QUANTITATIVE RESEARCH METHODS

DR. MEIKE MORREN

Lecture 4

contents

- Why functions?
- Structure of a function
  - Store results in object
  - Arguments
  - One vs multiple return values
- Loops
  - for, while, repeat, break, ifelse
  - print
- Vectorization

FUNCTIONS IN R

Functions are…

- Flexible
- (sometimes) Faster

- The foundation of programming

Structure of a function

```
function.name <-
    function(arguments)  {
        purpose of function i.e.
        computations involving the
        arguments
        return(value) / print(value)
    }
```

Anything can be a function

```
greetings <-
    function(partDay, name) {
        message <- paste("Good", partDay)
        persMessage <- paste(message, name, sep=", ")
        print(persMessage)
    }

greetings(partDay="morning",name= "Meike")
```

Arithmetic function

```
f1 <- function(x, y) {
                return(x+y)
        }

f1( 3, 4)

[1] 7
```

Function to calculate the mean

```
meanVec <-
      function(vector) {
         return(sum(vector)/length(vector))
       }

x<-1:10
x
[1] 1 2 3 4 5 6 7 8 9 10

meanVec(x)
[1] 5.5
```

Using the function `mean`

```
meanVec <-
    function(vector) {
      return(mean(vector))
    }
```

Or without explicit return (r returns last value):

```
meanVec <-
    function(vector) {
      mean(vector)
    }
```

Or even more simple…

```
mean(1:10)
```

[1] 5.5

APPLY

Preprogrammed loops

Family ‘apply’ functions loops over rows or columns

- 1 means rows
- 2 means columns

- apply : matrix
- lapply: vector
- tapply: table

- colMeans : matrix

apply

```
df <- cbind(rep(1,5), rep(2,5), rep(3,5),
rep(4,5))

colnames(df) <- c("first","second","third",
"fourth")

apply(df,2,mean)
colMeans(df)
```

Some notes on R functions (1)

- = instead of <- works similarly in R

  but not in functions:

  - `rm(x)  # first remove x`
  - `f(x=1:4); x`

- Usually functions work well with more complicated tasks
- Returns automatically last function, but you can explicitly call return to return any value in between
- Multiple return values become a list

Some notes on R functions (2)

- Short functions can be written on one line without brackets:

```
f1 <- function(x, y) return(x+y)
```

- Multiple commands: include brackets and indicate new line by ;

```
f1 <- function(x, y) {x<-x+1; return(x+y)}
```

LOOP

Types (1)

- For
  - Define for which objects it needs to repeat a command
  - `for (counter in vector) {command}`
- While
  - Execute a command as long as a value meets a condition
  - `while(condition) {command}`
- If else
  - Executes different commands when value meets a condition
  - `if (condition) {command} else {command}`

Types (2)

- Break
  - Identifies break
- Next
  - Goes to next value inside the loop
- Repeat
  - Execute a command until break
  - repeat(command) break

For ....

- Loop to square every element in df
- i = counter

```
df= seq(1, 100, by=2)
df.squared = NULL

for (i in 1:50) {
    df.squared[i] = df[i]^2
}
```

For ....

- Loop to square every element in df
- i = counter

```
df= seq(1, 100, by=2)
df.squared = NULL

for (i in seq(1,50)) {
    df.squared[i] = df[i]^2
}
```

Exercise 4_1.r

- Create function to calculate means for several variables at the same time
  - Without loop
  - With for loop
  - Using colMeans

While..

```
N <- 100
a <- 0
b <- 1
while (a<n){
      a <- b
      b <- a + b
      print(b)
}
```

While..

```
fib <- function(n){
      a <- 0
      b <- 1
      while (a<n){
            a <- b
            b <- a + b
            print(b)
       }
}
fib(100)
```

Break, Next

- break

```
x <- 1
while(x < 5) {x <- x+1; if (x == 3) break; print(x); }
```

- next

```
while(x < 5) {x <- x+1; if (x == 3) next; print(x);}
```

- repeat

Repeat

```
repeat{x<-x+1;if(x==5) break; print(x)}
```

if else

```
x <- 1
y <- 7
if (x==1) {
      print(y)
      } else {
      print("x is not 1")
}
```

if else (within for loop)

```
x <- 1:10
y <- 7
for (i in x){
  if (i>=6) {
     print(y)
     } else {
     print("x is not 1")
  }
}
```

if else (within for loop)

Short version:

```
x <- 1:10
y <- 7
for (i in x){
  if (i>=6) print(y)
     else print("x is not 1")
}
```

NOTE: if else loop can also be used without else

ifelse

- ifelse : same as if else but shorter:

```
ifelse(x>=6, print(y) ,print("x is not 1"))
```

- Useful to recode variables:

```
ifelse(df$age > 70, "older", "younger")
```

- Alternative way to recode (lecture 1 & 2)

```
df$agecat[df$age > 75] <- "Elder"
df$agecat[df$age > 45 & df$age <= 75] <- "Middle Aged"
df$agecat[df$age <= 45] <- "Young"
```

Exercise 4_2.r

- Replace the country codes by the country names
- Use countrycodes.txt (you can find it on bb)

Tips & tricks

- Make code faster by subsetting:

- Select the rows on which you want to execute your command (in this case assign missings to second column for those with values of x in first column):
  - `rows <- df[df$V1==x,]`
  - `rows[,"V2"] <- NA`

- Set these values back into the dataframe
  - `df[df$V1==x,] <- rows`

Exercise 4_2.r

- Make code faster by subsetting

Next week…

Regression