
}
```

```
loggedPoint(10, 20);
new loggedPoint(100);
```

```
// proxied object
let loggedPoint = Proxy.createFunction({
    get: function (rcvr, name) {
        console.log("get: ", name);
        return point[name];
    },
    set: function (rcvr, name, value) {
        console.log("set: ", name, value);
        point[name] = value;
    }
}, callTrap, constructTrap);
```

Перехват вызова

Перехват конструирования



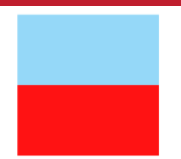
Proxy : простейший логгер на чтение

```
function logged(object) {
    return Proxy.create({
        get: function (rcvr, name) {
            console.log("get:", name);
            return object[name];
        }
    }, Object.getPrototypeOf(object));
}

let connector = logged({
    join: function (node) { ... }
});

connector.join("store@master-node"); // get: join
```

Proxy : примеры

```
// логгеры (на чтение и запись)
Proxy.create(logHandler(object));

// множественное наследование (делегирующие примеси)
Proxy.create(mixin(obj1, obj2));

// noSuchMethod
Proxy.create(object, noSuchMethod)

// Массивы с негативными индексами (как в Python, Ruby)
let a = Array.new([1, 2, 3]);
console.log(a[-1]); // 3
a[-1] = 10; console.log(a); // [1, 2, 10]
```

See: https://github.com/DmitrySoshnikov/es-laboratory/tree/master/examples

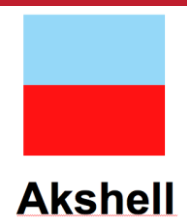


Система модулей

Модули в ES3, ES5

```
var DBLayer = (function (global) {
    /* save original */
    var originalDBLayer = global.DBLayer;
    function noConflict() {
        global.DBLayer = originalDBLayer;
    }
    /* implementation */
    function query() { ... }
    /* exports, public API */
    return {
        noConflict: noConflict,
        query: query
    };
})(this);
```

1. Создать локальный скоп
2. Функция восстановления
3. Имплементация
4. Публичный интерфейс



Модули в ES3, ES5

```
var DBLayer = (function (global) {
    /* save original */
    var originalDBLayer = global.DBLayer;
    function noConflict() {
        global.DBLayer = originalDBLayer;
    }
    /* implementation */
    function query() { ... }
    /* exports, public API */
    return {
        noConflict: noConflict,
        query: query
    };
})(this);
```

1. Создать локальный скоп
2. Функция восстановления
3. Имплементация
4. Публичный интерфейс

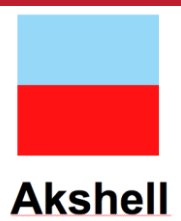
Слишком много
синтаксического «шума».
Нужен «сахар»

Модули в ES6

```
module DBLayer {
    export function query(s) { ... }
    export function connection(...args) { ... }
}

DBLayer.connection(”accounts/recent”);

import DBLayer.*; // импортируем все

import DBLayer.{query, connection: attachTo}; // только нужные экспорты

query(“SELECT * FROM books”).format(“escape | split”);

attachTo(“/books/store”, {
    onSuccess: function (response) { ... }
})
```

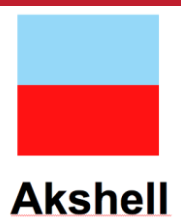
Внешние модули в ES6

```
// на файловой системе
module $ = require(“./library/selector.js”);

// глобально из сети; сами определяем имя модуля
module CanvasLib = require(“http:// ... /js-modules/canvas.js”);

// используем напрямую
let rect = new CanvasLib.Rectangle({width: 30, height: 40, shadow: true});

// или импортируем только нужные объекты
import CanvasLib.{Triangle, rotate};

rotate(-30, new Triangle($.query(...params)));
```



Деструктуризация и импорт внешних модулей

```
// запрашиваем модуль и напрямую
// импортируем через pattern-matching
let {read, format} = require(“fs.js”);

// используем импортированные функции
read(“storage/accounts.dat”).format(“%line: value”)
```



Генераторы : итераторы и кооперативная многозадачность

Генераторы : yield
«бесконечные» потоки

```
function fibonacci() {
    let [prev, curr] = [0, 1];
    while (true) {
        [prev, curr] = [curr, prev + curr];
        yield curr;
            ○        <----------- Точка следующего входа
    }
}

for (let n in fibonacci()) {
    // truncate the sequence at 1000
    if (n > 1000) break; // 1, 2, 3, 5, 8 ...
    console.log(n);
}
```

Ручная итерация:

```
let seq = fibonacci();
seq.next(); // 1
seq.next(); // 2
seq.next(); // 3
seq.next(); // 5
seq.next(); // 8
```

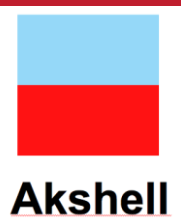
Генераторы : yield
кастомные итераторы

```
function iterator(object) {
  for (let k in object) {
    yield [k, object[k]];
  }
}

let foo = {x: 10, y: 20};

for (let [k, v] in iterator(foo)) {
  console.log(k, v); // x 10, y 20
}
```

Предложенные итераторы ES6:

```
// по свойствам (key+value)
for (let [k, v] in properties(foo))

// по значениям
for (let v in values(foo))

// по именам свойств
for (let k in keys(foo))
```

See: http://wiki.ecmascript.org/doku.php?id=strawman:iterators
See: https://gist.github.com/865630



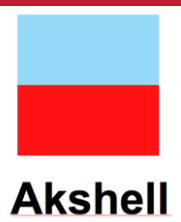
Генераторы: yield
асинхронное программирование

Callback'и

```
xhr("data.json", function (data) {
    xhr("user.dat", function (user) {
        xhr("user/save/", function (save) {
            /* code */
        }
    }
});

/* other code */
```

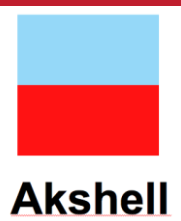
Сопрограммы

```
new Task(function () {
  let data  = yield xhr("data.json“);
  let user  = yield xhr(“user.dat“);
  let save  = yield xhr(“/user/save“);
  /* code */
});


/* other code */
```

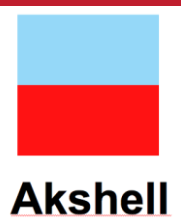


Генераторы : yield
кооперативная многозадачность

```
let thread1 = new Thread(function (...args) {
    for (let k in values([1, 2, 3])) yield k + " from thread 1";
}).start();

let thread2 = new Thread(function (...args) {
    for (let k in values([1, 2, 3])) yield k + " from thread 2";
}).start();

// 1 from thread 1
// 2 from thread 1
// 1 from thread 2
// 3 from thread 1
// 2 from thread 2
// etc.
```

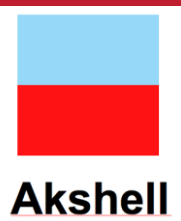


Array comprehensions



Array comprehensions

```
// map + filter

let scores = [1, 7, 4, 9]
    .filter(function (x) { return x > 5 })
    .map(function (x) { return x * x }); // [49, 81]
```



Array comprehensions

```
// map + filter

let scores = [1, 7, 4, 9]
    .filter(function (x) { return x > 5 })
    .map(function (x) { return x * x }); // [49, 81]


// array comprehensions

let scores = [x * x for (x in values([1, 7, 4, 9])) if (x > 5)];
```

Спасибо за внимание

Дмитрий Сошников

dmitry.soshnikov@gmail.com
http://dmitrysoshnikov.com

@DmitrySoshnikov