StackExchange sign up log in tour help stack overflow careers

Function returning more than one value

still trying to get into the R logic... what is the "best" way to unpack the results from a function returning multiple values?

I can't do this apparently:

```
R> functionReturningTwoValues <- function() { return(c(1, 2)) }
R> functionReturningTwoValues()
[1] 1 2
R> a, b <- functionReturningTwoValues()
Error: unexpected ',' in "a,"
R> c(a, b) <- functionReturningTwoValues()
Error in c(a, b) <- functionReturningTwoValues() : object 'a' not found
```

must I really do the following?

```
R> r <- functionReturningTwoValues()
R> a <- r[1]; b <- r[2]
```

or would the R programmer write something more like this:

```
R> functionReturningTwoValues <- function() {return(list(first=1, second=2))}
R> r <- functionReturningTwoValues()
R> r$first
[1] 1
R> r$second
[1] 2
```

--- edited to answer Shane's questions ---

I don't really need giving names to the result value parts. I am applying one aggregate function to the first component and an other to the second component (`min` and `max` . if it was the same function for both components I would not need splitting them).

r return-value multiple-results

edited Jul 10 at 12:58

user227710
2,331 ● 6 ● 22

asked Dec 1 '09 at 14:27

mariotomo
2,484 ● 1 ● 22 ● 48

5 FYI, another way to return multiple values is to set an `attr` on your return value. – Jonathan Chang Dec 1 '09 at 15:49

This is the equivalent of Python's tuple-unpacking. – smci Mar 20 at 11:58

7 Answers

(1) list[...]<- I had posted this nearly 10 years ago on r-help. It does not require a special operator but does require that the left hand side be written using `list[...]` like this:

```
# run the source command below first
list[a, b] <- functionReturningTwoValues()
```

Note: Recently `list` has been added to the development version of the gsubfn package and can be sourced via:

```
library(devtools)
source_url("https://raw.githubusercontent.com/ggrothendieck/gsubfn/master/R/list.R")
```

If you only need the first or second component these all work too:

```
list[a] <- functionReturningTwoValues()
list[a, ] <- functionReturningTwoValues()
list[, b] <- functionReturningTwoValues()
```

See the cited r-help thread for more examples.

(2) with If the intent is merely to combine the multiple values subsequently and the return values are named then a simple alternative is to use `with` :

```
myfun <- function() list(a = 1, b = 2)

list[a, b] <- myfun()
a + b

# same
with(myfun(), a + b)
```

(3) attach Another alternative is attach:

```
attach(myfun())
a + b
```

ADDED: with and attach

edited Aug 12 at 20:53

answered Feb 28 '13 at 16:23

G. Grothendieck

61.4k • 3 • 52 • 112

10 I accepted your answer because of the "with", but I can't reproduce what you describe for the left hand side usage of "list", all I get is "object 'a' not found" – mariotomo Mar 23 '13 at 15:42

2 It works for me. What did you try? Did you read the linked post and follow it? Did you define `list` and `[<-.result` as shown there? – G. Grothendieck Mar 23 '13 at 16:00

6 @G.Grothendieck, Would you mind if I put the content of your link into your answer? I think it would make it easier for for people to use it. – merlin2011 Apr 14 '14 at 7:29

6 I agree with @merlin2011; as written it seems like this syntax is embedded into R base. – knowah Jun 3 '14 at 22:52

3 @G.Grothendieck I agree with merlin2011 and knowah - it would be best if the actual code that is important here (the code referenced in the link) is in the answer. It might not be a bad idea to mention that the result object doesn't need to be named list. That confused me for a little while before reading your actual code. As mentioned the answer says that you need to run the code in the link but most people aren't going to read that code right away unless it's in the answer directly - this gives the impression that this syntax is in base R. – Dason Jul 22 '14 at 17:08

I somehow stumbled on this clever hack on the internet somehow ... I'm not sure if it's nasty or beautiful, but it lets you create a "magical" operator that almost allows you to unpack multiple return values into their own variable. The `:=` function is defined here, and included below for posterity:

```
':=' <- function(lhs, rhs) {
  frame <- parent.frame()
  lhs <- as.list(substitute(lhs))
  if (length(lhs) > 1)
    lhs <- lhs[-1]
  if (length(lhs) == 1) {
    do.call(`=`, list(lhs[[1]], rhs), envir=frame)
    return(invisible(NULL))
  }
  if (is.function(rhs) || is(rhs, 'formula'))
    rhs <- list(rhs)
  if (length(lhs) > length(rhs))
    rhs <- c(rhs, rep(list(NULL), length(lhs) - length(rhs)))
  for (i in 1:length(lhs))
    do.call(`=`, list(lhs[[i]], rhs[[i]]), envir=frame)
  return(invisible(NULL))
}
```

With that in hand, you can do what you're after:

```
functionReturningTwoValues <- function() { return(list(1, matrix(0, 2, 2))) }
c(a, b) := functionReturningTwoValues()
a
#[1] 1
b
#     [,1] [,2]
# [1,]    0    0
# [2,]    0    0
```

I don't know how I fell about that. Perhaps you might find it helpful in your interactive workspace. Using it to build (re-)usable libraries (for mass consumption) might not be the best idea, but I guess that's up to you.

... you know what they say about responsibility and power ...

edited Mar 11 '14 at 18:33

answered Dec 1 '09 at 23:21

Steve Lianoglou
4,634 ● 11 ● 19

2 Wow, what a hack! Thanks for sharing! – Amyunimus Jun 3 '12 at 4:42

That is scary and beautiful at the same time. – Mark Mar 11 '13 at 15:55

1 Also I'd discourage it a lot more now than when I originally posted this answer since the data.table package uses the `:=` operator mucho in a much handier way :-) – Steve Lianoglou Mar 12 '13 at 0:58

Usually I wrap the output into a list, which is very flexible (you can have any combination of numbers, strings, vectors, matrices, arrays, lists, objects int he output)

so like:

```
func2<-function(input) {
   a<-input+1
   b<-input+2
   output<-list(a,b)
   return(output)
}

output<-func2(5)

for (i in output) {
   print(i)
}

[1] 6
[1] 7
```

answered Dec 1 '09 at 15:01

Federico Giorgi
3,112 ● 3 ● 22 ● 41

There's no right answer to this question. I really depends on what you're doing with the data. In the simple example above, I would strongly suggest:

1. Keep things as simple as possible.
2. Wherever possible, it's a best practice to keep your functions vectorized. That provides the greatest amount of flexibility and speed in the long run.

Is it important that the values 1 and 2 above have names? In other words, why is it important in this example that 1 and 2 be named a and b, rather than just r[1] and r[2]? One important thing to understand in this context is that a and b are *also* both vectors of length 1. So you're not really changing anything in the process of making that assignment, other than having 2 new vectors that don't need subscripts to be referenced:

```
> r <- c(1,2)
> a <- r[1]
> b <- r[2]
> class(r)
[1] "numeric"
> class(a)
[1] "numeric"
> a
[1] 1
> a[1]
[1] 1
```

You can also assign the names to the original vector if you would rather reference the letter than the index:

```
> names(r) <- c("a","b")
> names(r)
[1] "a" "b"
> r["a"]
a
```

```
1
```

[Edit] Given that you will be applying min and max to each vector separately, I would suggest either using a matrix (if a and b will be the same length and the same data type) or data frame (if a and b will be the same length but can be different data types) or else use a list like in your last example (if they can be of differing lengths and data types).

```
> r <- data.frame(a=1:4, b=5:8)
> r
  a b
1 1 5
2 2 6
3 3 7
4 4 8
> min(r$a)
[1] 1
> max(r$b)
[1] 8
```

edited Dec 1 '09 at 15:03

answered Dec 1 '09 at 14:42

Shane

50.8k ● 16 ● 146 ● 185

edited the question in order to include your remarks. thanks. giving names to things like `r[1]` can help to make things more clear (all right, not if names like `a` come in their place). – mariotomo Dec 1 '09 at 15:02

Lists seem perfect for this purpose. For example within the function you would have

```
x = desired_return_value_1 # (vector, matrix, etc)

y = desired_return_value_2 # (vector, matrix, etc)

returnlist = list(x,y...)

}  # end of function
```

main program

```
x = returnlist[[1]]

y = returnlist[[2]]
```

edited Feb 28 '13 at 16:05

Jos Vinke

1,741 ● 1 ● 14 ● 34

answered Feb 28 '13 at 15:41

Arnold

21 ● 1

Yes to your second and third questions -- that's what you need to do as you cannot have multiple 'lvalues' on the left of an assignment.

answered Dec 1 '09 at 14:41

Dirk Eddelbuettel

171k ● 17 ● 298 ● 402

How about using assign?

```
functionReturningTwoValues <- function(a, b) {
  assign(a, 1, pos=1)
  assign(b, 2, pos=1)
}
```

You can pass the names of the variable you want to be passed by reference.

```
> functionReturningTwoValues('a', 'b')
> a
[1] 1
> b
[1] 2
```

If you need to access the existing values, the converse of `assign` is `get`.

answered Jul 4 '14 at 18:17

Steve Pitchers
1,795 ● 1 ● 15 ● 17

... but this requires you to know the names of the receiving variables in that environment – smci Mar 20 at 11:55

@smci Yes. That is why the "named list" method in the question is generally better: `r <- function() { return(list(first=1, second=2)) }` and reference the results using `r$first` and `r$second`. – Steve Pitchers Mar 23 at 10:09