Язык JavaScript (основы: области видимости, управление памятью и замыкание)

ОБЛАСТИ ВИДИМОСТИ

Все переменные в JavaScript имеют определенную область видимости, в пределах которой они могут действовать (быть доступными).

В языке JavaScript выделяют 2 области видимости:
- глобальная (переменная или функция, созданная в этой области видимости, может быть доступна из любой точки программы);
- локальная (переменная или функция, созданная в этой области видимости, может быть доступна только внутри неё).

ОБЛАСТИ ВИДИМОСТИ

До стандарта ECMAScript 2015:

- все переменные, которые объявлены вне функций, являются глобальными и становятся свойствами и методами глобального объекта window;
- переменная, определенная внутри функции, является локальной.

```
var x = 5; //глобальная переменная доступна через window.x
//глобальная функция, доступна через window.a()
function displaySquare(y){ /* y – локальная переменная
   функции, живёт пока выполняется функция */
   console.log(x * y);
}
displaySquare(3);
```

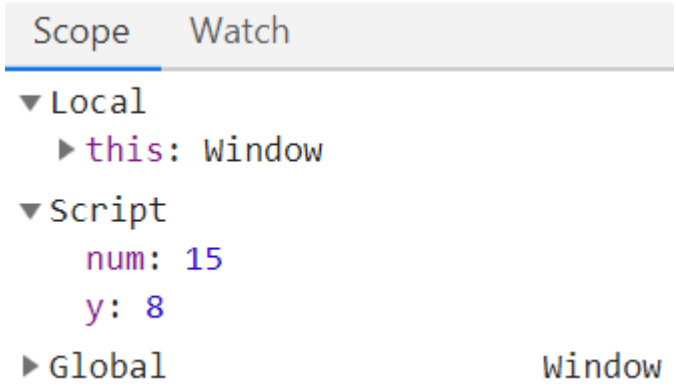
ОБЛАСТИ ВИДИМОСТИ

Со стандартом ECMAScript 2015 (при использовании оператора let или const):
- глобальные переменные (let и const) определены в своей области видимости (область видимости скрипта) и не становятся свойствами глобального объекта;
- каждый блок кода (заданный фигурными скобками) определяет новую область видимости, в которой существует переменная (относится к let и const, а также и к функциям при включенном режиме "use strict";).

[[SCOPE]]

[[Scope]] - это скрытое внутреннее свойство функции. Данное свойство содержит ссылку на те области видимости, в которой данная функция была объявлена. При вызове функции к [[Scope]] добавляется локальная область видимости.
Пример (останов внутри функции в отладчике):

```
12 // global scope (глобальная область видимости)
13 let num = 15;
14 function outputNum() {
15     console.log(num);
16 }
17 outputNum(); // 15
```



[[SCOPE]]

Пример просмотра [[scope]] в командной строке:

```
> console.dir(outputNum)
```

VM1936:1

```
▼ ƒ outputNum()
    arguments: null
    caller: null
    length: 0
    name: "outputNum"
  ▶ prototype: {constructor: ƒ}
  ▶ __proto__: ƒ ()
    [[FunctionLocation]]: script.js:14
  ▼ [[Scopes]]: Scopes[2]
    ▶ 0: Script {num: 15}
    ▶ 1: Global {postMessage: ƒ, blur: ƒ, focus: ƒ, close: ƒ, parent: Window, …}
```

[[SCOPE]]

JavaScript **всегда начинает поиск переменной или функции с текущей области видимости**. Если она в ней не будет найдена, то интерпретатор переместится **к следующей области**, указанной в **[[Scope]]**, и попробует отыскать её там. После этого **действия повторяются** пока не будет найдено искомое имя переменной или функции, или пока интерпретатор не дойдёт до глобальной области видимости. **Глобальная область видимости** - это последнее звено в цепочке областей видимости (объект window). Она не содержит ссылку на другую область, дальше неё ничего нет. Если интерпретатор не нашёл нужное имя переменной или функции, то в этом случае он бросает ошибку о том, что данный идентификатор не определён.

УПРАВЛЕНИЕМ ПАМЯТЬЮ

Главной концепцией управления памятью в JavaScript является принцип достижимости (англ. reachability).

Определённое множество значений считается достижимым изначально, в частности:

- Значения, ссылки на которые содержатся в стеке вызова, то есть – все локальные переменные и параметры функций, которые в настоящий момент выполняются или находятся в ожидании окончания вложенного вызова.

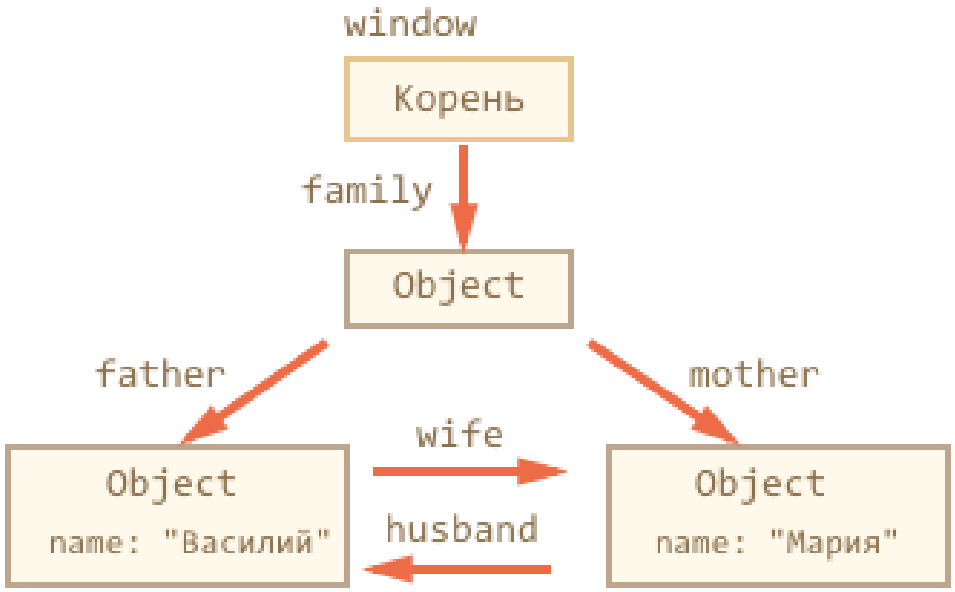
- Все глобальные переменные.
Эти значения гарантированно хранятся в памяти. Мы будем называть их корнями.

- Любое другое значение сохраняется в памяти лишь до тех пор, пока доступно из корня по ссылке или цепочке ссылок.

УПРАВЛЕНИЕМ ПАМЯТЬЮ

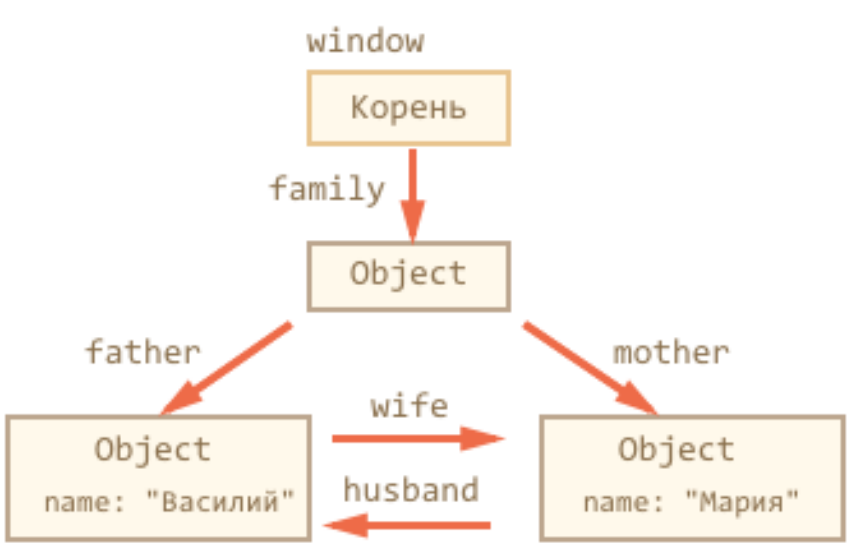
Для очистки памяти от недостижимых значений в браузерах используется автоматический Сборщик мусора.

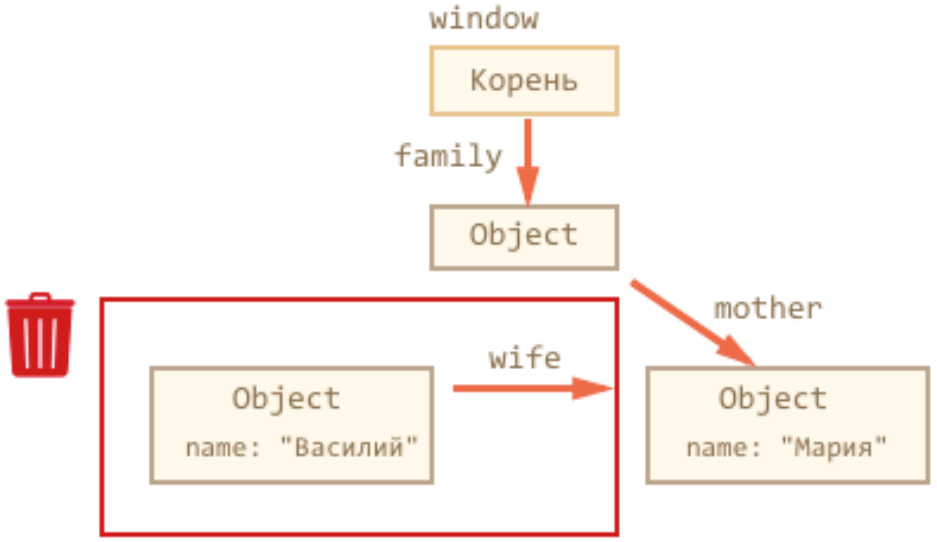
```
1  function marry(man, woman) {
2    woman.husband = man;
3    man.wife = woman;
4
5    return {
6      father: man,
7      mother: woman
8    }
9  }
10
11 var family = marry({
12   name: "Василий"
13 }, {
14   name: "Мария"
15 });
```



УПРАВЛЕНИЕМ ПАМЯТЬЮ

```
1  delete family.father;
2  delete family.mother.husband;
```





ЗАМЫКАНИЯ

В JavaScript функции могут находиться внутри других функций.

Когда одна функция находится внутри другой, то внутренняя функция имеет доступ к области видимости (окружению) внешней функции.

Этот способ организации кода в JavaScript позволяет создавать **замыкания**.

ЗАМЫКАНИЯ (ПРИМЕР)

```
function outName(name, lastname){
    function getName(){
        return name + " " + lastname;
    }

    return getName;
}

let getName = outName("Иван", "Иванов");
console.log(getName());
console.dir(getName);
```

ЧТО ПОЧИТАТЬ ДОПОЛНИТЕЛЬНО

https://learn.javascript.ru/var - Устаревшее ключевое слово "var“
https://learn.javascript.ru/global-object - Глобальный объект
https://learn.javascript.ru/garbage-collection - Сборка мусора
https://learn.javascript.ru/closure - Замыкание