Java-Success.com

Industrial strength Java Career Companion

Register | Login | Logout | Contact Us

search here … Go

Home | Java FAQs | 600+ Java Q&As | Career | Tutorials | Member | Why?

Can u Debug? | Java 8 ready? | Top X | Productivity Tools | Judging Experience?

Home › Interview › Other Interview Q&A for Java Developers › JavaScript Interview Q&A › JavaScript Top Interview Q&A › ♥ Q1 – Q10 JavaScript Interview Questions & Answers on variable scopes & context

♥ Q1 – Q10 JavaScript Interview Questions & Answers on variable scopes & context

Posted on July 27, 2015 by Arulkumaran Kumaraswamipillai

You can try the code samples in this JavaScript interview questions and answers as follows:

1) In Goggle chrome or FireFox browser, you can bring up the developer tool with the "**F12**" on windows. You can also get to via the browser menu.

2) You can go to "**http://jsfiddle.net/**" on a browser. To get console.log(…) outputs, add "**https://getfirebug.com/firebug-lite-debug.js**" to the "External Resources" on the LHS menu if using FireFox.

Q1. Can you list which variables are accessible without any errors in locations marked **//1**, **//2**, **//3**, and **//4** in the code shown below?

```
var var1 = "I am var1"

function a() {
   var var2 = "I am var2"
   var3 = "I am var3"

   function b() {
       var var4 = "I am var4"
      // 2 - inside function b()
   }

   if(var2){
      var var5 = "I am var5"
      // 3 inside block
   }

   // 1 inside function a()

   b()//invoking function b()
}

a(); //invoking function a()

// 4 outside

```

A1. It is important to understand global variables and functional variables.

//1

1. var1 is accessible as it is defined in a global context outside the function. The "this" variable points to the window object of the browser window. Even if it is defined with "var" keyword.

2. var2 is locally defined within the function a(), and is accessible.

3. var3 is accessible as it is defined within function a(), but var3 is globally accessible outside as well as it is defined without the "**var**" keyword. var3 is a global variable. This is bad.

16 Technical Key Areas

80+ step by step Java Tutorials

4. The block variable "**var5**" is accessible as well. If it were defined with the newly introduced "set" keyword for the block only scope, it would not have been accessible.

//2

In this position the accessible variables are

1. global varibles var1 and var3.

2. functional variables defined within "function b()" var4

3. any outer functional variables encclosing "function b()", in this case "funaction a()" is enclosing "function b()", hence var2 is accessible as well. But function a() can't access any variables inside "function b()".

//3

is a block scope. This is a "if" block. In this position the accessible variables are

1. global varibles var1 and var3.

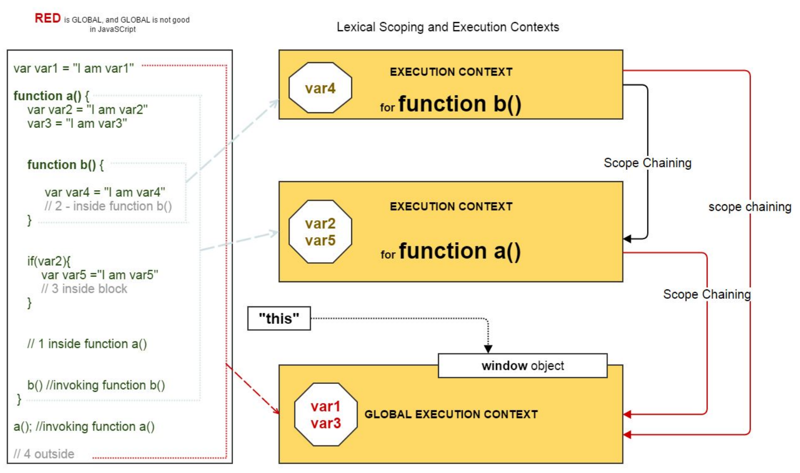
2. functional variables defined within "function a()", i.e. var2 and var5.

//4

Only global variables var1 and var3 are accessible.

100+ Java pre-interview coding tests

How good are your .....?


RED is GLOBAL, and GLOBAL is not good in JavaSCript
Lexical Scoping and Execution Contexts
EXECUTION CONTEXT for function b()
var4
Scope Chaining
EXECUTION CONTEXT for function a()
var2
var5
scope chaining
Scope Chaining
"this"
window object
var1
var3
GLOBAL EXECUTION CONTEXT

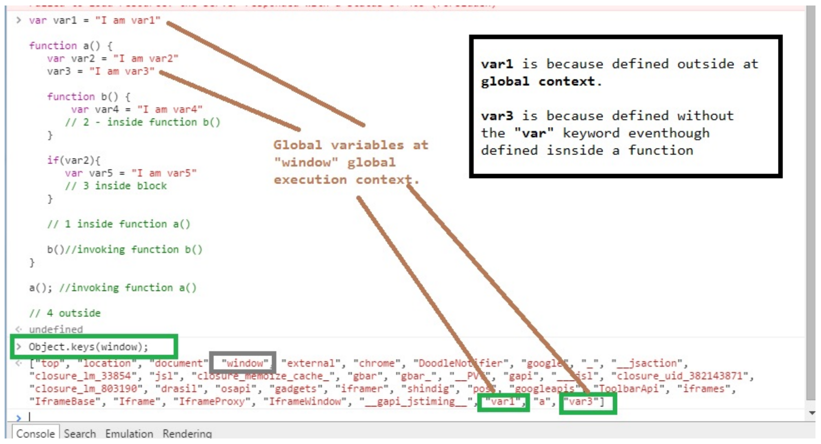
```
var var1 = "I am var1"

function a() {
   var var2 = "I am var2"
   var3 = "I am var3"

   function b() {

      var var4 = "I am var4"
      // 2 - inside function b()
   }

   if(var2){
      var var5 ="I am var5"
      // 3 inside block
   }

   // 1 inside function a()

   b() //invoking function b()
}
a(); //invoking function a()

// 4 outside
```

JavaScript variable scopes

Q2. Is there a way to list all the global variables defined in the code?
A2. Yes.

```
Object.keys(window);
```

Output when run on Google Chrome console with F12 or Ctrl+Shift+J or More Tools -> JavaScript console.



```
> var var1 = "I am var1"

  function a() {
     var var2 = "I am var2"
     var3 = "I am var3"

     function b() {
        var var4 = "I am var4"
        // 2 - inside function b()
     }

     if(var2){
        var var5 = "I am var5"
        // 3 inside block
     }

     // 1 inside function a()

     b()//invoking function b()
  }

  a(); //invoking function a()

  // 4 outside
< undefined
> Object.keys(window);
< ["top", "location", "document", "window", "external", "chrome", "DoodleNotifier", "google", "_", "__jsaction",
  "closure_lm_33854", "jsl", "closure_memoize_cache_", "gbar", "gbar_", "__PVT", "gapi", "___jsl", "closure_uid_382143871",
  "closure_lm_803190", "drasil", "osapi", "gadgets", "iframer", "shindig", "post", "googleapis", "ToolbarApi", "iframes",
  "IframeBase", "Iframe", "IframeProxy", "IframeWindow", "__gapi_jstiming__", "var1", "a", "var3"]
> |
```

Global variables at "window" global execution context.

var1 is because defined outside at **global context**.

var3 is because defined without the **"var"** keyword eventhough defined isnside a function

JavaScript Global varaibles

Q3. Is there a "block level" scope in JavaScript like in other languages like Java? For example, In Java, if you define a varaible within an "if" block, it is not accessible outside that block.

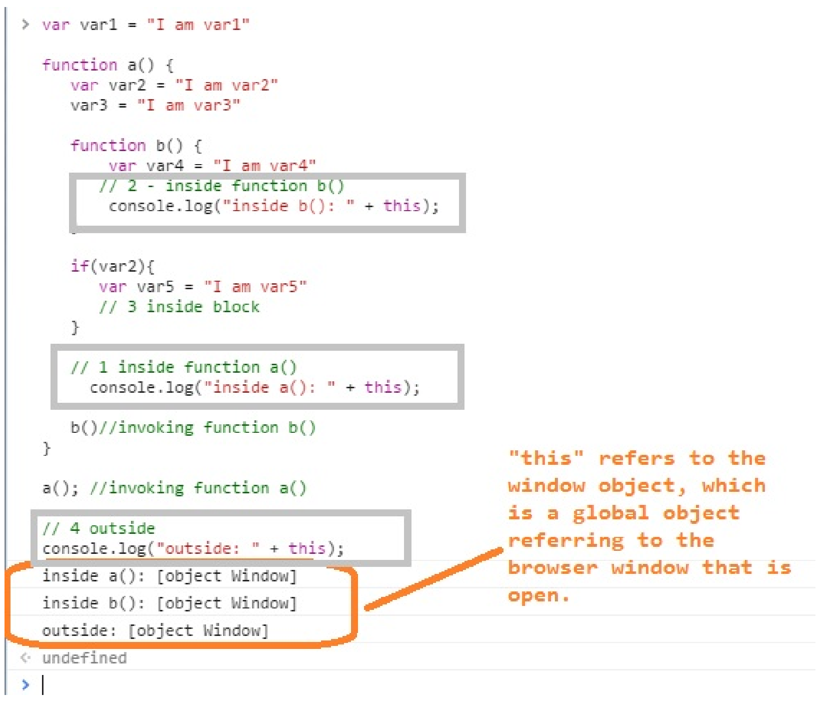
A3. The new “ECMAScript 2015 (ES6)” has introduced a “**let**” keyword. let allows you to declare variables that are limited in scope to the block, statement, or expression on which it is used. This is unlike the var keyword, which defines a variable globally, or locally to an entire function regardless of block scope. So, in the above code, if you had defined

```
if(var2){
       let var5 = "I am var5"
       // 3 inside block
    }

```

The var5 would have been accessible only in //3. The //1 can’t see it when defined with “let”. This keyword “let” is not still supported by the browsers at the time of writing. The support may be provided in the future.

Q4. Are JavaScript implicit variable “**this**” referring to the context and “scope” the same thing
A4. No. Scope is function-based, and context is object-based. In other words, scope pertains to the variable access of a function when it is invoked and is unique to each invocation. Context is always the value of the this keyword which is a reference to the object that “owns” the currently executing code.

If you take the above example, the code runs in the “window” object context.



```
> var var1 = "I am var1"

  function a() {
     var var2 = "I am var2"
     var3 = "I am var3"

     function b() {
        var var4 = "I am var4"
       // 2 - inside function b()
        console.log("inside b(): " + this);

     if(var2){
       var var5 = "I am var5"
       // 3 inside block
     }

     // 1 inside function a()
       console.log("inside a(): " + this);

     b()//invoking function b()
  }

  a(); //invoking function a()

  // 4 outside
  console.log("outside: " + this);
  inside a(): [object Window]
  inside b(): [object Window]
  outside: [object Window]
<· undefined
>
```

"this" refers to the window object, which is a global object referring to the browser window that is open.

JavaScript Context

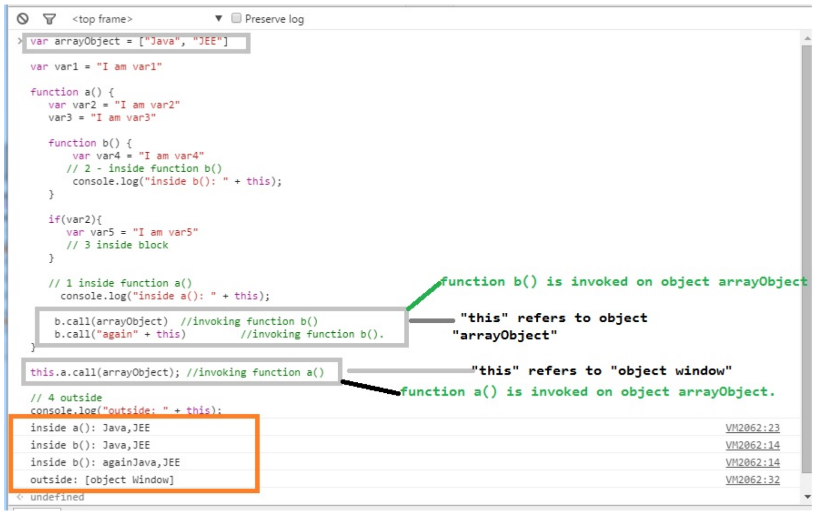
Q5. What do you think will be the output of the following code?

```
var arrayObject = ["Java", "JEE"]

var var1 = "I am var1"

function a() {
   var var2 = "I am var2"
   var3 = "I am var3"

   function b() {
      var var4 = "I am var4"
      //2 - inside function b()
      console.log("inside b(): " + this);
   }

   if(var2){
      var var5 = "I am var5"
      //3 inside block
   }

    //1 inside function a()
    console.log("inside a(): " + this);

    b.call(arrayObject)  //invoking function b()
    b.call(" again " + this)         //invoking
}

this.a.call(arrayObject); //invoking function a(

// 4 outside
console.log("outside: " + this);
```

A5. An alternative approach to calling a function is with the the “call(object)” method. It takes the object to call the function on as an argument.

```
this.a.call(arrayObject);
```

In the above snippet “this” refers to the object “window”. In other words, the context is “window”. The “function a()” is invoked on object “arrayObject” as the new context. Hence, within the “function a()” the implcit varible “this” points to the “arrayObject”. Even within “function b()”, the context is “arrayObject” as it was invoked with “b.call(arrayObject)”. It is the same result if you did call “b.call(” again ” + this) “. But if you had called with just “b()”, inside “function b()”, the context would be a “object window”.



<top frame>
Preserve log
function b() is invoked on object arrayObject
"this" refers to object "arrayObject"
"this" refers to "object window"
function a() is invoked on object arrayObject.

```
var arrayObject = ["Java", "JEE"]

var var1 = "I am var1"

function a() {
   var var2 = "I am var2"
   var3 = "I am var3"

   function b() {
      var var4 = "I am var4"
     // 2 - inside function b()
      console.log("inside b(): " + this);
   }

   if(var2){
     var var5 = "I am var5"
     // 3 inside block
   }

   // 1 inside function a()
    console.log("inside a(): " + this);

    b.call(arrayObject)  //invoking function b()
    b.call("again" + this)         //invoking function b().
}

this.a.call(arrayObject); //invoking function a()

// 4 outside
console.log("outside: " + this);
inside a(): Java,JEE                                              VM2062:23
inside b(): Java,JEE                                              VM2062:14
inside b(): againJava,JEE                                         VM2062:14
outside: [object Window]                                          VM2062:32
< undefined
```

JavaScript contexts

Q6. What is the difference between the .call(….) and .apply(…) methods?

A6. The difference is that **apply(argArray)** lets you invoke the function with arguments as an **array** whereas **call(arg1, arg2, arg3)** requires the parameters be listed explicitly.

call(…) and apply(…) methods of function object

```
var arrayObject = ["Java", "JEE"]

function a(arg1, arg2, arg3) {
     console.log(this);
     console.log(arg1  + " " + arg2 + " " + arg3)
}

a.call(arrayObject, "Hello", "Mr" , "John" );   /
a.apply(arrayObject, ["Hello", "Mr" , "John"]); /
```

Q7. What is the difference between undefined and null?
A7. The value of a variable with **no value** is undefined (i.e.it has been declared, but has not been initialized).

```
var a;
console.log(a) // undefined
```

Variables can be emptied by setting their value to **null**. For example, to get a variable or closure to be garbage collected to release the memory.

```
function doSomething(arg1) {
   var local = "Some Text"

   //a closure that has access to arg1 & local
   return function() {
        console.log(arg1 + " " +  local);
   }

}

var fn1 = doSomething("Printing...");    //closu
var fn2 = doSomething("Outputting...");  //closu

fn1(); //invoking - Printing... Some Text
fn2(); //invoking - Outputting... Some Text

//now clear the closure memory. Will be Garbage
fn1 = null;
fn2 = null;

fn1(); //error fn1 is not a function
```

Q8. What is the difference between "==" and "==" in JavaScript?
A8. The **==** operator will compare for equality after doing any necessary type conversions. The **===** operator will not do the conversion, so <u>if two values are not the same type</u> === will simply return false.

```
"abc" == new String("abc")    // true
"abc" === new String("abc")   // false
```

We are comparing a string literal with a string object. Hence "===" returns false.

Q9. How does "===" differ from "==" when checking for null or undefined?
A9.

===

```
if (nullRef === null) {
  // executes this block only if null
}

if (undefinedRef === undefined) {
    // executes this block only if undefined
}
```

==

Executes the block if the the "ref" is either null or undefined.

```
if (nullRef === null) {
    // executes this block
}

if (undefinedRef === undefined) {
    // executes this block
}
```

Q10. What is the purpose of "use strict" directive in JavaScript?
A10. Strict mode makes it easier to write "**secure**" JavaScript. Strict mode changes previously accepted "bad syntax" into real errors. For example, with strict mode you cannot use undeclared variables.

If "**use strict**" directive is declared at the top of the JavaScript file, it has a global scope. If declared within a function, it has a functional scope.

```
"use strict";
```

```
x = 3.14;       // ReferenceError. assignement to
var y = 3.14;   //good
```

Key Points on JavaScript Interview Questions and Answers

1. Global variables are bad as they can conflict with third party library variable names.
2. **Functions are objects** in JavaScript as they can be nested, passed to another function, and you can invoke methods like call(…) and apply(…) on a function.
3. JavaScript has 3 scopes: **global**, **functional**, and **block**. The block scope was recently introduced.
4. Scope and context are different concepts. A context refers to an object that invokes a function with the keyword "**this**". The "window" object is a global context. When you create an object in JavaScript, and invoke a function on that object, then "this" refers to the object that declares the function.
5. You can pass a different object to a function with the **call(…)** and **apply(…)** methods defined in the object function.

Popular Member Posts

01: ♦ 15+ Hibernate basics Q1 – Q7 interview questions & answers

1,819 views

♦ 11 Spring boot interview questions & answers

968 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

855 views

01: ♦ 15 Beginner level Java multi-threading interview Q&A

548 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

442 views

18 Java scenarios based interview Questions and Answers

418 views

♦ 7 Java debugging interview questions & answers

346 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

314 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

312 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

310 views

11 Like Share Tweet submit reddit 3 G+1 1 Share

Bio **Latest Posts**

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. Published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site. **945+** paid members. join my LinkedIn Group to get notified of special offers. User & expert **reviews**.

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job

interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. Published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site. **945+** paid members. join my LinkedIn Group to get notified of special offers. User & expert **reviews**.

‹ Groovy method chaining for DSL explained

♦ Q11 – Q20 JavaScript Interview Q&A on “this” variable & context ›

Posted in JavaScript Top Interview Q&A

Tags: JavaScript Interview Q&A

Select Category ▼

© Disclaimer