Data Science for Economists

Lecture 4: Functions

Drew Van Kuiken
University of North Carolina | ECON 370

Table of contents

Introduction

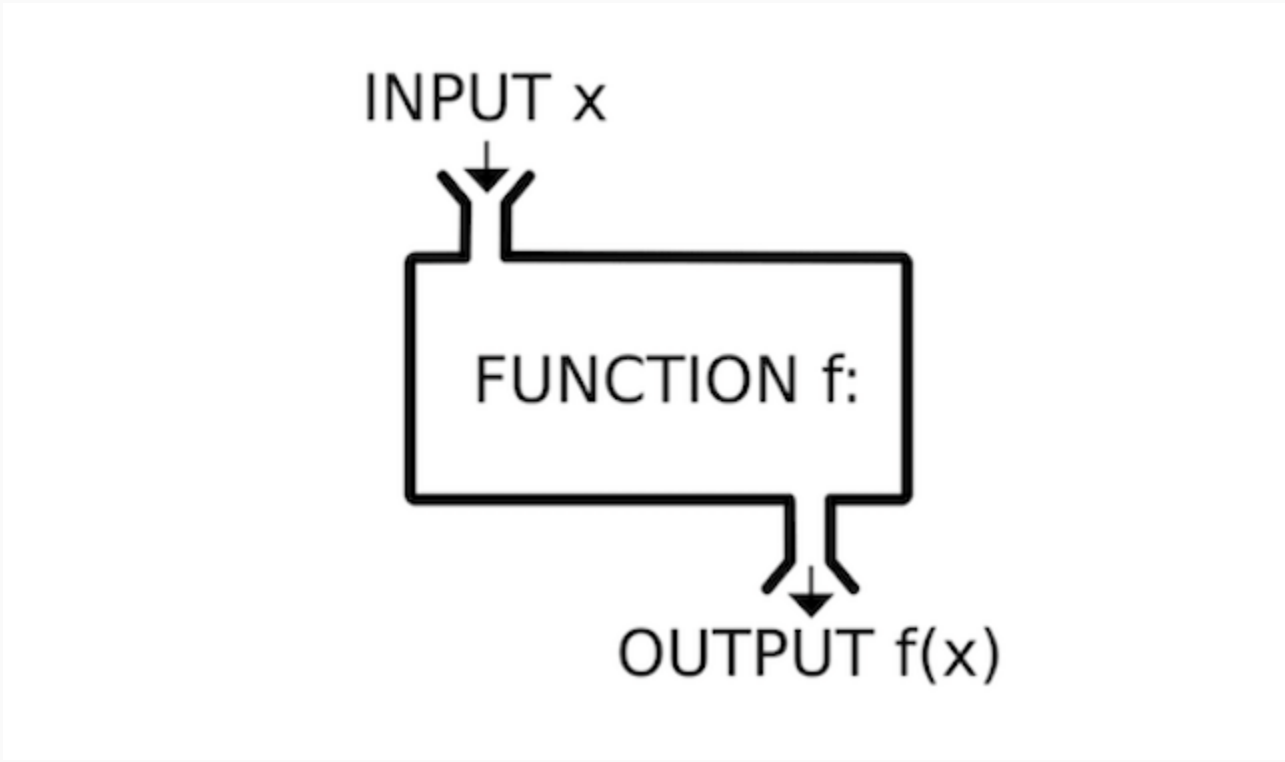
Agenda

Today we will finally officially cover functions.

While we have already used and talked about them quite a lot, there are
we should go over along with learning how to write our own.

Functions

What is a function? Just like in m



Terminology

- Function: f
- Function input/arguments: x
- Function output: $f(x)$

Motivation

Let's generate a simple data.frame:

```
d = data.frame(x=runif(6),y=rnorm(6),z=rchisq(6,1))
```

Imagine we want to rescale each of these vectors so the minimum value
and the maximum in the column is 1. Here is one way we could do this:

```
d$x = (d$x - min(d$x,na.rm=TRUE))/(max(d$x, na.rm=TRUE) - min(d$x,
d$y = (d$y - min(d$x,na.rm=TRUE))/(max(d$y, na.rm=TRUE) - min(d$y,
d$z = (d$z - min(d$z,na.rm=TRUE))/(max(d$z, na.rm=TRUE) - min(d$z,

head(d)
```

```
##           x           y          z
## 1 0.2704415  0.02365953 0.14054315
## 2 0.8299695  0.04338331 0.54221604
## 3 0.4060968  0.57550079 0.00000000
## 4 0.9358038  0.15466332 1.00000000
## 5 1.0000000 -0.42449921 0.33814069
## 6 0.0000000 -0.23047777 0.04121895
```

Something's wrong here.

Motivation

Look more closely:

```
d$y = (d$y - min(d$x,na.rm=TRUE))/(max(d$y, na.rm=TRUE) - min(d$y,
```

I accidentally included the minimum from column x as opposed to colur
what we want is this:

```
d$Var = (d$Var - min(d$Var,na.rm=TRUE))/(max(d$Var, na.rm=TRUE) -
```

where we can give R a list of columns and it performs the same process
This is what a function does. Our initial ones will be a little bit simpler t

What is a function

Functions in programming are just like functions in math: they take in in
unique output.

Functions allow you to put code that you use frequently into a single lin

> You should consider writing a function whenever you’ve copied a
> block of code more than twice (i.e. you now have three copies of

R for Data Science

Using functions appropriately makes for much cleaner code and code w

Functions are verbs; arguments are nouns.

A Trivial Function

```r
return_input = function(x){
  x #return the input as output
}

return_input(1)
```

```
## [1] 1
```

```r
return_input(letters)
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o"
## [20] "t" "u" "v" "w" "x" "y" "z"
```

```r
return_input = function(x){
  return(x) #this is equivalent, I prefer this
}

return_input(1)
```

```
## [1] 1
```

Pythagorean Theorem

```r
hypotenuse = function(a,b){
  sqrt(a^2+b^2)
}
hypotenuse(3,4)
```

```
## [1] 5
```

```r
hypotenuse(1:5,2:6)
```

```
## [1] 2.236068 3.605551 5.000000 6.403124 7.810250
```

```r
hypotenuse(3,1:5)
```

```
## [1] 3.162278 3.605551 4.242641 5.000000 5.830952
```

```r
hypotenuse(3:5,1:5) #don't do this
```

```
## Warning in a^2 + b^2: longer object length is not a multiple of
## length
```

```
## [1] 3.162278 4.472136 5.830952 5.000000 6.403124
```

A Weighted Mean

```r
wt_mean = function(x,w){
  sum(x*w)/sum(w)
}
wts = runif(20)
wt_mean(1:20,wts)
```

```
## [1] 11.99396
```

```r
wt_mean = function(x,w){
  if(length(x)≠length(w)){
    stop("x and w must be the same length")
  }
  sum(x*w)/sum(w)
}

wt_mean(1:20,wts[-1])
```

```
## Error in wt_mean(1:20, wts[-1]): x and w must be the same lengt
```

```r
wt_mean(w = wts, x=1:20)
```

```
## [1] 11.99396
```

Default Arguments

In `R` you can define default arguments for functions. Typically you do th
that is used often and you don't want to always pass it to the function.

We've already seen one example of default arguments:

```
rnorm(1)
```

```
## [1] -0.3200564
```

```
rnorm(1,mean=0,sd=1)
```

```
## [1] -1.311522
```

To define a default argument, simply add it to the list of arguments with
the default value.

```
test_fun = function(x, y=2){
  x+y
}
test_fun(3)
```

```
## [1] 5
```

Default Arguments: Weighted M

```r
wt_mean = function(x,w=rep(1,length(x))){

  # Description: Takes the weighted average of x using weights w
  # Default w is a vector of 1s the same length as x.

  if(length(x)≠length(w)){
    stop("x and w must be the same length")
  }

  sum(x*w)/sum(w)
}

wt_mean(1:20)     # my weighted mean fun with equal weights
```

```
## [1] 10.5
```

```r
mean(1:20)         # same as my function
```

```
## [1] 10.5
```

```r
wt_mean(1:20,wts) # with the random weights
```

```
## [1] 11.99396
```

Default Arguments

```
normalize = function(x, m = mean(x,na.rm=na.rm),s = sd(x,na.rm=na.
  (x - m)/s
}
normalize(1:10)
```

```
##  [1] -1.4863011 -1.1560120 -0.8257228 -0.4954337 -0.1651446  0.1
##  [7]  0.4954337  0.8257228  1.1560120  1.4863011
```

```
normalize(c(1:10,NA))
```

```
##  [1] NA NA NA NA NA NA NA NA NA NA NA
```

```
normalize(c(1:10,NA),na.rm=TRUE)
```

```
##  [1] -1.4863011 -1.1560120 -0.8257228 -0.4954337 -0.1651446  0.1
##  [7]  0.4954337  0.8257228  1.1560120  1.4863011         NA
```

Writing Functions: Good Style

The following are some recommendations for good programming style w

1. Name your functions something descriptive.
    - Remember, they are verbs!
2. Try to foresee errors and incorrect inputs to your functions and prog
   warnings.
    - This is less important if your functions are only for you.
3. Comment, comment comment!
4. If you write a "family" of functions, try to use similar naming schem
5. Scope....

Scope

I've referred to the global environment a lot throughout lectures. In term
most general.

However, the environment within a function is a separate, more specific
Understanding this difference is important.

Variables in the global environment can be referred to in `R` but variable
environment that are not returned *will not* be saved in the global enviro

It is generally frowned upon to refer to too many global variables within

- It also depends on how lazy you're being

Let's see some examples.

Scope Examples

```r
y = 2
add_xy = function(x){
  x + y
}
add_xy(3)
```

```
## [1] 5
```

```r
my_mean = function(x){
  x_sum = sum(x)
  x_sum/length(x)
}
my_mean(1:10)
```

```
## [1] 5.5
```

```r
x_sum
```

```
## Error in eval(expr, envir, enclos): object 'x_sum' not found
```

Advice Regarding Scope

- Variables that are unlikely to change throughout a script are safe to referred to as "global variables."
    - e.g. N_sim in a simulation exercise.
- When writing functions, only refer to variables in the global environ requirements described above. Relying on globals too much is slop
- However, writing functions with too many arguments is also bad pro to find a balance.
- There are ways to save variables created in a function to the global up <<-). I would generally avoid these. They can get you into trouble
    - If you want to return multiple objects, make a list!!

Returning vs Printing

I have hinted at the difference between returning an object and printing

This distinction matters the most for functions.

When you return an object from a function, that is the only thing that ca

When you print an object, it shows output but does not return the objec
unless you also specify it to print.

The best thing I can say to understand the difference is that printing is f
is for the computer!

Let's look at some examples.

Returning vs Printing

```r
plus_delta = function(x,delta=1){
  print(paste0("We are adding ", delta, " to ", x, "!"))
  x + delta
}

plus_delta(5)
```

```
## [1] "We are adding 1 to 5!"

## [1] 6
```

```r
plus_delta(4.5,0.75)
```

```
## [1] "We are adding 0.75 to 4.5!"

## [1] 5.25
```

Returning vs Printing

```r
mult_plus1 = function(x,y){
  xy = x*y
  print(xy)
  xy+1
}

mult_plus1(2,3)
```

```
## [1] 6

## [1] 7
```

```r
xy
```

```
## Error in eval(expr, envir, enclos): object 'xy' not found
```

```r
out1 = mult_plus1(2,3)
```

```
## [1] 6
```

```r
out1
```

```
## [1] 7
```

#

Returning vs Printing

If the last line of a function is a print statement, will also return the prin
printed characters)

```
mult_plus1 = function(x,y){
  xy = x*y
  print(xy+1)
}

out2 = mult_plus1(2,3)
```

```
## [1] 7
```

```
out2
```

```
## [1] 7
```

```
class(out2)
```

```
## [1] "numeric"
```

Misc Aspects of Functions

Functions don't have to have arguments.

Functions don't have to return an object.

Functions can only return one object; however, if you're using if stateme
multiple returns specified. It's just ultimately only one will be used.

You can write functions to take an arbitrary number of inputs using `...`

No arguments or Returns

```r
say_hello = function(){
  print("Hello! :)")
} #notice, nothing is being returned either!!

say_hello()
```

```
## [1] "Hello! :)"
```

```r
say_my_name = function(name){
  print(name)
}

say_my_name("Alex")
```

```
## [1] "Alex"
```

Conditional Returns

```r
is_prime = function(x){
  if ( x %% 1 != 0 ) stop("x must be an integer!")
  if ( length(x)!=1 ) stop("x can only be length 1!")
  if ( x %in% 1:2 ){ # if x is 1 or 2, return FALSE or TRUE
    return(x == 2)
  } else { # otherwise, loop through numbers 3 to x
    num_vec    = 3:x
    prime_list = 2
    i          = 1
    for(n in num_vec){
      if(sum((n %% prime_list) == 0) == 0){
        i             = i + 1
        prime_list[i] = n
      }
    }
    if(x %in% prime_list){ # if x is in list of primes, return TRU
      return(TRUE)
    }else{ # otherwise, return FALSE
      return(FALSE)
    }
  }
}
```

Conditional Returns

```r
primes1to100 = sapply(1:100,is_prime)
names(primes1to100) = 1:100
primes1to100
```

```
##     1     2     3     4     5     6     7     8     9    10    1
## FALSE  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRU
##    14    15    16    17    18    19    20    21    22    23    2
## FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE FALS
##    27    28    29    30    31    32    33    34    35    36    3
## FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE  TRU
##    40    41    42    43    44    45    46    47    48    49    5
## FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALS
##    53    54    55    56    57    58    59    60    61    62    6
##  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALS
##    66    67    68    69    70    71    72    73    74    75    7
## FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALS
##    79    80    81    82    83    84    85    86    87    88    8
##  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE  TRU
##    92    93    94    95    96    97    98    99   100
## FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
```

Arbitrary Inputs

```r
commas = function( ... ){
  out = paste( ... ,sep = ", ")
  out
}

commas("red","blue", "yellow","green")
```

```
## [1] "red, blue, yellow, green"
```

Any arguments that come after `...` *must have default arguments!*

Next lecture(s): Misc.