**ОБЛАСТЬ ВИДИМОСТИ И ЗАМЫКАНИЕ**

Область видимости (Scope)

В JavaScript существуют две области видимости: глобальная и локальная.  
Переменные определенные внутри функции, оказываются в локальной области видимости, а переменные определенные вне функции – в глобальной области видимости.

Каждая функция, после ее вызова, создает новую локальную область видимости.

Глобальная область видимости (Global Scope)

Все, что вы пишете вне функции, находится в глобальной области видимости. То есть, вам не нужно создавать глобальную область видимости при помощи какой-то конструкции, вы находитесь в ней сразу, как только начинаете писать свой код.

var language = "JavaScript"; // по умолчанию, область видимости - глобальная

Переменные внутри глобальной области видимости доступны в любой части кода и другой области видимости.  
Глобальная область видимости «живет» столько, сколько «живет» приложение.

var language = "JavaScript";

console.log(language); // "JavaScript"

function show() {

console.log(language); // здесь есть доступ к переменной "language"

}

show(); // "JavaScript"

Локальная область видимости (Local Scope)

Переменные определенные внутри функции находятся в локальной области видимости. И, так как, при каждом вызове функции, создается новая локальная область видимости, то одна и та же переменная, может использоваться в разных функциях с разными значениями. Это возможно благодаря тому, что переменные, которые находятся в локальной области видимости, не доступны в любом другом месте.

// Глобальная область видимости

function someFunction() {

// Локальная область видимости №1

function someOtherFunction() {

// Локальная область видимости №2

}

}

Локальная область видимости «живет» только с момента вызова функции и до конца ее выполнения.

Контекст (Context)

Контекст используется для ссылки на значение this в конкретной части кода.  
Область видимости ссылается на видимость переменных, а контекст на значение this в этой же самой области видимости. То есть this – это в контексте чего вызывается функция.  
В глобальной области видимости контекстом всегда является объект Window.

console.log(this); // Window { ... }

function functionThis() {

console.log(this);

}

functionThis(); // так же Window { ... }, поскольку функция не является свойством объекта (object)

Если вызывать функцию в строгом режиме, то контекстом будет undefined.

'use strict'

function strictThis() {

console.log(this);

}

strictThis(); // undefined

Если функция является методом объекта (функции, которые являются свойством объекта, называются методами), то контекстом будет уже не объект Window, а объект, частью которого является данная функция.

var object = {

language: "JavaScript",

returnLanguage: function() {

console.log(this.language);

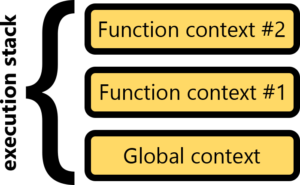
}

};

object.returnLanguage(); // в this подставляется object, по этому - "JavaScript"

То есть в this подставляется объект в контексте которого выполняется метод.

Контекст исполнения (Execution Context)

Контекст исполнения – ссылается на область видимости, а не на контекст.  
Контекст исполнения – понятие, которое связывает воедино все рассмотренные нами прежде понятия. Это некая сущность, которая содержит все необходимое для запуска каждой отдельно взятой функции (переменные, аргументы функции, области видимости, this).  
Когда интерпретатор JavaScript начинает выполнять код, контекст (область видимости) по умолчанию является глобальным. То есть контекст this – Window (в use strict – undefined), область видимости – глобальная. Изначально контекст исполнения для всего кода – глобальный контекст, он автоматически «прикрепляется» к контексту исполнения при начале выполнения кода.  
После этого, когда функция вызывается и исполняется, она может «прикрепить» собственный контекст к контексту исполнения. То же самое происходит, когда одна функция вызывается внутри другой функции или где-либо еще.  
Каждая функция создает свой собственный контекст исполнения. Глобальный контекст может быть только один, но контекстов функции может быть сколько угодно. Во время создания функциями своих контекстов, все они попадают в стек исполнения () по правилу последний созданный помещается наверх и первым же выполняется. После выполнения он убирается из стека и выполняется следующий.  
  
Контекст исполнения имеет две фазы: создания и выполнение кода.  
Фаза создания это первая фаза, которая начинается во время вызова функции, но ее код еще не выполняется.  
На этом этапе происходят три ключевые вещи:

* • создание объекта переменных (объект активации) (Variable (Activation) Object);
* • создание цепочки областей видимости (Scope Chain);
* • присваивание значение контексту (value of the context).

Объект переменных (Variable Object) содержит все переменные, функции и другие декларации (объявленные конструкции), определенные в конкретном контексте исполнения. Когда функция вызывается, интерпретатор сканирует код на декларации, принадлежащие этой функции. Все это помещается в один объект, который и называется объектом переменных.

"variableObject": {

// содержит аргументы функции, внутренние переменные и function declarations

}

Цепочка областей видимости (Scope Chain) создается после создания объекта переменных и содержит его в себе. Другими словами, содержит в себе свой объект переменных и объекты родительских функций.

"scopeChain": {

// содержит собственный объект переменных и другие объекты переменных из родительских контекстов исполнения

}

Цепочка областей видимости, грубо говоря, используется для поиска переменных. Помните, что переменные определенные внутри функции, снаружи не видны, мы не можем получить их значение, но получить доступ к переменным, которые объявлены вне функции, уровнем выше, мы можем. Все это благодаря цепочке областей видимости. Например, есть две объявленные функции, одна вложена в другую. В родительской функции есть переменная «а», но обращаемся мы к этой переменной во вложенной функции.

// Глобальная область видимости

console.log(a); // Uncaught ReferenceError: a is not defined

function outer(param) {

// область видимости функции outer

var a = param;

function nested() {

// область видимости функции nested

console.log(a);

}

nested();

}

outer(5); // 5

Здесь происходит следующее, мы обращаемся к переменной «а» во вложенной функции «nested», интерпретатор не находит ее в локальной области видимости этой функции и благодаря цепочки областей видимости, начинает подниматься вверх, в родительские области видимости, и ищет необходимую переменную. Как только переменная будет найдена, поиск прекращается, и мы получаем значение этой переменной. Если же переменная не найдена, то вернется значение undefined. Поиск по цепочке областей видимости может продолжаться до глобальной области видимости, это конечная точка поиска. В случае, если переменная находится в этой же функции, то поиск по цепочке областей видимости производится не будет. Это позволяет определять переменные с одинаковыми именами, не боясь переопределить такую же переменную в глобальной области видимости.  
Поиск по цепочке областей видимости возможен благодаря тому, что Scope Chain содержит в себе не только собственный объект переменных, но и родительские объекты переменных.  
Присваивание значение контексту означает, что в this подставляется нужный контекст.  
Как результат первой фазы, создания контекста исполнения, создается объект контекста исполнения (Execution Context Object), который можно представить, как абстрактный объект:

executionContextObject = {

"scopeChain": {},

"variableObject": {},

"this": valueOfThis

}

Фаза выполнения кода означает выполнение JavaScript кода.

Лексическая область видимости

Лексическая область видимости понятие, которое означает, где внутренние функции имеют доступ к переменным их родительской области видимости, то есть к тем переменным, что находятся в функциях уровнем выше.  
Дочерние, вложенные, функции лексически связаны с контекстом исполнения их родителя. Ранее это уже рассматривалось в примере про цепочку областей видимости. Вот еще один пример:

function grandfather() {

var name = "John";

// likes не доступно здесь

function parent() {

// name доступно здесь

// likes не доступно здесь

function child() {

// name доступно здесь

var likes = "JavaScript";

}

}

}

Как видно, можно получить доступ в переменным, что находятся уровнем выше, но наоборот, получить доступ с верхних уровней к переменным, что находятся уровнем ниже, невозможно.

Замыкание (Closure)

Концепция замыкания близка к лексической области видимости. Замыкание создается, когда внутренняя функция, пытается получить доступ к цепочке области видимости внешней функции, то есть к переменным вне лексической области видимости. Замыкание содержит собственную цепочку области видимости, цепочку области видимости ее родителя и глобальную область видимости.  
Благодаря замыканию можно получить доступ не только к переменным определенным во внешней функции, но и к аргументам внешней функции.  
Также, замыкание дает возможность получить доступ к переменным внешней функции даже после того, как внутренняя функция будет возвращена. После возвращения внутренней функции, доступ ко всем ресурсам внешней функции, сохраняется.  
Когда возвращается внутренняя функция из функции, то возвращенная функция не вызовется автоматически после вызова внешней функции. Прежде всего необходимо сохранить вызов внешней функции в переменную, а после этого вызвать ее (переменную) как функцию.

function greet() {

var name = "John";

return function () {

console.log("Hi " + name + "!");

}

}

greet(); // ничего не произойдет

// возвращенная функция из greet() сохранена в greetLetter

var greetLetter = greet();

// вызов greetLetter вызовет возвращенную функцию из функции greet()

greetLetter(); // "Hi John!"

Следует запомнить, что функция «greetLetter» может получить доступ к переменной «name», которая принадлежит функции «great», даже функция «great» была выполнена. Можно сказать, что переменная «name» осталась в замыкании.  
Возвращаемую функцию можно вызвать и без присвоения в переменную используя конструкцию из двухразового указания пары круглых скобок «()()».

greet()(); // "Hi John!"

Блочные конструкции (Block Statements)

Блочные конструкции, такие как «if», «for» и т.п. (кроме функций), не создают новую область видимости. Переменные, объявленные внутри блочной конструкции при помощи ключевого слова «var», видны везде.  
Но, переменные объявленные при помощи ключевого слова «let» или «const», внутри таких блочных конструкций, не будут видны вне них. Такие переменные «создают» свою блочную область видимости.

if (true) {

var name = "John";

let language = "JavaScript";

const framework = "React";

}

console.log(name); // "John"

console.log(likes); // Uncaught ReferenceError: language is not defined

console.log(skills); // Uncaught ReferenceError: framework is not defined

Операторы «let» и «const» позволяет объявить локальную переменную с областью видимости, ограниченной текущим блоком кода.