Scope and Lexical Environment

Использование инструкции 'use strict‘

Строгий режим был введен в ECMAScript 5, и старые браузеры (IE9 и старше) его не поддерживают.

В строгом режиме:

- некоторые ошибки можно найти быстрее;
- более опасные и не полезные черты JavaScript либо запрещены, либо приводят к ошибке.

'use strict' используется:

- в начале файла – тогда ее действие распространяется на файл;
- в начале функции – ее действие распространяется в пределах функции.

Например, одним из назначений использования инструкции 'use strict' есть то, что нельзя объявлять переменную без ключевых слов var, let, constant.

```
"use strict";
person = "Douglas Crockford"; //  Uncaught ReferenceError
console.log(person);
"use strict";
function test(){
   a = 33; // ReferenceError: a is not defined
   console.log(a);
}
test();
```

```
b = 10;

console.log(b); // 10

function test(){
  "use strict";
   a = 33; // ReferenceError: a is not defined
   console.log(a);
}

test();
```

Область видимости (Scope) переменной

Это область памяти из которой мы можем получить доступ к переменной.

Есть глобальная область видимости и локальные области видимости

Соответственно, глобальные переменные – переменные и функции, которые

- не находятся внутри какой-то функции;
- не находятся блока, ограниченного { }

В JavaScript все глобальные переменные и функции являются свойствами специального объекта, который называется глобальный объект, который явно доступен под именем window.

```
var a = "Bill";
function aTest() {
        console.log(name);
}
console.log(window); // в объекте window найдем a, aTest

function aTest() {
        var a = "Bill";
        console.log(a);
}
console.log(window); // в объекте window найдем только aTest
```

Так как переменная `var a = "Bill";` объявлена внутри функции то она не попала в глобальный объект window.

Локальные переменные – определяются:

- в ES 5 пределами function(){}
- в ES 6 пределами function(){} и блоками ограниченными скобками {} - для переменных, объявленных использованием ключевого слов

Локальные переменные доступны только внутри функции или в блоке.

Переменная, объявленная внутри этих скобок существует только внутри этих скобок, и после выхода кода за закрывающую скобку } функции переменная уничтожается сборщиком мусора

(Garbage collection).

Local scope можно определить фигурными скобками {}

Local scope также определяют конструкции ветления и цикла

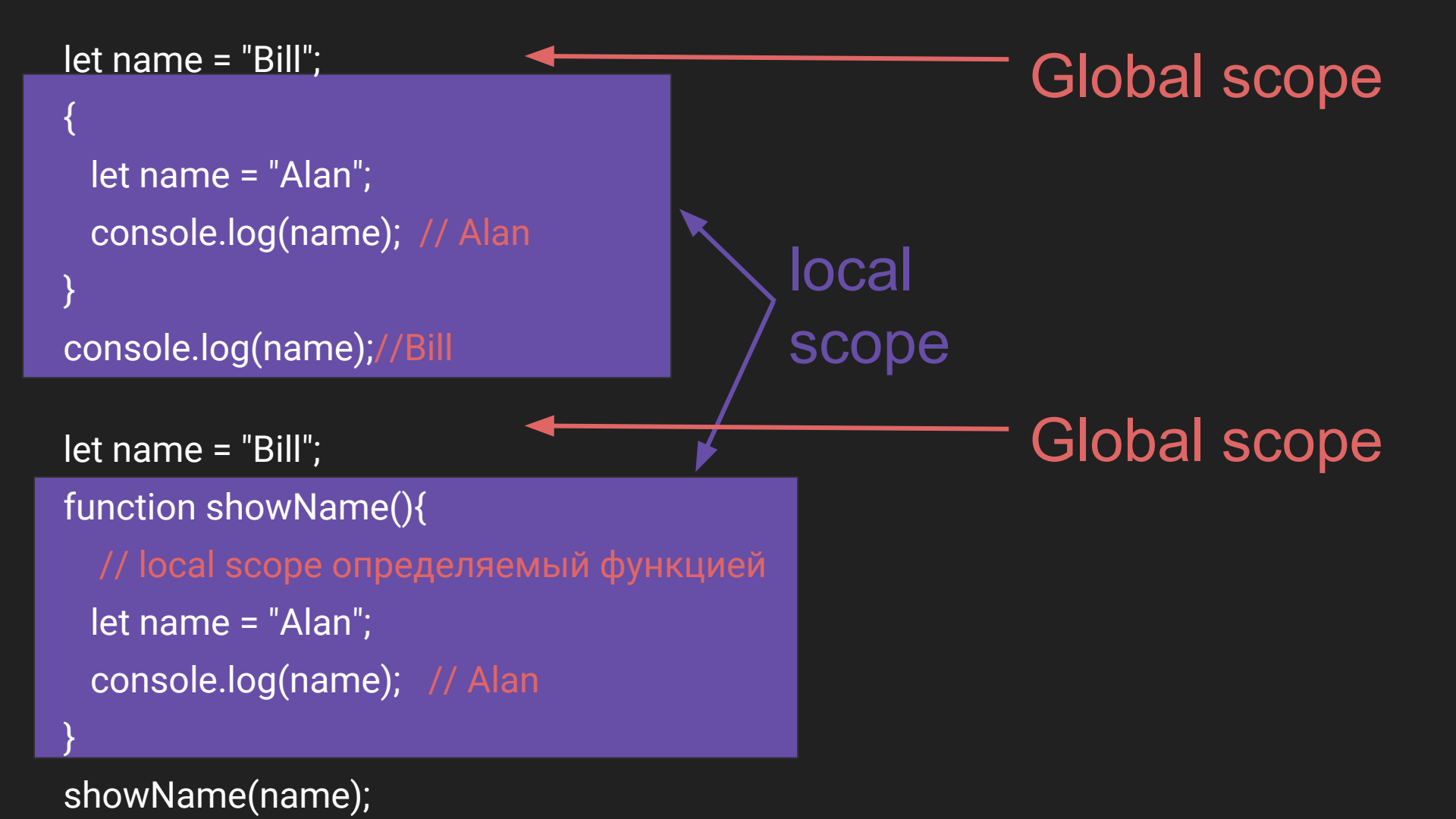
- for, while, do-while, switch, if...

Local scope также определяют function(){}

Зачем нужны локальные переменные ?

Одна из причин - экономия памяти.

После того как функция была вызвана и отработала все локальные переменные уничтожаются освобождая оперативную память.


let name = "Bill";
{
let name = "Alan";
console.log(name); // Alan
}
console.log(name);//Bill
Global scope
local
scope
let name = "Bill";
function showName(){
// local scope определяемый функцией
let name = "Alan";
console.log(name); // Alan
}
showName(name);
Global scope

Nested local scope

```
let name = "Bill";
{
    let name = "Alan";
    {
      let name = "Arnold";
      console.log(name); // Arnold
    }
    console.log(name); // Alan
}
console.log(name); //Bill
```

```
let name = "Bill";
function one(){
  let name = "Alan";
  function two(){
    let name = "Arnold";
     console.log(name);
  }
  two(); // Arnold
}
one();
```

Lexical Enviroment

Это место, где физически размещен написанный код.

```
function test(){
  var a = "Hello";
}
```

Например – переменная a физически размещается в функции test.

Интерпретатор JavaScript при переводе кода в инструкции (компиляции кода) для машины, размещает сущности (переменные, функции, объекты) в памяти, исходя из того где они указаны в коде. Эти области памяти называются [[Scope]]

И это очень важно с точки зрения взаимодействия этих всех сущностей между собой.

В JavaScript scope создаются также функциями.

Функция всегда объявляется в определённом контексте (в нашем случае это window)

В свойство f.[[Scope]] записывается ссылка на контекст, то есть f.[[Scope]] = window

Это свойство привязывается к функции один раз – в момент ее создания, далее его нельзя изменить, доступа к нему тоже у программиста нет.

var, let, const

```
{
  var a = 20;
}
console.log(a) // 20
```

```
{
  let a = 20;
}
console.log(a) // Refference Error
```

!!! Внимание - если мы используем для объявления переменной ключевое слово let или const то мы не можем в рамках одной области видимости определить более одной переменной с одинаковым именем (a с var можем)

```
let name = "Brendan Eich";
let name = "Douglas Crockford"; // Uncaught SyntaxError:
```

Scope chain - цепочка областей видимости

```
let name = "Bill";
function show(){
    function log(){
    console.log(name);
    }
    log();
}
show(); // Bill
```

Переменную name объявили с использованием let

Из вложенного scope переменная name доступна

Переменную name никак не объявили, но пытаемся к ней обратиться внутри функции

```
"use strict";
function show(){
        function log(){
                name = "Bill";
        }
        log();
}
// здесь еще нет переменной name
console.log(name); // Refference Error -> not name
show(); // переменная name была создана в функции
console.log(name); // Bill
console.log(window.name); // Bill
```

ВЫВОД Нужно всегда объявлять переменные с использованием одного из ключевых слов var, let или const.

Тогда можно быть уверенным, что интерпретатор не создаст нам ненужных переменных (следующий слайд).

```
function show(){
    function log(){
        var name = "Bill";
    }
    log();
}
show();
console.log(name); // Refference Error -> not name (Node.js) браузер нет
```

Если посмотреть на текст программы приведенной ниже, то можно однозначно сказать, что будет выведено в консоль.

```
let name = "Bill";

function show(){

   console.log(name);

}

show(); // Bill.
```

В этом коде что выведется ? Откуда функция show() будет брать переменную name.

```
let name = "Bill";
function show(){
    console.log(name);
}
function log(){
    let name = "Alan";
    show();
}
log(); // Bill
```

ЗАПОМНИТЬ !

Поиск переменной проводится согласно lexical scope.

То есть в нашем случае функция show() берет переменную name оттуда, где она определена, а не оттуда, откуда она вызывается.