Win-Vector Blog

The Win-Vector LLC data science blog

An R function return and assignment puzzle

December 29, 2015 John Mount Programming, Tutorials assignment, R, R as it is

Here is an R programming puzzle. What does the following code snippet actually do? And ever harder: what does it mean? (See here for some material on the difference between what code does and what code means.)

```
f <- function() { x <- 5 }
f()
```

In `R version 3.2.3 (2015-12-10) -- "Wooden Christmas-Tree"` the code *appears* to call the function `f()` and return nothing (nothing is printed). When teaching I often state that you should explicitly use a non-assignment expression as your return value. You should write code such as the following:

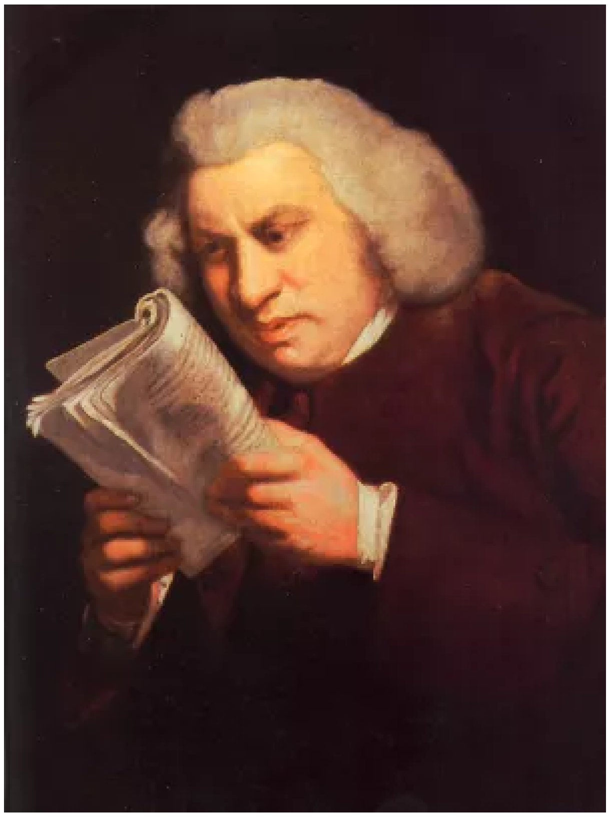
```
f <- function() { x <- 5; x }
f()
## [1] 5
```

(We are showing an R output as being prefixed with `##`.)

But take a look at the this:

```
f <- function() { x <- 5 }
print(f())
## [1] 5
```

It prints! Read further for what is really going on.



What is going on is: in R in the absence of an explicit `return()` statement functions always return the value of the last statement executed. Also in R assignment is itself a value returning expression (returning the value assigned). So the original function `f <- function() { x <- 5 }` is in fact returning a 5. We just don't see it. The 5 returned is "invisible" (see the "return values" section of *Advanced R*, Hadley Wickham, CRC 2015 for details).

As we said: R assignments return values. So you can return them and you can chain them like so:

```
a <- b <- c <- 5
print(a)
## [1] 5
```

What happens is the assignment `x <- 5` returns a value (in this case `5`), but that value has an attribute marking it invisible. This is why when you assign a value to a variable in R you don't see printing as a side effect. For example we don't see anything printed when we type the following:

```
x <- 5
```

We can remove the invisible attribute by adding parenthesis as follows:

```
( x <- 5 )
## [1] 5
```

Assignment also strips the invisible attribute, so we can write code like the following:

```
f <- function() { x <- 5 }
z <- f()
z
## [1] 5
```

(We can think of the expression `z <- f()` as removing the invisible attribute from the `5` stored in the variable `z` and then returning a new value `5` that is again invisible. So we don't see any printing during the assignment, but the value stored in z is now visible. Likely all of the visibility notes are stored in a reference handle and not actually in the values to allow efficient re-use of values.)

This is subtle and strange, and one of the reasons it can be hard to first approach R. R has fairly subtle semantics, but that is part of why it is so safe to program in and so powerful to use.

In language design I tend to prefer more transparency (the user reliably seeing something more directly related to what is going on- something vitally important for learning and debugging) and would have opted for assignment not returning a value (another way to suppress needless printing).

RELATED

You don't need to understand pointers to program using R
R is a statistical analysis package based on writing short scripts or programs (versus being based on GUIs like spreadsheets or directed
In "Coding"

R style tip: prefer functions that return data frames
While following up on Nina Zumel's excellent Trimming the Fat from glm() Models in R I got to thinking about code style in R. And I realized: you can make
In "Practical Data Science"

Prefer = for assignment in R
We share our opinion that = should be preferred to the more standard <- for assignment in R. This is from a draft of the appendix of our upcoming book.
In "Mathematics"

December 29, 2015 John Mount Programming, Tutorials assignment, R, R as it is

One thought on “An R function return and assignment puzzle”

ksankar

December 29, 2015 at 5:33 pm

Good one. Thanks.
P.S: On a slightly different topic f <- function() {x <<- 5} is also interesting

Comments are closed.