Data Science for Economists

Lecture 6: Functions

Alex Marsh
University of North Carolina | ECON 390

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 a few quirks that we should go over along with learning how to write our own.

Functions

What is a function

Functions in programming are just like functions in math: they take in inputs and return a unique output.

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

> You should consider writing a function whenever you've copied and pasted a block of code more than twice (i.e. you now have three copies of the same code).

R for Data Science

Using functions appropriately makes for much cleaner code and code with fewer errors.

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" "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
```

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

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 shorter object
## 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] 10.07217
```

```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 length
```

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

```
## [1] 10.07217
```

Default Arguments

In `R` you can define default arguments for functions. Typically you do this if there's a value 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] 1.224082
```

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

```
## [1] 0.3598138
```

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

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

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

Defaul Arguments

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

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

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

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

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

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

Writing Functions: Good Style

The following are some recommendations for good programming style with functions:

1. Name your functions something descriptive.
    - Remember, they are verbs!
2. Try to foresee errors and incorrect inputs to your functions and program in errors and 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 schemes.
5. Scope....

Scope

I've referred to the global environment a lot throughout lectures. In terms of scope, it is the most general.

Scope

I've referred to the global environment a lot throughout lectures. In terms of scope, it is the most general.

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

Scope

I've referred to the global environment a lot throughout lectures. In terms of scope, it is the most general.

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

Variables in the global environment can be referred to in `R` but variables in a function environment that are not returned *will not* be saved in the global environment.

Scope

I've referred to the global environment a lot throughout lectures. In terms of scope, it is the most general.

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

Variables in the global environment can be referred to in `R` but variables in a function environment that are not returned *will not* be saved in the global environment.

It is generally frowned upon to refer to too many global variables within functions (not by me, though 😄)

- There are legitimate reasons, I just have a bad habit.

Scope

I've referred to the global environment a lot throughout lectures. In terms of scope, it is the most general.

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

Variables in the global environment can be referred to in `R` but variables in a function environment that are not returned *will not* be saved in the global environment.

It is generally frowned upon to refer to too many global variables within functions (not by me, though 😁)

- There are legitimate reasons, I just have a bad habit.

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 be created and referred to as "global variables."
  - e.g. N_sim in a simulation exercise.
- When writing functions, only refer to variables in the global environment that meet the requirements described above. Relying on globals too much is sloppy programming.
- However, writing functions with too many arguments is also bad programming. You have to find a balance.
- There are ways to save variables created in a function to the global environment (look 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 an object before.

This distinction matters the most for functions.

When you return an object from a function, that is the only thing that can be returned.

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

The best thing I can say to understand the difference is that printing is for you and returning 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
```

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 statements, there might be multiple returns specified. It's just ultimately only one will be used.

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

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){
  num_vec    = 3:x
  prime_list = c(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){
    return(TRUE)
  }else{
    return(FALSE)
  }}

primes2to20 = sapply(2:20,is_prime)
names(primes2to20) = 2:20
primes2to20
```

```
##     2     3     4     5     6     7     8     9    10    11    12    13    14
##  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE
##    15    16    17    18    19    20
```

Arbitrary Inputs

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

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

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

Next lecture(s): Misc.