JavaScript in Brief

Of course, it is not possible to explain JavaScript in any reasonable amount of detail in just a few paragraphs, but we note a few facts about it:

- Although it is not the same as Java, it shares much common syntax:
  - statements end with a semicolon “;”
  - Most of the usual arithmetic, comparison, and boolean operators are used, as well as the boolean constants `true` and `false`:
    
    + - * / % && || ! == < <= > >= !=
    
  - However, the “/" operator will not do integer division, i.e., “25/10” is 2.5 in JavaScript (in Java it would be 2)
  - Strings can be enclosed either within single quotes or double quotes; the “+” operator is concatenation, just as in Java (so "hello "+ ‘world’+35 is the same as "hello world35"
  - The syntax of “if” “if...else”, “for”, and “while” loops is the same in Java and JavaScript

- However, there are many differences as well:
  - There are no type declarations; a variable’s type can change at run time (we say JavaScript is dynamically typed)
  - As a result, to declare a variable we use the word “var”, e.g., “var a = 20, b = true, c = "hello";
  - Furthermore, variables do not even have to be declared (except in “strict mode,” which we will consider later); if it doesn’t appear in a var statement, a variable is considered to be declared the first time it is used
  - Because JavaScript is used for scripting Web pages, most output consists of generated HTML; consequently, we will use a debugging environment called the JavaScript console to see program output

With these few simple facts in mind, let’s look at scope rules in Java.
An Online JavaScript Test Environment

Go to the site http://jsfiddle.net. Make sure that the menus on the left say “No-Library (pure JS)” and “onLoad”. In your Browser menus, open the “JavaScript console” (it may be under “Tools/Developer Tools” or “View/Developer” or some other menu/submenu heading; it may simply be named “Console”).

1. On the jsfiddle page, in the Window labeled “JavaScript,” type the following short program, then click the “Run” button at the top of the screen:

   ```javascript
   var glob = 'one'; // global variable
   function test() {
       glob = glob + "two";
       var loc = 'three'; // local variable
       unknown = 'four'; // which is this?
   }
   console.log("Initially, glob = "+glob);
   //console.log(" loc = "+loc);
   //console.log(" unknown = "+unknown);
   test();
   console.log("After calling test, glob = "+glob);
   //console.log(" loc = "+loc);
   console.log(" unknown = "+unknown);
   ```

   Now try uncommenting each of the commented-out calls to `console.log` to see what happens. You should see errors.

   (a) Why is “unknown” defined after `test` is called but not before?
   (b) Why is “loc” undefined both before and after `test` is called?

2. There is a concept in programming languages called “hole in scope.” Try this code:

   ```javascript
   var glob = 'one';
   
   function test1() {
       glob = glob + 'two';
   }
   
   function test2() {
       var glob = 'three';
   }
   
   test1();
   test2();
   console.log(glob);
   ```
Is function test2 within the scope of the first variable glob?

3. Java doesn’t allow variables to be re-declared; JavaScript does. Therefore, things like the following are legal in JavaScript; the second declaration overwrites the first:

```javascript
function test3() {
    var x = 10;
    var x = 'hello';
    console.log("x = "+x);
}
test3();
```

4. The `var` declaration in JavaScript does not obey block-scope (unlike declarations within block in Java and C). Opening a new block with “{...}” does not create a new scope for `var` variables. In the following, the declaration of `x` inside the for-loop overwrites the first declaration of `x`:

```javascript
function test4() {
    var x = 10;
    for (i = 0; i < 10; i++) {
        var x = 'hello';
    }
    console.log("x = " + x);
}
test4();
```

We say that JavaScript has *function scope* rather than block scope.

5. Before running this, can you guess the output?

```javascript
function test5() {
    var x = 10;
    for (var i = 0; i < 10; i++) {
        var x = 'hello';
    }
    console.log("x = " + x);
    console.log("i="+i);
}
test5();
```

6. A few years ago, a new type of declaration was added to JavaScript: the “let” declaration. This behaves like `var` except that, within a new block, it respects block scope. In order to use this (at least in jsfiddle), we must add a line at the top to enter *strict mode*:
"use strict";
function test6() {
    var x = 10;
    for (let i = 0; i < 10; i++) {
        let x = 'hello';
    }
    console.log("x = " + x);
    //console.log("i"+i);
}
test6();
//console.log("i"+i);