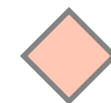


Программирование на JavaScript

Лекция 3
Функции. Замыкания.

Hexlet University

Где писать JS?



Функция – объект

- Со свойствами name, length и prototype
- Может использоваться как любой другой объект: храниться в переменных, других объектах, передаваться как аргумент и возвращать как значение
- Функциям можно задавать свойства и методы (!)
- В отличие от других объектов, функцию можно вызвать

Общий вид

зарезервированное слово

необязательное имя

```
function myFuncName (x, y, z) {  
    ...  
};
```

параметры

тело функции

зарезервированное слово

необязательное имя

function myFuncName (x, y, z) {

**...
};**

параметры

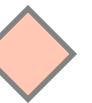
тело функции

свойство	значение
name	myFuncName
length	3
prototype	.

Создание с помощью литералов

```
var average = function (x, y) {  
    return (x+y)/2;  
};
```

this

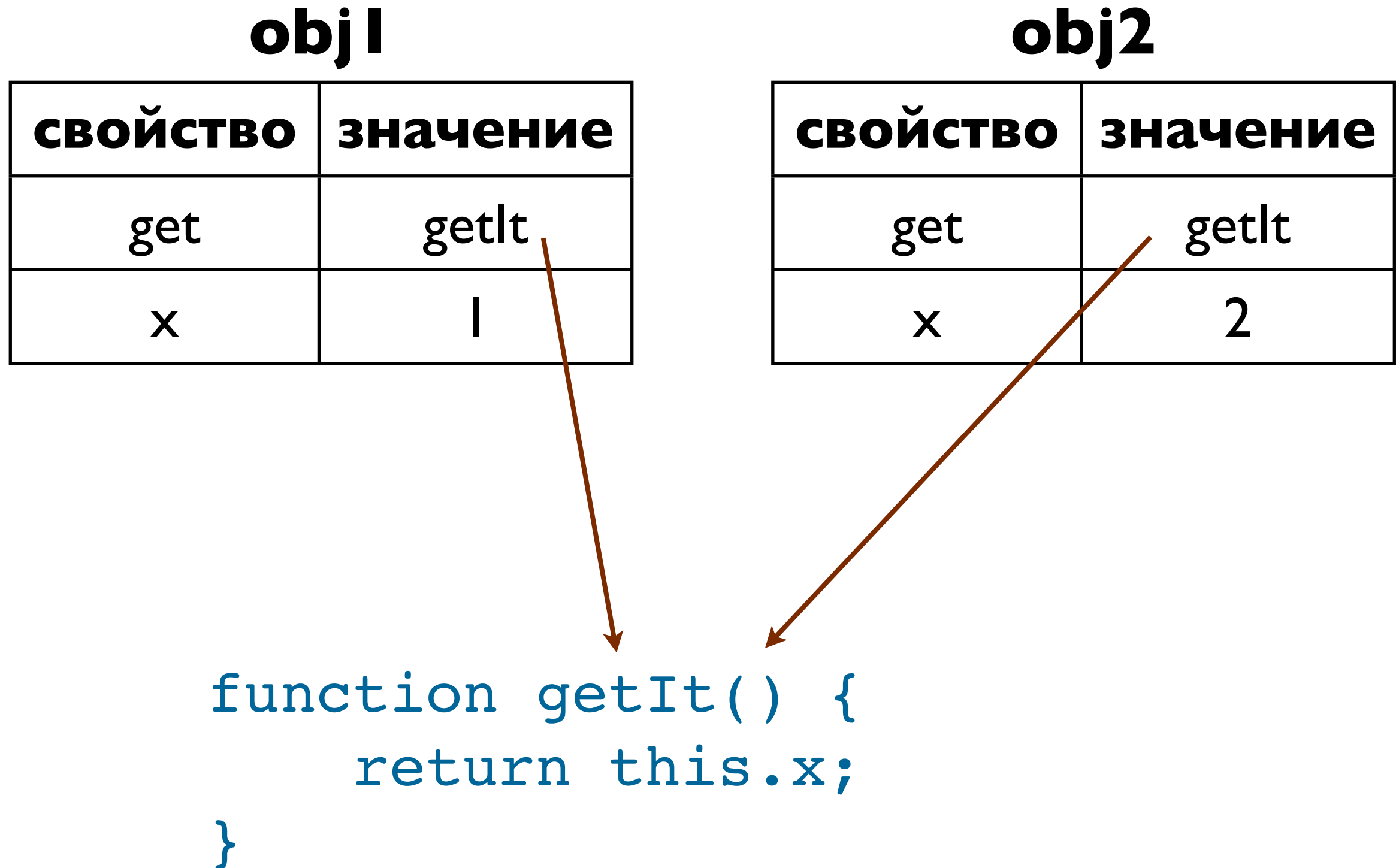


obj1

свойство	значение
get	getIt
x	1

obj2

свойство	значение
get	getIt
x	2



```
function getIt() {  
    return this.x;  
}
```


`obj1.get();`

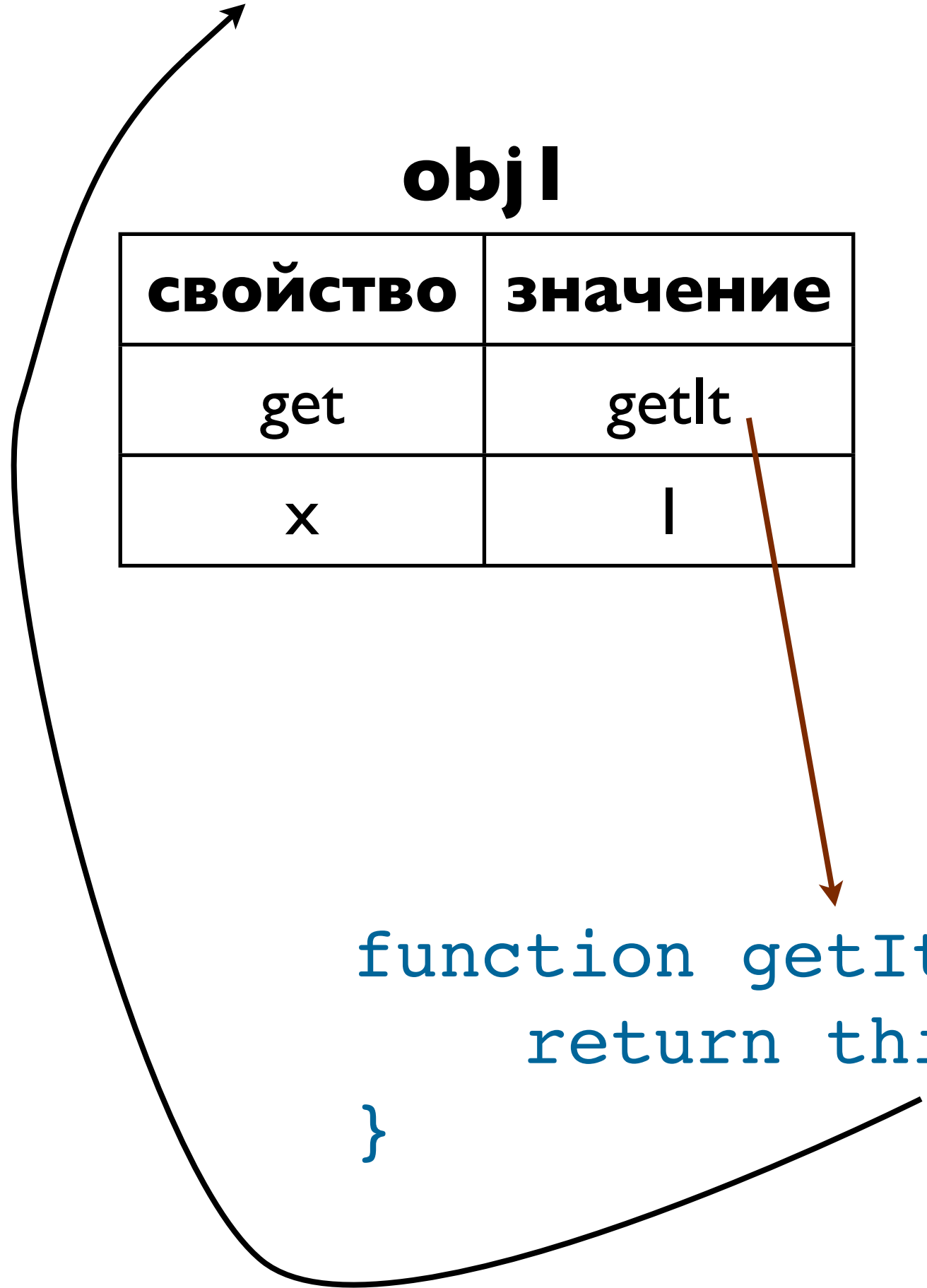
obj1

свойство	значение
get	getIt
x	1

obj2

свойство	значение
get	getIt
x	2

`function getIt() {
 return this.x;
}`



obj2.get();

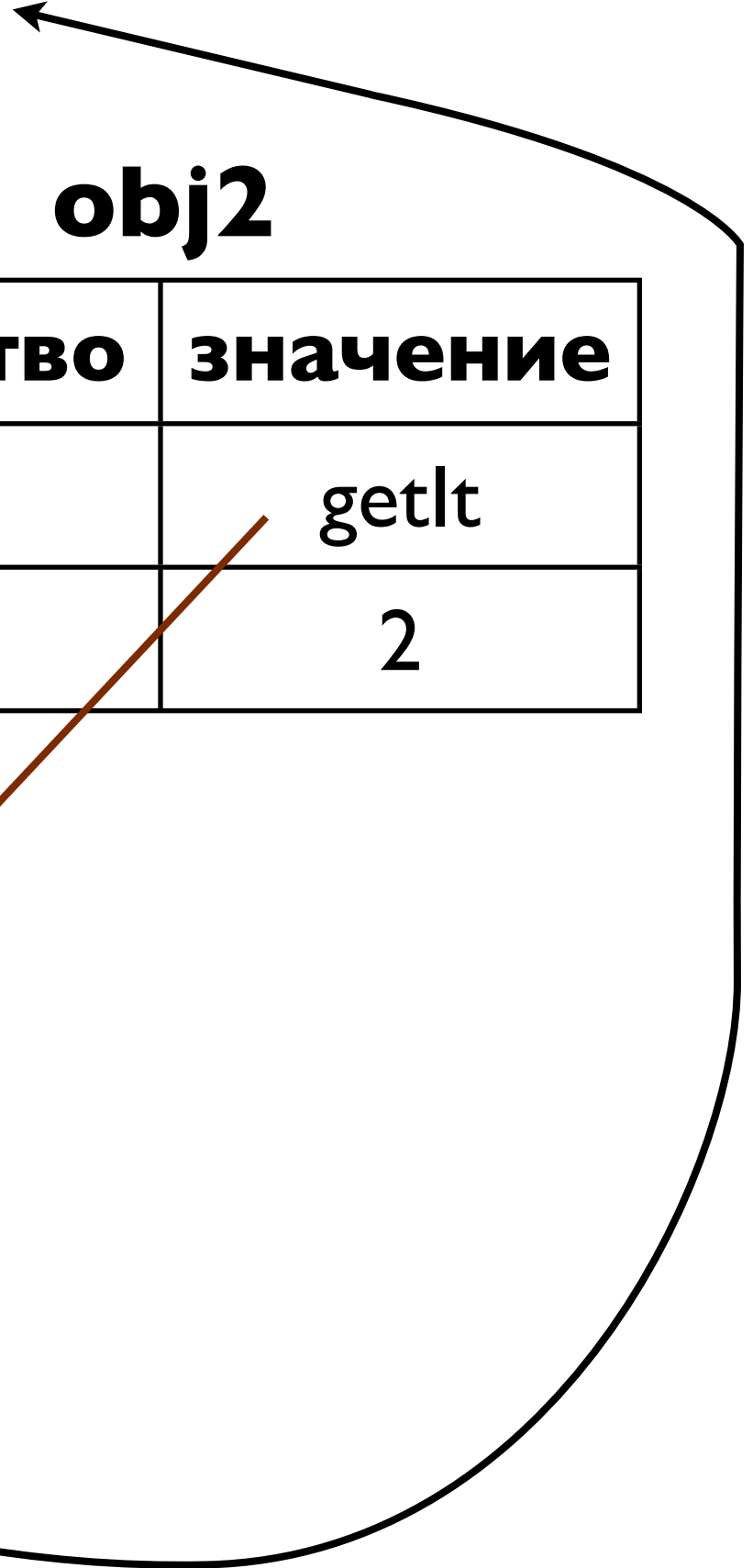
obj1

свойство	значение
get	getIt
x	1

obj2

свойство	значение
get	getIt
x	2

```
function getIt() {  
    return this.x;  
}
```



Функция как свойство объекта

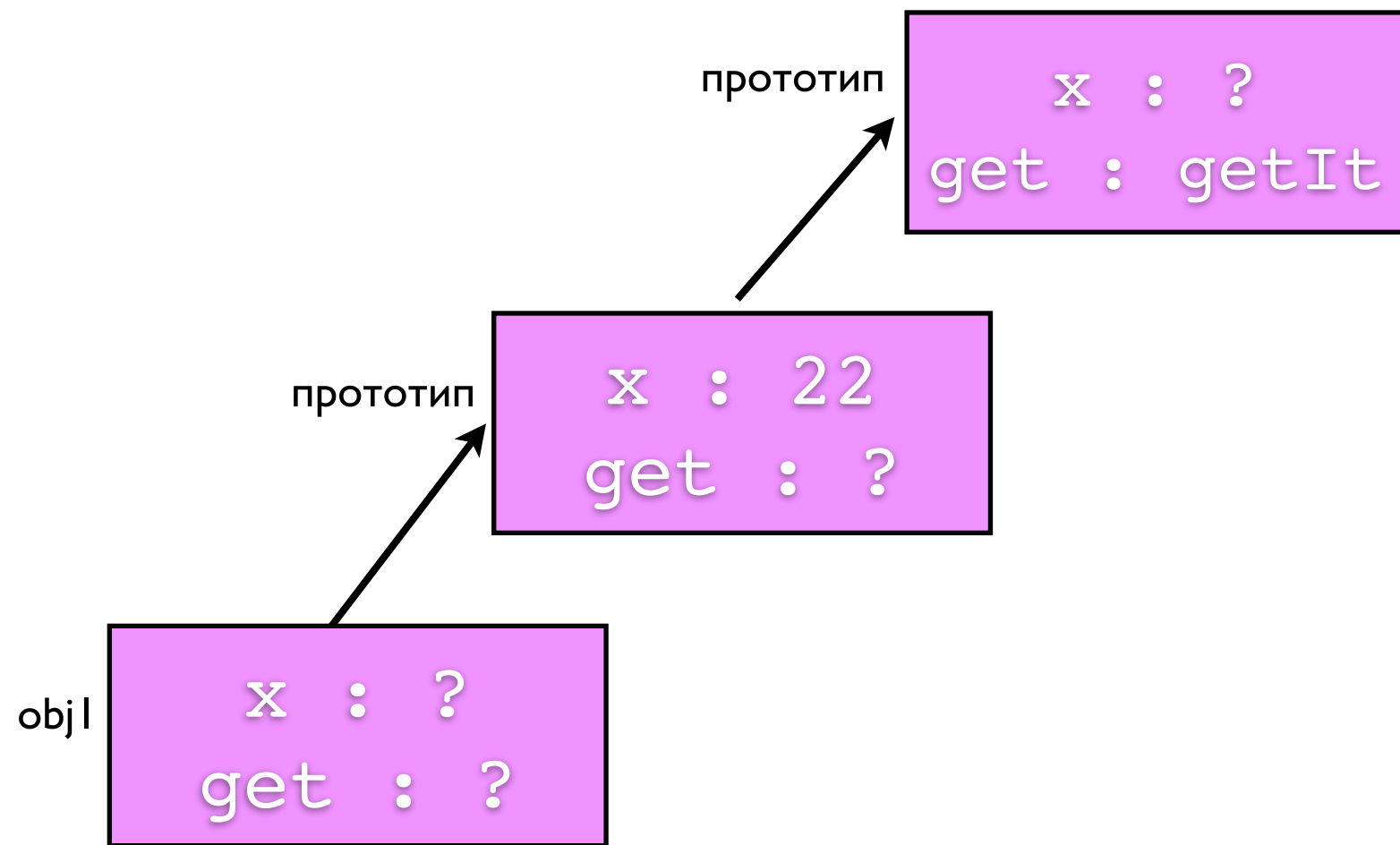
```
var obj = {  
  base      : 13,  
  average : function (x, y) {  
    return (this.base+x+y)/3;  
  };  
};
```



указывает на объект obj

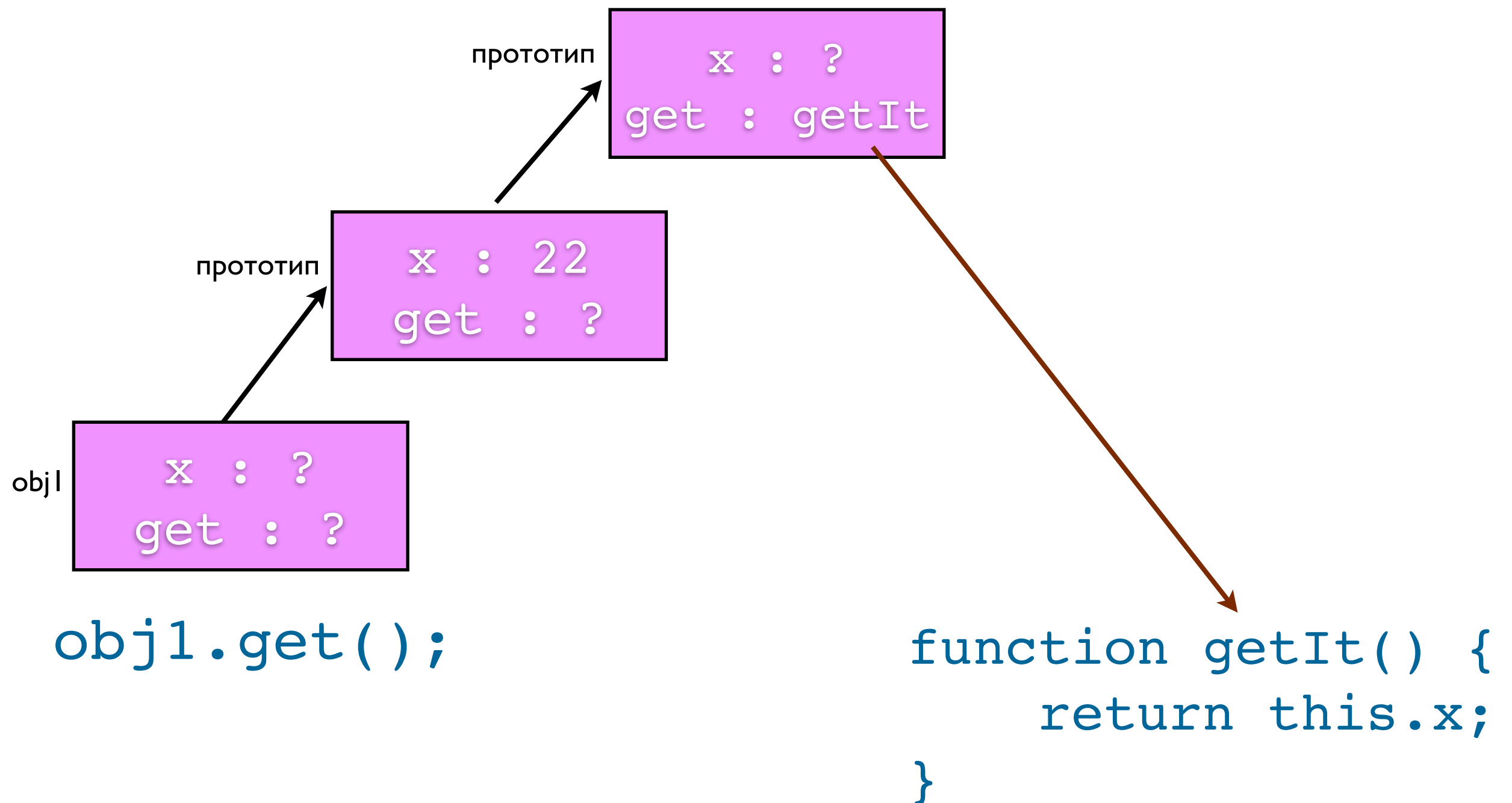
В момент вызова

Цепочки

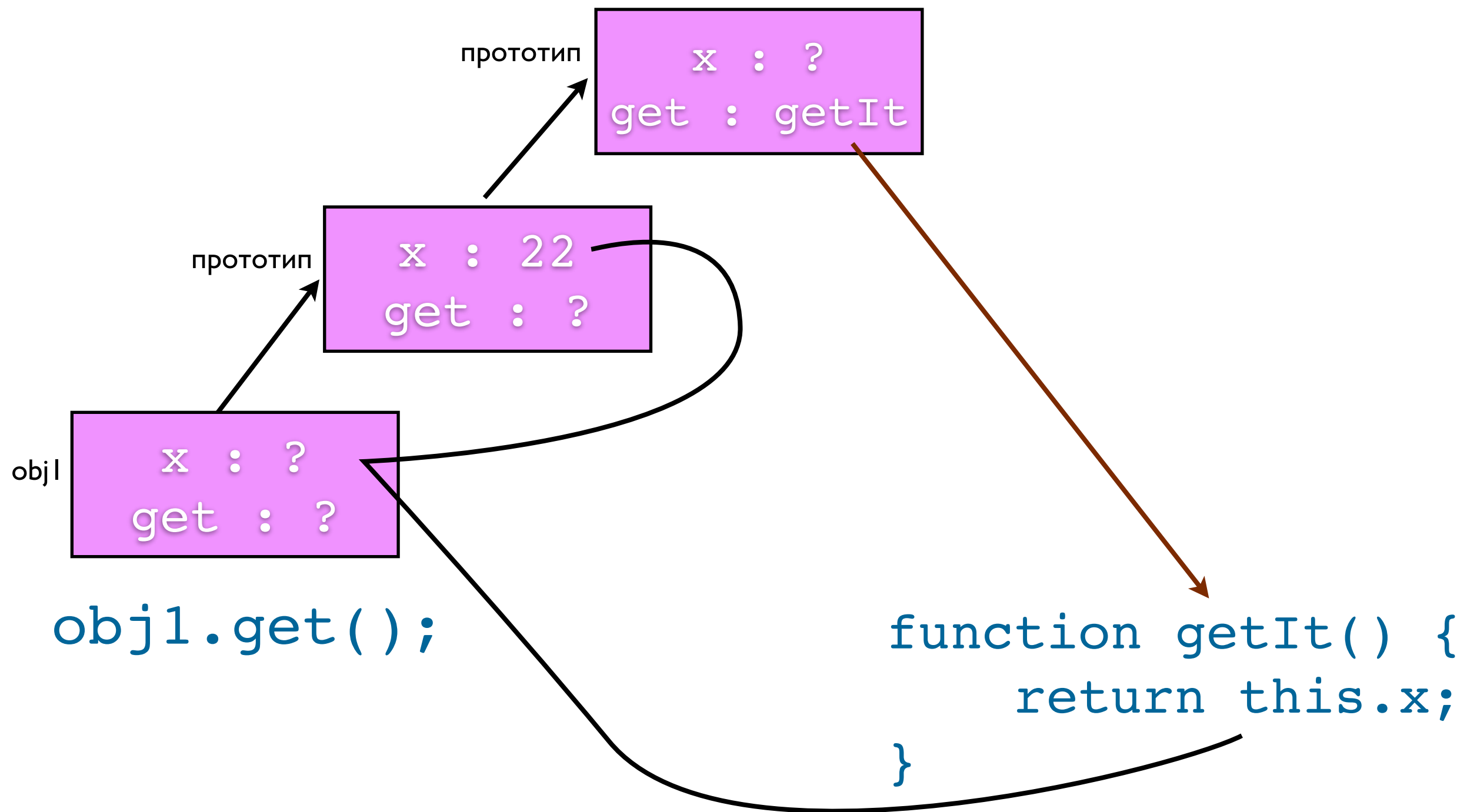


`obj1.get();`

Цепочки



Цепочки



Функция вне объекта

```
var answer = someFunc(1,3,19);
```



В этом вызове `this` указывает
на глобальный объект

(в ECMAScript 5 Strict Mode `this` будет
иметь значение `undefined`)

arguments

- Дополнительный параметр, доступный во время вызова функции
- Доступ ко всем аргументам (а не только параметрам)
- Позволяет писать функции с неопределенным количеством аргументов

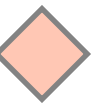
параметры



```
function myFuncName (x, y, z) {...};  
myFuncName (1,2,3,4,5,6,7,8);
```



аргументы

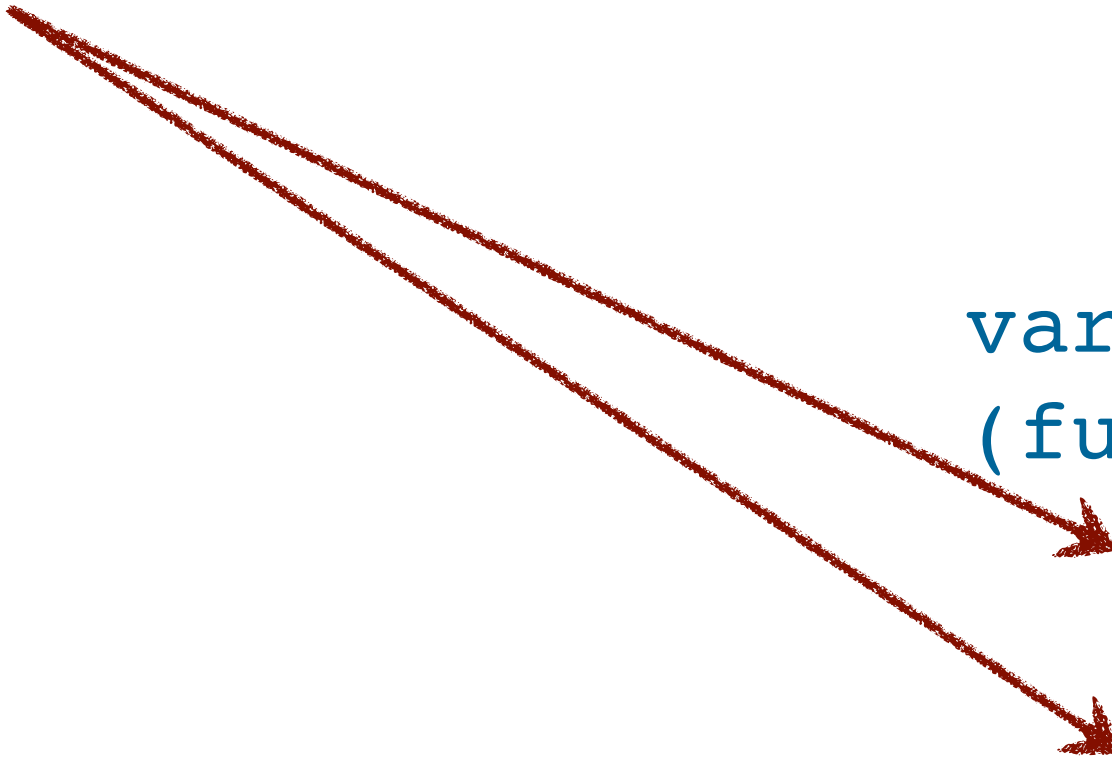


Область видимости

- Во многих языках – блочная {...}
- В JavaScript – функциональная

Variable hoisting

```
var a = 10;  
(function() {  
    console.log(a);  
    var a = 11;  
})();
```



The diagram illustrates the process of variable hoisting. Two red arrows originate from the left side of the code block on the right. The top arrow points from the space before the first line of code to the space before the first line of the transformed code. The bottom arrow points from the space before the last line of code to the space before the last line of the transformed code. This visualizes how the variable declaration is moved to the top of the scope.

```
var a = 10;  
(function() {  
    var a;  
    console.log(a);  
    a = 11;  
})();
```

Замыкания (closures)

```
var answer = 42;
```

```
var getAnswer = function() {  
    return answer;  
};
```

```
getAnswer();    // 42
```

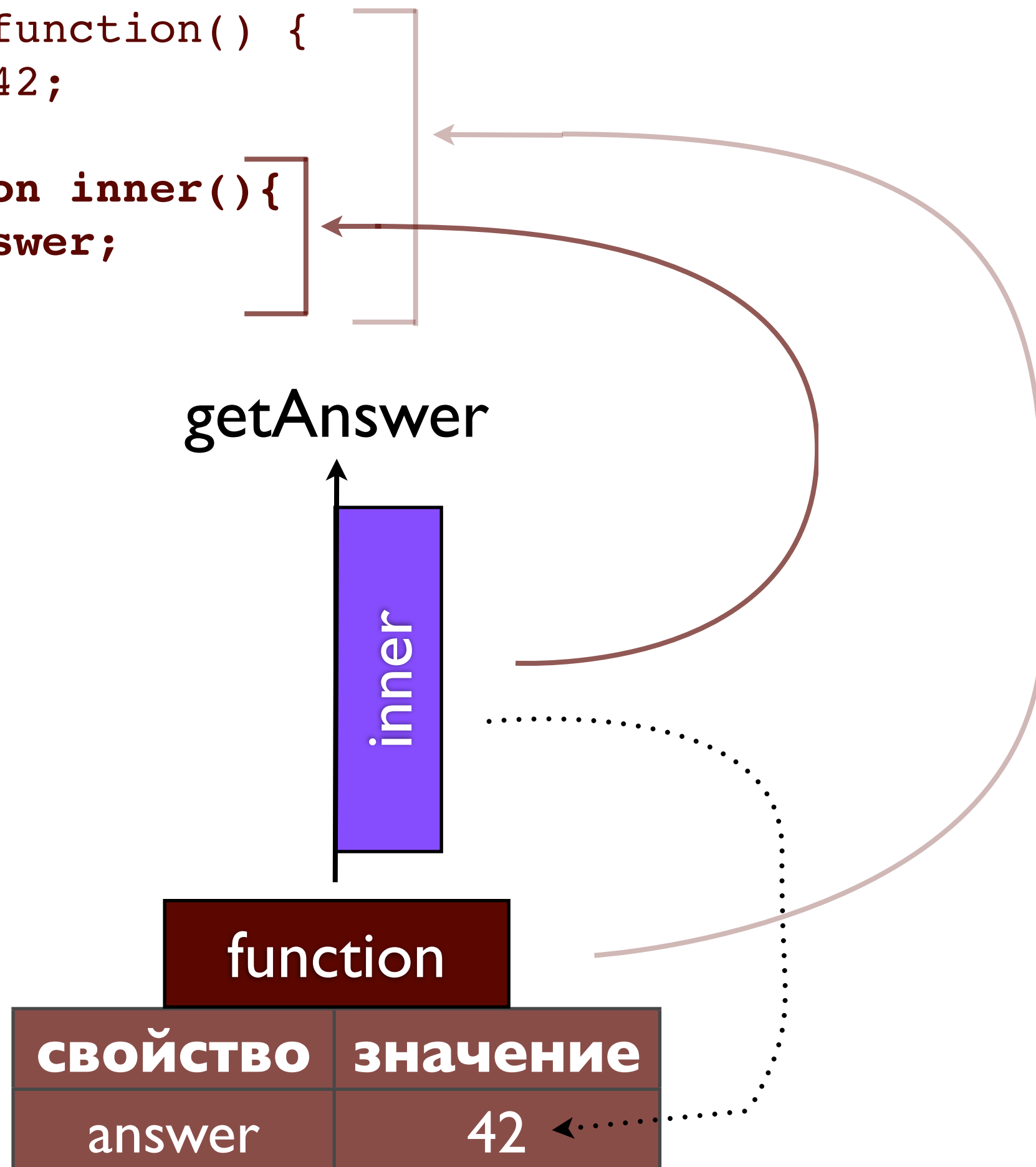
```
var getAnswer = function() {  
    var answer = 42;  
    return answer;  
};
```

```
getAnswer();    // 42
```

```
var getAnswer = (function() {  
    var answer = 42;  
  
    return function inner(){  
        return answer;  
    };  
})();
```

```
getAnswer();    // 42
```

```
var getAnswer = (function() {  
    var answer = 42;  
  
    return function inner(){  
        return answer;  
    };  
})();
```



```
var getAnswer = (function() {  
    var answer = 42;  
  
    return function inner(){  
        return answer;  
    };  
})();
```

getAnswer =

inner



свойство	значение
answer	42

Другой пример

