Functions tutorial

POLISCI 251A

July 19, 2018

Functions

While most variable types are for storing data, functions let us do things with data. In other words are `verbs` rather than `nouns`. Like environments, they are just another data type that we can assign and manipulate and even pass into other functions.

Functions have the following elements:

1. Arguments: Values passed to a function
2. The body of the function which is bounded by curly braces `{}`
3. In R, the last value that is calculated in the function is automatically returned unless we explicitly design a value with `return`

Let's see an example. The following function calculates the length of the hypotenuse of a right-angled triangle (for simplicity, we'll use the obvious algorithm; for real-world code, this doesn't work well with very big and very small numbers, so you shouldn't calculate hypotenuses this way).

Is always a good practice to provide some comments to your functions expressing the syntax, and usage.

```
################
# Function to calculate hypotenuse
# hypotenuse(x,y)
# x and y are numeric values
################
hypotenuse <- function(x,y){
  sqrt(x^2 + y^2)

}
## We can call this function now
hypotenuse(3,4)
```

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

When we call a function, if we don't name the arguments, then R will match them based on the position. We can also specify a preferred order:

```
hypotenuse(y=24, x=7)
```

```
## [1] 25
```

Note that you can do a little better by including a return argument. Again, this won't change the calculations but is a good coding practice.

```
################
# Function to calculate hypotenuse
# hypotenuse(x,y)
# x and y are numeric values
################
hypotenuse2 <- function(x,y){
  h <- sqrt(x^2 + y^2)
  return(h)
}
## test
hypotenuse2(5,12)
```

```
## [1] 13
```

We can also pass vectors and other functions to our function. Let's create a function that scales a vector using the build in functions `mean()` and `sd()`. Note that you don't have to create those functions since they already exist in R. The arguments m and s are, by default, the mean and standard deviation of the first argument, so the returned vector will have mean 0 and standard deviation 1.

```
## [1] -1.0690450 -0.7126966 -0.1781742  0.5345225  1.4253933
```

```
## [1] -5.572799e-18
```

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

As another example, we can create a function to calculate a sample correlation:

$$r_{X,Y} = \frac{\sum(x-\bar{x})*(y-\bar{y})}{\sqrt{(\sum(x-\bar{x})^2 * \sum(y-\bar{y})^2)}}$$

Here: - x and y refer to vectors of variables. - $\bar{x}$ and $\bar{y}$ refer to the means of x and y respectively.

```
### YOUR CODE HERE
my_cor <- function(x,y){
         mean_x <- mean(x)
         mean_y <- mean(y)
         ## Covariance
         cov_x_y  <- sum((x-mean_x) * (y-mean_y))
         ## sdx
         sdx <- sum((x-mean_x)^2) * sum((y-mean_y)^2)
         ## Cor
         cor_x_y <- cov_x_y/sqrt(sdx)
         return(cor_x_y)

}
```

Let's test this function using the `iraqVote` dataset. Correlate `rep` and `gorevote` `using`my_cor'

```
### Load the pscl library
library(pscl)
```

```
## Warning: package 'pscl' was built under R version 3.5.1
```

```
## Classes and Methods for R developed in the
## Political Science Computational Laboratory
## Department of Political Science
## Stanford University
## Simon Jackman
## hurdle and zeroinfl functions by Achim Zeileis
```

```
## Uncomment this
data(iraqVote)
###
my_cor(iraqVote$rep, iraqVote$gorevote)
```

```
## [1] -0.445052
```

Corroborate your result with the build-in function `cor`

```r
cor(iraqVote$rep, iraqVote$gorevote)
```

```
## [1] -0.445052
```