# Лекция №1

## Функции

# Strict mode

Строгий режим принёс ряд изменений в обычную семантику JavaScript. Во-первых, строгий режим заменяет исключениями некоторые ошибки, которые интерпретатор JavaScript ранее молча пропускал. Во-вторых, строгий режим исправляет ошибки, которые мешали движкам JavaScript выполнять оптимизацию -- в некоторых случаях код в строгом режиме может быть оптимизирован для более быстрого выполнения, чем код в обычном режиме. В-третьих, строгий режим запрещает использовать некоторые элементы синтаксиса, которые, вероятно, в следующих версиях ECMAScript получат особый смысл.

# Строгий режим для скриптов

Чтобы активизировать строгий режим для всего скрипта, нужно поместить оператор `"use strict";` или `'use strict';` перед всеми остальными операторами скрипта (выдержать приведённый синтаксис буквально).

```
// Синтаксис переключения в строгий режим всего скрипта
"use strict";
var v = "Привет! Я скрипт в строгом режиме!";
```


Функции

# Функции

Зачастую нам надо повторять одно и то же действие во многих частях программы.

Например, необходимо красиво вывести сообщение при приветствии посетителя, при выходе посетителя с сайта, ещё где-нибудь.

Чтобы не повторять один и тот же код во многих местах, придуманы функции. Функции являются основными «строительными блоками» программы.

Примеры встроенных функций вы уже видели – это `alert(message)`, `prompt(message, default)` и `confirm(question)`. Но можно создавать и свои.

# Объявление функций

## Функции вида "function declaration statement"

Объявление функции (*function definition*, или *function declaration*, или *function statement*) состоит из ключевого слова `function` и следующих частей:

- Имя функции.
- Список параметров (принимаемых функцией) заключённых в круглые скобки `()` и разделённых запятыми.
- Инструкции, которые будут выполнены после вызова функции, заключают в фигурные скобки `{ }`.

Например, следующий код объявляет простую функцию с именем `square:`

```
function square(number) {
  return number * number;
}
```

# Вызовы функций

Объявление функции не выполняет её. Объявление функции просто называет функцию и указывает, что делать при вызове функции.

**Вызов** функции фактически выполняет указанные действия с указанными параметрами. Например, если вы определите функцию `square`, вы можете вызвать её следующим образом:

```
square(5);
```

Функция может вызвать саму себя. Например, вот функция рекурсивного вычисления факториала:

```
function factorial(n) {
  if ((n === 0) || (n === 1))
    return 1;
  else
    return (n * factorial(n - 1));
}
```

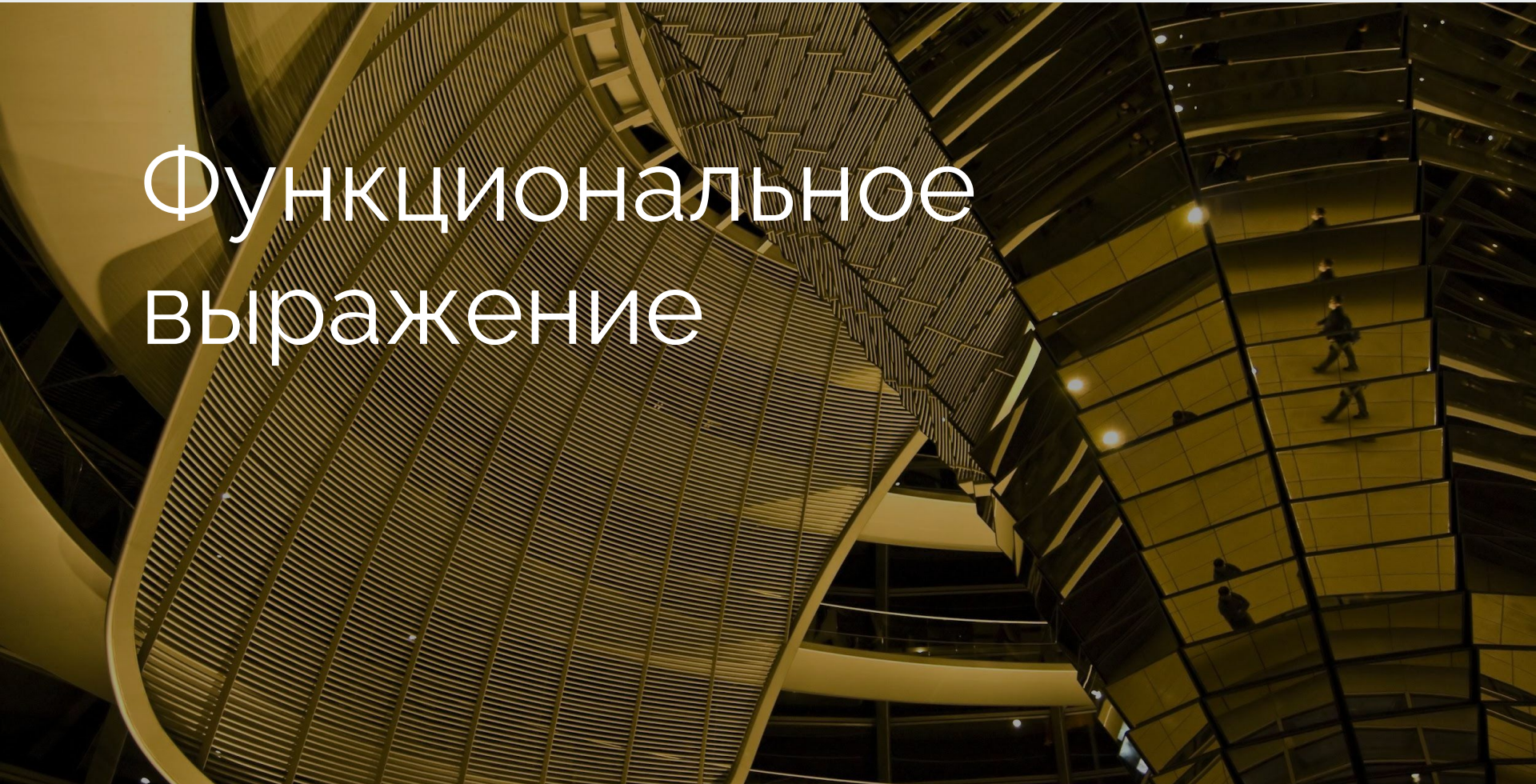
# Область видимости функций

*(function scope)*

Переменные объявленные в функции не могут быть доступными где-нибудь вне этой функции, поэтому переменные (которые нужны именно для функции) объявляют только в scope функции. При этом функция имеет доступ ко всем переменным и функциям, объявленным внутри её scope. Другими словами функция объявленная в глобальном scope имеет доступ ко всем переменным в глобальном scope. Функция объявленная внутри другой функции ещё имеет доступ и ко всем переменным её родительской функции и другим переменным, к которым эта родительская функция имеет доступ.

```
// Следующие переменные объявленны в глобальном scope
var num1 = 20,
    num2 = 3,
    name = 'Chamahk';

// Эта функция объявленна в глобальном scope
function multiply() {
  return num1 * num2;
}
```

```
// Пример вложенной функции
function getScore() {
  var num1 = 2,
      num2 = 3;

  function add() {
    return name + ' scored ' + (num1 + num2);
  }

  return add();
}

getScore(); // вернёт "Chamahk scored 5"
```



# Функциональное выражение

# Функциональное выражение

Ключевое слово **`function`** может использоваться для определения функции внутри выражения.

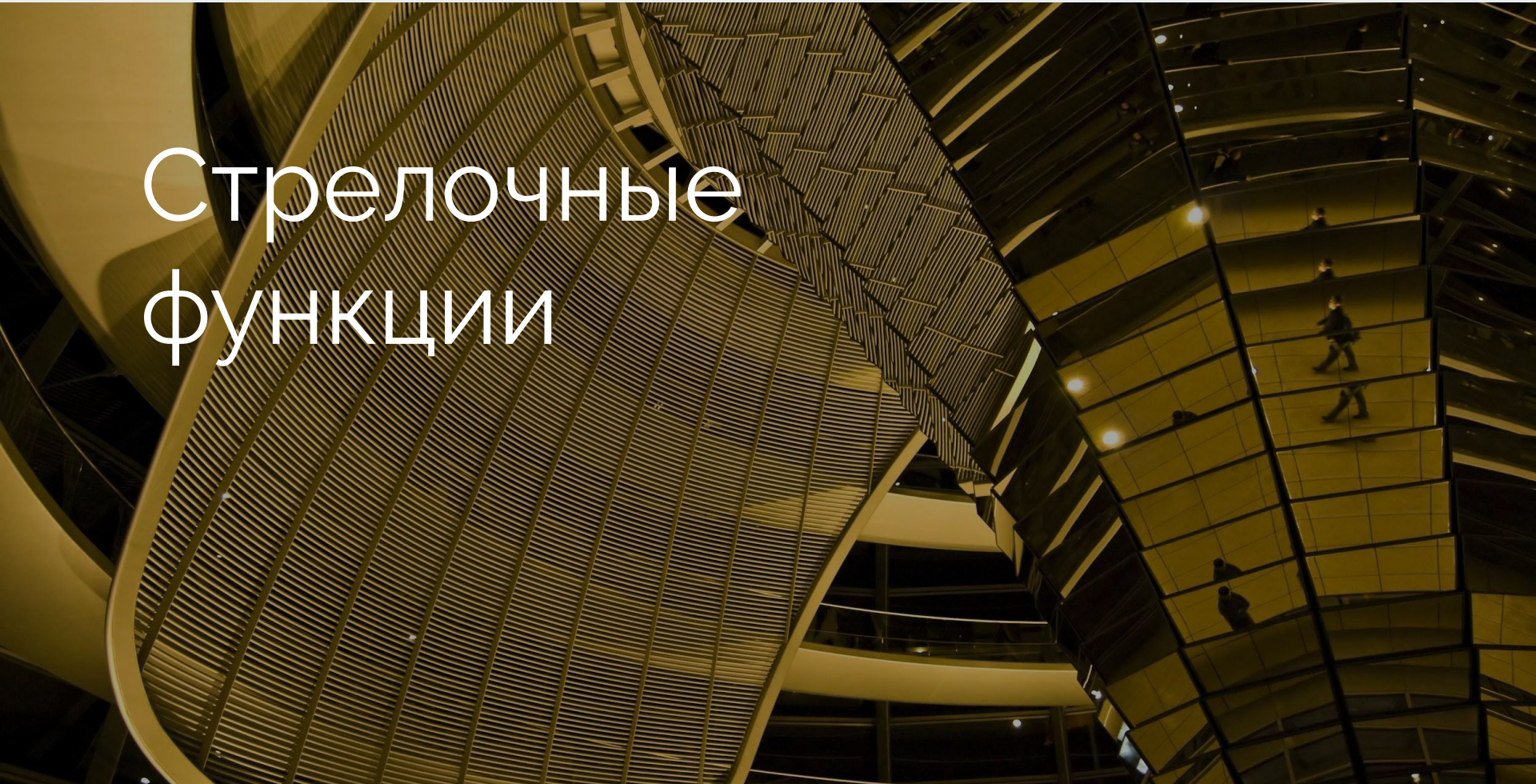
Вы можете также определять функции используя конструктор `Function` и `объявление функции`.

## Синтаксис

```
var myFunction = function [name]([param1[, param2[, ..., paramN]]]) {
   statements
};
```

# Описание

Функциональное выражение и объявление функции очень похожи и имеют почти одинаковый синтаксис. Главным отличием между ними является *имя функции*, которое в случае функциональных выражений может быть опущено для создания *анонимных* функций. Функциональное выражение может быть использовано для создания самовызывающейся функции IIFE (Immediately Invoked Function Expression), которая исполняется сразу же после того, как она была определена. Более подробная информация изложена в разделе о функциях.



# Стрелочные функции

# Стрелочные функции

## Сводка

**Выражения стрелочных функций** имеют более короткий синтаксис по сравнению с функциональными выражениями и лексически привязаны к значению this (но не привязаны к собственному this, arguments, super, или new.target). Выражение стрелочных функций не позволяют задавать имя, поэтому стрелочные функции анонимны, если их ни к чему не присвоить.

# Базовый синтаксис

```
(param1, param2, …, paramN) => { statements }
(param1, param2, …, paramN) => expression
// эквивалентно: (param1, param2, …, paramN) => { return expression; }

// Круглые скобки не обязательны для единственного параметра:
(singleParam) => { statements }
singleParam => { statements }

// Функция без параметров нуждается в круглых скобках:
() => { statements }
() => expression
// Эквивалентно: () => { return expression; }
```

# Полезные ссылки

https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Strict_mode
https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Functions
https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/function#syntax
https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Functions/Arrow_functions

# Спасибо!