Untitled

Leah Brooks

February 24, 2021

Today in R

Why Functions?

- Many times, you need to repeat very similar code
- You can copy and paste, but ..
    - Subject to error when you make your small changes
    - A real bother when you need to change things
- For example

Why Functions?

- ▶ Many times, you need to repeat very similar code

- ▶ You can copy and paste, but ..
  - ▶ Subject to error when you make your small changes
  - ▶ A real bother when you need to change things

- ▶ For example
  - ▶ Make many similar graphs
  - ▶ Load multiple files with similar names
  - ▶ Create summary stats with different subsets

Good Functions

1. Make code more readable
2. Avoid coding errors
3. Make you more productive

From "Nice R Code" on github.

However: Never Start Writing a Function by Writing a Function

- ▶ Get one version of your code working first
- ▶ Then build the function
- ▶ When you've been programming for two years, try the function first

What We Cover About Functions

1. Defining a function
2. Calling a function
3. Getting things out of a function
4. Modifying a dataframe
5. Functions and `ggplot`

1. Defining a Function

```
function.name <- function(arg1, arg2){
  # stuff your function does
}
```

- ▶ function.name: what you call the function
- ▶ function: needed to tell R this is a function
- ▶ arg1: first argument of the function
- ▶ arg2: second argument of the function
- ▶ inside the curly braces: what you want the function to do

Simple Function Example

```
summer <- function(x,y){
  x^y
}
```

- ▶ function name?
- ▶ arguments?
- ▶ body of the function?

2. Calling a Function

```
summer <- function(x,y){
  x^y
}
```

```
summer(x = 2,y = 3)
```

2. Calling a Function

```
summer <- function(x,y){
  x^y
}
```

```
summer(x = 2,y = 3)
```

```
## [1] 8
```

2. Calling a Function

```
summer <- function(x,y){
  x^y
}
```

```
summer(x = 2,y = 3)
```

```
## [1] 8
```

```
summer(x = 3,y = 2)
```

2. Calling a Function

```
summer <- function(x,y){
  x^y
}
```

```
summer(x = 2,y = 3)
```

```
## [1] 8
```

```
summer(x = 3,y = 2)
```

```
## [1] 9
```

3. Getting things out of a function

- Suppose you want to use the output of `summer` elsewhere in your program
- Functions "return" the last line
- "Return" means makes a value that exists outside of the function
- Best explained via example

3. Getting things out of a function

- ▶ Suppose you want to use the output of summer elsewhere in your program
- ▶ Functions "return" the last line
- ▶ "Return" means makes a value that exists outside of the function
- ▶ Best explained via example

However, if you save a graph with ggsave() in the function, that will exist outside the function.

```r
summer2 <- function(x,y){
  o1 <- x^y
  o1
  print(paste0("o1 is ", o1))
  o2 <- x + y
  print(paste0("o2 is ", o2))
}
```

```r
summer2(x = 1,y = 2)
```

```
## [1] "o1 is 1"
## [1] "o2 is 3"
```

```
summer2 <- function(x,y){
  o1 <- x^y
  o1
  print(paste0("o1 is ", o1))
  o2 <- x + y
  print(paste0("o2 is ", o2))
}
```

```
summer2(x = 1,y = 2)
```

```
## [1] "o1 is 1"
## [1] "o2 is 3"
```

What if I write o2?

```
summer2 <- function(x,y){
  o1 <- x^y
  o1
  print(paste0("o1 is ", o1))
  o2 <- x + y
  print(paste0("o2 is ", o2))
}
```

```
summer2(x = 1,y = 2)
```

```
## [1] "o1 is 1"
## [1] "o2 is 3"
```

What if I write o2?

```
o2
```

```
## Error in eval(expr, envir, enclos): object 'o2' not found
```

```
summer2 <- function(x,y){
  o1 <- x^y
  print(paste0("o1 is ", o1))
  o2 <- x + y
  print(paste0("o2 is ", o2))
}
```

```
o3 <- summer2(x = 1,y = 2)
```

```
## [1] "o1 is 1"
## [1] "o2 is 3"
```

```r
summer2 <- function(x,y){
  o1 <- x^y
  print(paste0("o1 is ", o1))
  o2 <- x + y
  print(paste0("o2 is ", o2))
}
```

```r
o3 <- summer2(x = 1,y = 2)
```

```
## [1] "o1 is 1"
## [1] "o2 is 3"
```

What if I call o3?

```r
summer2 <- function(x,y){
  o1 <- x^y
  print(paste0("o1 is ", o1))
  o2 <- x + y
  print(paste0("o2 is ", o2))
}
```

```r
o3 <- summer2(x = 1,y = 2)
```

```
## [1] "o1 is 1"
## [1] "o2 is 3"
```

What if I call o3?

```r
o3
```

```
## [1] "o2 is 3"
```

```
summer2 <- function(x,y){
  o1 <- x^y
  print(paste0("o1 is ", o1))
  o2 <- x + y
  #print(paste0("o2 is ", o2))
}
```

```
o3 <- summer2(x = 1,y = 2)
```

```
## [1] "o1 is 1"
```

What Gets Returned, 3 of 4

```
summer2 <- function(x,y){
  o1 <- x^y
  print(paste0("o1 is ", o1))
  o2 <- x + y
  #print(paste0("o2 is ", o2))
}
```

```
o3 <- summer2(x = 1,y = 2)
```

```
## [1] "o1 is 1"
```

What if I call o3?

```r
summer2 <- function(x,y){
  o1 <- x^y
  print(paste0("o1 is ", o1))
  o2 <- x + y
  #print(paste0("o2 is ", o2))
}
```

```r
o3 <- summer2(x = 1,y = 2)
```

```
## [1] "o1 is 1"
```

What if I call o3?

```r
o3
```

```
## [1] 3
```

```
summer2 <- function(x,y){
  o1 <- x^y
  print(paste0("o1 is ", o1))
  o2 <- x + y
  print(paste0("o2 is ", o2))
  return(o2)
}
```

```
o3 <- summer2(x = 1,y = 2)
```

```
## [1] "o1 is 1"
## [1] "o2 is 3"
```

```r
summer2 <- function(x,y){
  o1 <- x^y
  print(paste0("o1 is ", o1))
  o2 <- x + y
  print(paste0("o2 is ", o2))
  return(o2)
}
```

```r
o3 <- summer2(x = 1,y = 2)
```

```
## [1] "o1 is 1"
## [1] "o2 is 3"
```

What if I call o3?

```r
summer2 <- function(x,y){
  o1 <- x^y
  print(paste0("o1 is ", o1))
  o2 <- x + y
  print(paste0("o2 is ", o2))
  return(o2)
}
```

```r
o3 <- summer2(x = 1,y = 2)
```

```
## [1] "o1 is 1"
## [1] "o2 is 3"
```

What if I call o3?

```r
o3
```

```
## [1] 3
```

4. What About Modifying a Dataframe?

```r
# load north korean data
nkd <- data.frame(year = seq(2000,2017,1),
                  defectors = c(0,0,1,0,0,
                                0,0,0,2,0,
                                1,0,3,0,0,
                                1,1,4))
```

What About Modifying a Dataframe?

```
nkd
```

```
##    year defectors
## 1  2000         0
## 2  2001         0
## 3  2002         1
## 4  2003         0
## 5  2004         0
## 6  2005         0
## 7  2006         0
## 8  2007         0
## 9  2008         2
## 10 2009         0
## 11 2010         1
## 12 2011         0
## 13 2012         3
## 14 2013         0
## 15 2014         0
```

First Try

```
addone <- function(fixyear){
  nkd$defectors <- ifelse(test = nkd$year == fixyear,
                          yes = 100,
                          no = nkd$defectors)
}
```

First Try

```
addone <- function(fixyear){
  nkd$defectors <- ifelse(test = nkd$year == fixyear,
                          yes = 100,
                          no = nkd$defectors)
}
```

How do you call this?

First Try

```
addone <- function(fixyear){
  nkd$defectors <- ifelse(test = nkd$year == fixyear,
                          yes = 100,
                          no = nkd$defectors)
}
```

How do you call this?

```
addone(fixyear = 2002)
addone(fixyear = 2005)
nkd
```

```
##   year defectors
## 1 2000         0
## 2 2001         0
## 3 2002         1
## 4 2003         0
## 5 2004         0
```

Second Try: Successful

```r
addone <- function(fixyear){
  nkd$defectors <- ifelse(test = nkd$year == fixyear,
                          yes = 100,
                          no = nkd$defectors)
  return(nkd)
}
nkd <- addone(fixyear = 2002)
nkd <- addone(fixyear = 2005)
nkd
```

```
##   year defectors
## 1 2000         0
## 2 2001         0
## 3 2002       100
## 4 2003         0
## 5 2004         0
## 6 2005       100
## 7 2006         0
```

Create a new variable as a function of an old variable

This doesn't work:

```
multiplypls <- function(varo){
  nkd$new_varo <- nkd$varo * 5
  return(nkd)
}
nkd2 <- multiplypls(var = defectors)
```

R doesn't know to plug in defectors for varo in the dataframe$variable construction

Create a new variable as a function of an old variable

Instead

```r
multiplypls <- function(varo){
  nkd[[paste0("new_",varo)]] <- nkd[[varo]] * 5
  return(nkd)
}
nkd2 <- multiplypls(var = "defectors")
head(nkd2)
```

```
##   year defectors new_defectors
## 1 2000         0             0
## 2 2001         0             0
## 3 2002       100           500
## 4 2003         0             0
## 5 2004         0             0
## 6 2005       100           500
```

5. And a Word of Warning About ggplot()

- ▶ many `tidyverse` commands, including `ggplot()` use non-standard evaluation
- ▶ for your purposes, that means that these command don't always work in expected ways in functions
- ▶ BUT there are work-arounds – see tutorial

Bottom Line

- Use functions!
- Write a non-function example first
- Test
- Write the function
- Check output