Recitation: Functions

Functions

The big idea: A function wraps up a bunch of commands that we run over and over again.

Another perspective: You can also think of a function as a black box that takes in arguments, and returns a result.

A key detail: Things that you define within the function won't be available outside of it, unless they're part of what you `return`. And they'll supercede things defined outside the function.

Wrapping up commands

Let's take an example from the "cats" dataset:

```
library(MASS)
data(cats)
```

Suppose that I'm interested in summary statistics for many different subgroups of cats. I could write a function for that:

```
cat_summary = function(selection){
  mean_Bwt = mean(cats$Bwt[selection])
  std_Bwt  = sd(cats$Bwt[selection])
  mean_Hwt = mean(cats$Hwt[selection])
  std_Hwt  = sd(cats$Hwt[selection])
  to_return = list(mean_Bwt=mean_Bwt,std_Bwt=std_Bwt,mean_Hwt=mean_Hwt,std_Hwt=std_Hwt)
  return(to_return)
}
```

Wrapping up commands (cont.)

So I could look at just heavy male cats:

```
bigboys = (cats$Sex=='M') & (cats$Bwt>=2.5)
cat_summary(bigboys)
```

```
## $mean_Bwt
## [1] 3.04125
##
## $std_Bwt
## [1] 0.38309
##
## $mean_Hwt
## [1] 11.92375
```

```
##
## $std_Hwt
## [1] 2.333425
```

Or all of the female cats:

```
girls = (cats$Sex=='F')
cat_summary(girls)
```

```
## $mean_Bwt
## [1] 2.359574
##
## $std_Bwt
## [1] 0.2739879
##
## $mean_Hwt
## [1] 9.202128
##
## $std_Hwt
## [1] 1.357666
```

The Black Box

Now that I've written cat_summary, I could think of it as a black box – a vector of TRUE/FALSE goes in, summary statistics come out. That's how I'd describe it if I shared it with a friend. This is probably how you think about functions like `mean()` and `plot()` that you've used already.

Note that the number of things that go *in* to the function can vary. Suppose we defined cat_summary a little differently:

```
cat_summary = function(selection,catdata=cats){
  mean_Bwt = mean(catdata$Bwt[selection])
  std_Bwt  = sd(catdata$Bwt[selection])
  mean_Hwt = mean(catdata$Hwt[selection])
  std_Hwt  = sd(catdata$Hwt[selection])
  to_return = list(mean_Bwt=mean_Bwt,std_Bwt=std_Bwt,mean_Hwt=mean_Hwt,std_Hwt=std_Hwt)
  return(to_return)
}
```

We now have some options for how we call `cats_summary`

```
cat_summary(girls)
```

```
## $mean_Bwt
## [1] 2.359574
##
## $std_Bwt
## [1] 0.2739879
##
```

```
## $mean_Hwt
## [1] 9.202128
##
## $std_Hwt
## [1] 1.357666
```

```
cat_summary(girls,cats)
```

```
## $mean_Bwt
## [1] 2.359574
##
## $std_Bwt
## [1] 0.2739879
##
## $mean_Hwt
## [1] 9.202128
##
## $std_Hwt
## [1] 1.357666
```

Again, you've seen this with `plot()`.

The Black Box (cont.)

When you're writing a function you probably make some assumptions about what kind of data you're going to get. For example, I assumed that the vector of TRUE/FALSE and the data frame had the same number of rows. When you make those kinds of assumptions, you should check to make sure that the input you're given matches them. `stopifnot` helps with this.

```
cat_summary = function(selection,catdata=cats){
  stopifnot(length(selection)==nrow(catdata))
  mean_Bwt = mean(catdata$Bwt[selection])
  std_Bwt  = sd(catdata$Bwt[selection])
  mean_Hwt = mean(catdata$Hwt[selection])
  std_Hwt  = sd(catdata$Hwt[selection])
  to_return = list(mean_Bwt=mean_Bwt,std_Bwt=std_Bwt,mean_Hwt=mean_Hwt,std_Hwt=std_Hwt)
  return(to_return)
}
#cat_summary(girls,cats[1:40,])
```

A pesky detail

If you create a variable within the function, it won't "survive" outside the function.

```
numberchanger = function(){
  the_best_number = 4
  return(the_best_number)
}
numberchanger()
```

```
## [1] 4
```

```
#the_best_number
```

A pesky detail (cont.)

If a variable outside the function and inside the function have the same name, the function will ignore the "outside" copy of the variable. It won't modify it, and will work with its own copy.

```
the_best_number = 2
numberchanger = function(){
  the_best_number = 4
  return(the_best_number)
}
numberchanger()
```

```
## [1] 4
```

```
the_best_number
```

```
## [1] 2
```

A pesky detail (cont.)

If a variable isn't defined within the function, but is defined outside, the function will refer to the outside value:

```
the_best_number = 2
numberchanger = function(){
  return(the_best_number)
}
numberchanger()
```

```
## [1] 2
```

```
the_best_number
```

```
## [1] 2
```

But remember, if I try to modify `the_best_number` from inside my function, R will instead create a local copy of `the_best_number`. Generally speaking, getting variable values this way instead of as arguments is a poor programming practice.

A pesky detail (cont.)

Here's a tricky example: what's going to happen?

```
the_best_number = 2
numberchanger = function(){
  the_best_number = the_best_number + 2
  return(the_best_number)
}
numberchanger()
the_best_number
```

A cute example: Recursion

```
fibonacci = function(n){
  stopifnot(n>0)
  if ((n==2) | (n==1)){
    return( 1 )
  } else {
    return( fibonacci(n-1) + fibonacci(n-2))
  }
}
```

```
entries = 1:10
for (i in entries){
  print(fibonacci(i))
}
```

```
## [1] 1
## [1] 1
## [1] 2
## [1] 3
## [1] 5
## [1] 8
## [1] 13
## [1] 21
## [1] 34
## [1] 55
```