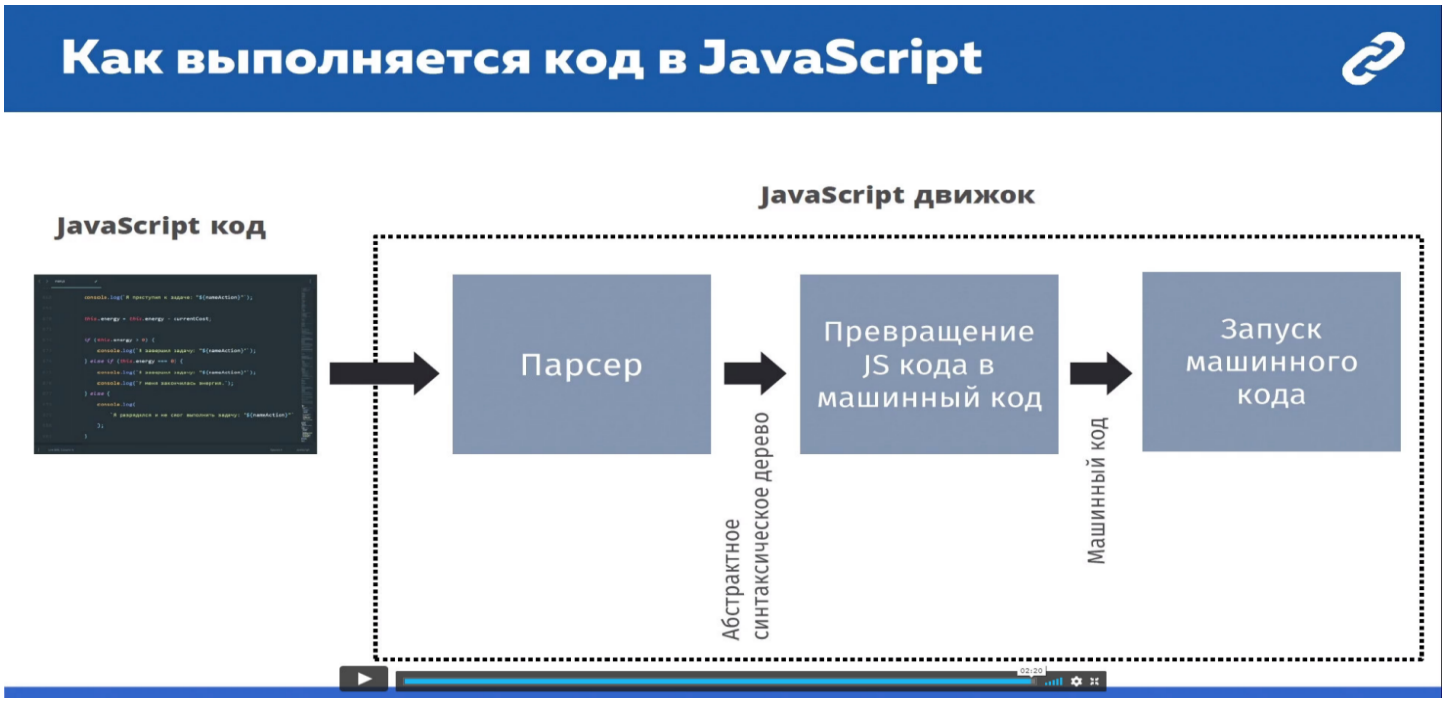
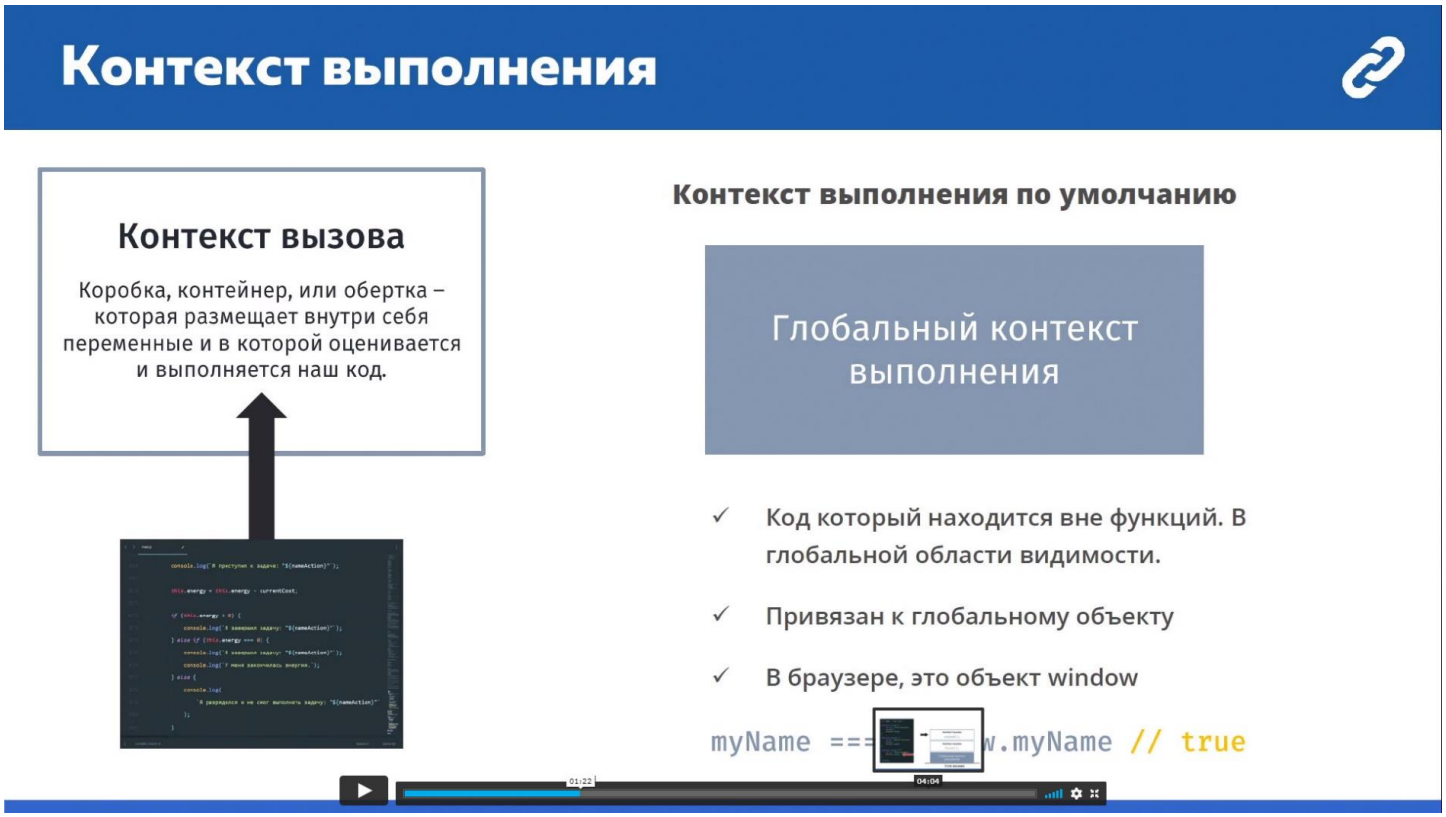
Контекст, области видимости и др.

Как выполняется код в JS



Код JS всегда выполняется в каком-либо окружении, н-р, браузер, Node.js. Когда запускаем код, он попадает в **JS-движок** (программа, которая анализирует код, запускае, выполняет), н-р, V8 для Chrome. Сначала код попадает в **парсер** и проверяет на синтаксические ошибки. Далее, код превращается в **абстрактное синтаксическое дерево**, потом код превращается в **машинный код**, потом **выполняется**.

Контекст выполнения



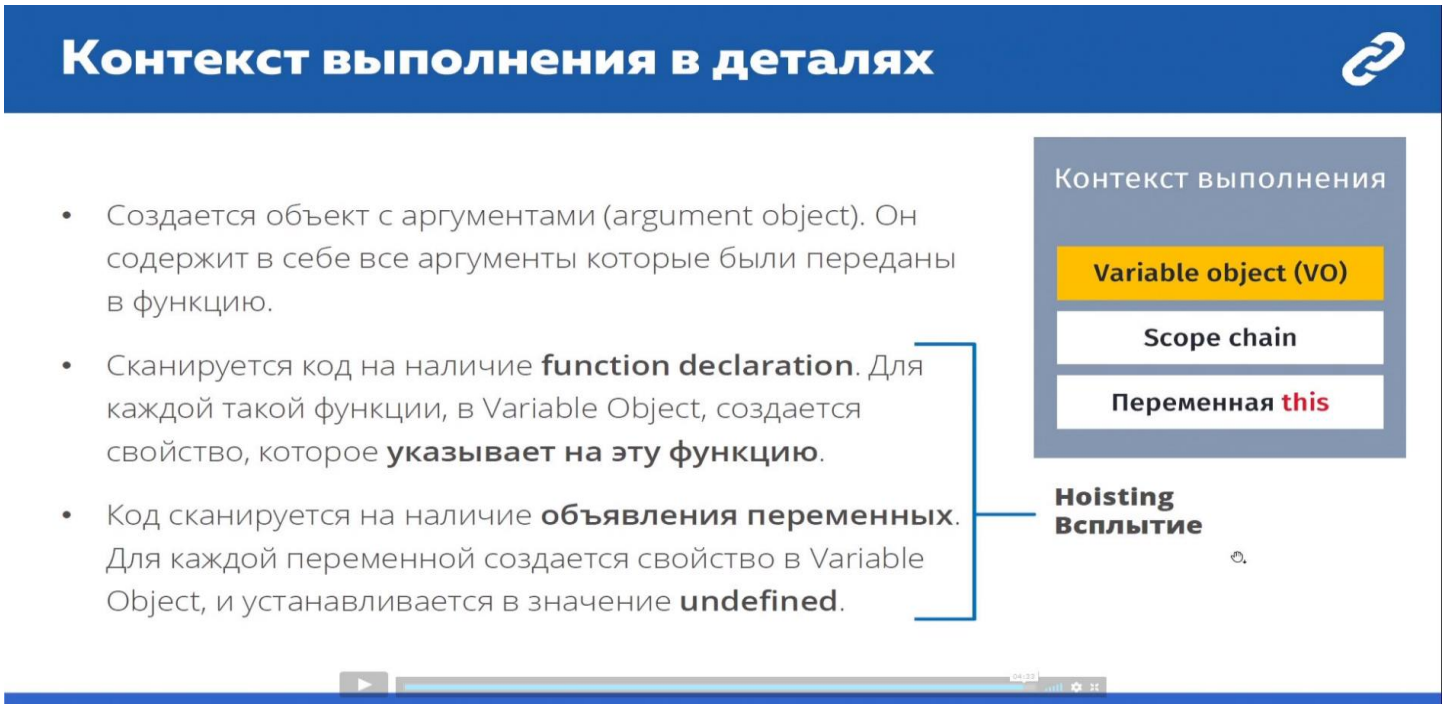
Любой JS код при выполнении находится в определенном окружении – **контексте выполения.** По умолчанию это **глобальный контекст выполнения**. В него входят все переменные и функции, которые не находятся в других функциях. Они привязаны к **глобальному объекту** – в браузере это window.

При выполнении функций создается новый контекст выполнения, который привязан к этой функции. После того, как функция отработала, ее контекст удаляется.

Контекст выполнения в деталях:

1**. Фаза создания**

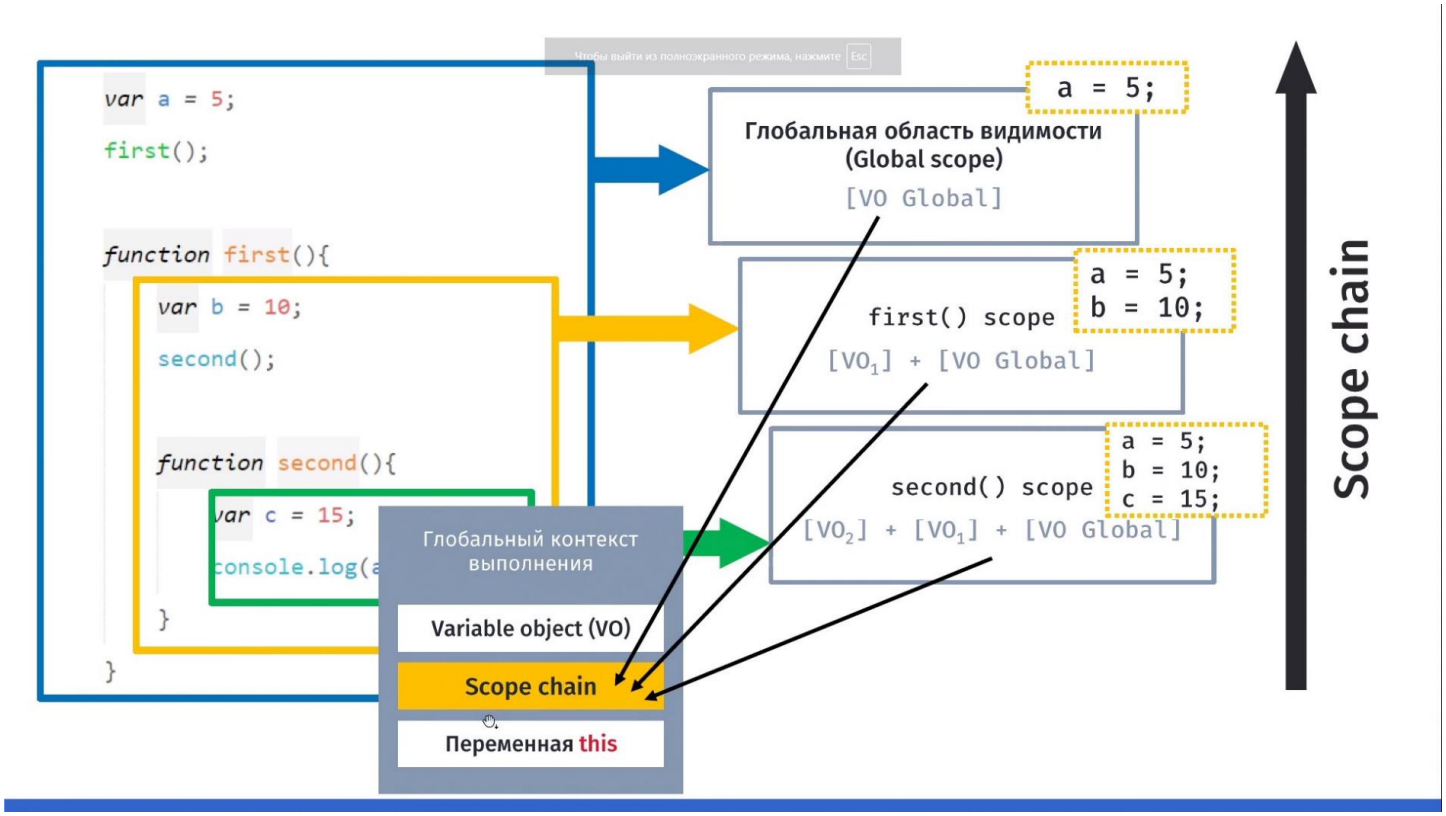
A) Создается Variable Object (VO) – объект, который содержит в себе **аргументы функции**, **переменные** (им устанавливается значение **undefined**) внутри функции и внутренние функции **function declaration.**



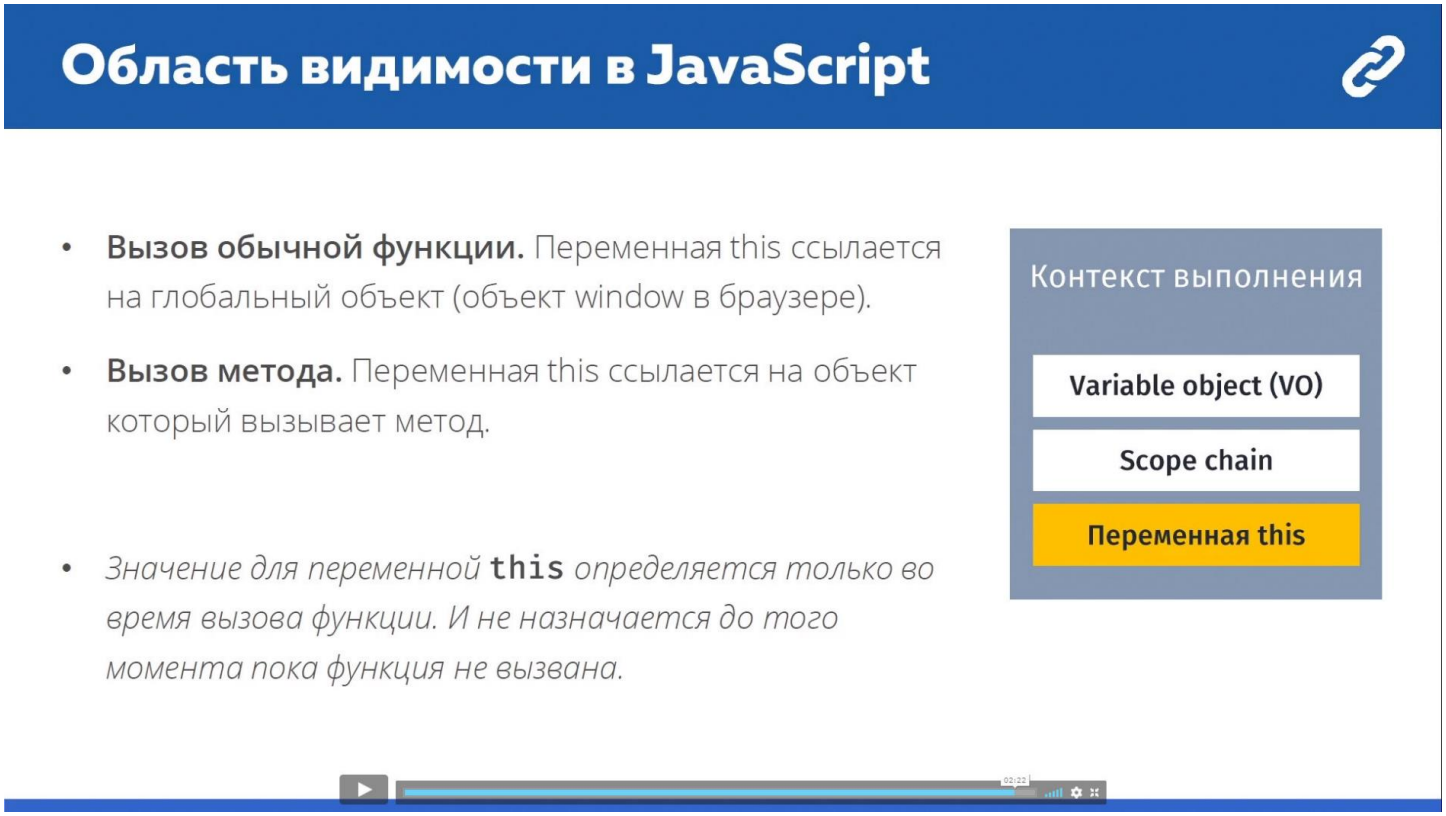
B) Создается Scope Chain – к каким переменным (variable object) мы имеем доступ. Особенности:

- Область видимости определяется **лексически** – то есть, где описана наша функция
- JS использует область видимости, **основанную на функциях** (не скобками кода)
- Вложенные функции создают **цепь областей видимости**

Hoisting – всплытие переменных. Код видит переменные, которые еще не объявлены, но переменные уже есть со значением **undefined,** ошибки нет



C) Определяется переменная this



- Заметим, что при вызове метода this ссылается не на тот объект, в котором описан, а тот, который вызывается
- Значение this определяется только в момент вызова функции.
- Это не функция!
- Ссылается на объект
- Привязка значения к this может быть неявной и устанавливаться интерпретатором JS или быть явной и устанавливаться Вами.

Примеры:

```
"use strict";
// 1 пример
var function1 = function(){
   console.log("function1", this);
}
function1();
// function1 Window {window: Window, self: Window, document: document, name: '', location: Location, …}
// При "use strict";
function1 undefined

var function2 = function(){
    function1();
}
function2();
// function1 Window {window: Window, self: Window, document: document, name: '', location: Location, …}
// При "use strict";
function1 undefined

// 2 пример
var object1 = {
    name: "Object 1",
    fun: function(){
       console.log(this);
      function function3 (){
          console.log('function3', this);
       }
       function3();
    }
 }
object1.fun();
// {name: 'Object 1', fun: f}

function3 Window {0: Window, window: Window, self: Window, document: document, name: '', location: Location, …}
// При "use strict";
Function3 undefined

 var object2 = {
    name: "Object 2",
    fun2: object1.fun
 }
 object2.fun2();
{name: 'Object 2', fun2: f}

function3 Window {0: Window, window: Window, self: Window, document: document, name: '', location: Location, …}
// При "use strict";
Function3 undefined

// 3 пример
'use strict';
 var function1 = function () {
    console.log("function1", this);
 }
function1();

window.action = function1;
window.action();
//При "use strict";
function1 undefined
 function1 Window {0: Window, window: Window, self: Window, document: document, name: '', location: Location, …}

// 4 пример
'use strict';
var function1 = function () {
    console.log("function1", this);
 }
```

```
setTimeout(function1, 1000);
//При "use strict";
// function1 Window {0: Window, window: Window, self: Window, document: document, name: '', location: Location, …}
Так как setTimeout является методом windows

var object1 = {
   name: "Object 1",
   fun: function(){
      console.log(this);
   }
}
setTimeout(object1.fun, 1000);
//Window {0: Window, window: Window, self: Window, document: document, name: '', location: Location, …}
```

2. **Фаза выполения**. Код запускается.