Core Concepts: Scope

Mark Bools

June 1, 2021

Last Modified: 2022-02-01

Abstract

Examining the core concept of scope

You will come across scope in several contexts all off which generally mean "area of effect".

Examples

Documents often contain a 'Scope' section that sets expectations about the application of the document's content; to which systems the document applies, which stakeholders have an interest in the document, etc.

Projects will often have a 'scope' document that specifies which stakeholders, processes, systems, etc. are affected by the project.

In software development scope is a widely used concept, perhaps the most common being variable scope.

Variable Scope

Variable scope can be a complex topic. To keep things simple we will consider simple scoping in a fictitious computer language.

pseudocode

```
1  var A = 10
2
3  function X (var B) {
4    var C = 30
5
6    print "First A in X = ", A
7
8    A = 99
9
10   print "Second A in X = ", A
11   print "B in X = ", B
12   print "C in X = ", C
13 }
14
15 print "A before calling X = ", A
16 print "B before calling X = ", B
17 print "A before calling X = ", C
18
19 call X(20)
20
21 print "A after calling X = ", A
22 print "B after calling X = ", B
23 print "A after calling X = ", C
```

If we run this program we get the following output.

output

```
A before calling X = 10
B before calling X = undefined
C before calling X = undefined
First A in X = 10
Second A in X = 99
B in X = 20
C in X = 30
A after calling X = 99
B after calling X = undefined
C after calling X = undefined
```

This program (*pseudocode*) runs from top to bottom. Line 1 defines a variable `A` assigning it the value `10`. As this definition is textually outside any other construct it is in the 'global' scope; visible throughout the rest of the program.

Next, lines 3–13, defines a function. The code inside this function is not executed at this point so we will pass over it for now.

Line 15 prints out a message showing the current value of variable `A`. Recall, we defined this on line 1 so the value is `10` as we see in *output* line 1.

Lines 16 and 17 print the values of variables `B` and `C`. These are defined in function `X`, but this has not been executed yet so they are `undefined`.

Line 19 calls function `X` passing the value `20`. *Now* the code of function `X` is executed.

Line 3 starts our `X` function and 'binds' the value `20` supplied in the `call` (line 19) to the variable `B`. Variable `B` is 'locally' scoped and only visible inside the `X` function.

Next we define `C`, assigning it the value `30`. Variable `C` is only visible within function `X`, like `B` it is locally scoped.

Line 6 prints variable `A`. Since variable `A` is globally scoped it is available even inside function `X`, so we see on *output* line 4 that `A` still has the value `10`.

Line 8 assigns a new value to `A`. This is the global `A` which now has `99` assigned to it.

Line 10 repeats the printing of `A` and *output* line 5 confirms that `A` now holds the value `99`.

Lines 11 and 12 print the values associated with `B` and `C`. *output* lines 6 and 7 confirm that they hold `20` and `30` respectively.

We now exit function `X`.

Line 21 again prints variable `A`. Line 8 of the *output* shows `A` still holds `99` as we assigned inside `X`. Remember `A` is global so each reference to `A` in the program are referring to the same `A`.

Lines 22 and 23 print `B` and `C`, *output* lines 9 and 10 show that they are once again `undefined` because `B` and `C` were local to function `X` but we are no longer in function `X` so they are not visible.