Psychology 350: An introduction to R for Psychological Research
Week 5b: Loops and flow control

William Revelle
Northwestern University
Evanston, Illinois USA

https://personality-project.org/courses/350

Outline

One of the most powerful operations in any programming language is the loop

1. Computers are very good at doing complex calculations
2. But even more useful is doing these again!
3. Doing a simple or complex calculation multiple times is one of the great powers of computers
4. You can repeat an operation in a *loop*.
5. There are many ways of doing loops, we will examine several.

Consider some basic operation that you do repeatedly

R code

```
x  <- 1
print(x)
x <- x + 1
print(x)
x <- x + 1
print (x)
...
```

```
x  <- 1
> print(x)
[1] 1
> x <- x + 1
> print(x)
[1] 2
> x <- x + 1
> print (x)
[1] 3
```

Making this a little better

Save the results, only print once

R code

```
result <- rep(NA,10)  #we will do this 10 times
i<- 1
result[i] <- i
i <- i + 1
result[i] <- i
i <- i + 1
result[i] <- i
i <- i + 1
result[i] <- i
i <- i + 1
...
print(result)
```

```
 i<- 1
 > result[i] <- i
> i <- i + 1
> result[i] <- i
> i <- i + 1
> result[i] <- i
> i <- i + 1
> result[i] <- i
> i <- i + 1

> print(result)
 [1]  1  2  3  4 NA NA NA NA NA NA
```

Somewhat better, store the results in a dynamic list

Do something more interesting

R code

```
x <- 2
result <- list()
i<- 1
 result[i] <- x ^ i
 i <- i + 1
 result[i] <- x ^ i
 i <- i + 1
 result[i] <- x ^ i
  i <- i + 1
 result[i] <- x ^ i
  i <- i + 1
 result[i] <- x ^ i
  i <- i + 1
 result[i] <- x ^ i
 print(result)
```

```
>  print(result)
[[1]]
[1] 2
[[2]]
[1] 4
[[3]]
[1] 8
[[4]]
[1] 16
[[5]]
[1] 32
[[6]]
[1] 64
```

unlist the list

Do something more interesting

R code

```
x <- 2
result <- list()
i<- 1
 result[i] <- x ^ i
 i <- i + 1
 result[i] <- x ^ i
 i <- i + 1
 result[i] <- x ^ i
  i <- i + 1
 result[i] <- x ^ i
  i <- i + 1
 result[i] <- x ^ i
  i <- i + 1
 result[i] <- x ^ i
 result <- unlist(result)
 print(result)
```

```
>  result <- unlist(result)
>  print(result)
[1]  2  4  8 16 32 64
```

Conditional loops automate the procedure

From the help page help("for")

1. These are the basic control-flow constructs of the R language. They function in much the same way as control statements in any Algol-like language. They are all reserved words.
2. Usage
   - if(cond) expr
   - if(cond) cons.expr else alt.expr
   - for(var in seq) expr
   - while(cond) expr
   - repeat expr
   - break
   - next

Lets see if we can elaborate that discussion of if

What is missing is some implied operation

1. `if`(some logical condition is TRUE) [then] {do something}
2. `if`(some logical condition is TRUE) [then] {do something} else {do something else}
3. Consider this example:

R code

```
now <- as.POSIXlt(Sys.time())$hour
   if (now < 12) {
   cat("Good morning Bill. \nAre you ready to have fun?")
   } else { if (now < 18) {cat("Good afternoon Bill.\nAre you ready to have fun?")
         } else {
    cat("Good evening Bill. \nAre you ready to have fun?")}
    }
```

```
now <- as.POSIXlt(Sys.time())$hour
>    if (now < 12) {
+    cat("Good morning Bill. \nAre you ready to have fun?")
+    } else { if (now < 18) {cat("Good afternoon Bill.\nAre you ready to have fun?")
+          } else {
+     cat("Good evening Bill. \nAre you ready to have fun?")}
+     }
Good evening Bill.
Are you ready to have fun?
```

Elaborating the print function

We use the cat function to concatenate the output and "n" for line feeds.

R code

```
my.name <-  readline("Hi, what is your name?" )
#need to pause for some input
now <- as.POSIXlt(Sys.time())$hour
   if (now < 12) {
   cat("Good morning ",my.name, " \nAre you ready to have fun?")
   } else { if (now < 18) {cat("Good afternoon ", my.name,
                "\nAre you ready to have fun?")
         } else {
    cat("Good evening  ", my.name,"  \nAre you ready to have fun?")}
    }
```

```
Hi, what is your name? Bill

Good evening   Bill
Are you ready to have fun?
```

The for loop is perhaps the easiest to use

1. for(var in seq) expr
2. increment var from the lowest to the highest value in the sequence while doing some expression
3. Typically use i, j, k as the index variable, but this is not necessary

R code

```
for ( i in 1:10) {cat( i, "  ",2^i,"\n") }
 #use the cat function for fancy printing
 # "\n" in a print is a line feed
```

```
for ( i in 1:10) {cat( i, "  ",2^i,"\n") }
1    2
2    4
3    8
4    16
5    32
6    64
7    128
8    256
9    512
10    1024
```

generalize this loop

R code

```
x <- 3
for ( i in 1:10) {cat( i, "  ",x^i,"\n") }
```

```
x <- 3
> for ( i in 1:10) {cat( i, "  ",x^i,"\n") }
1    3
2    9
3    27
4    81
5    243
6    729
7    2187
8    6561
9    19683
10    59049
```

Can control the loop by what is in sequence part

R code

```
for ( i in seq(1,17,2)) {cat( i, "  ",2^i,"\n") }
for ( i in seq(20,1,-2)) {cat( i, "  ",2^i,"\n") }
```

```
or ( i in seq(1,17,2)) {cat( i, "  ",2^i,"\n") }
1    2
3    8
5    32
7    128
9    512
11    2048
13    8192
15    32768
17    131072


> for ( i in seq(20,1,-2)) {cat( i, "  ",2^i,"\n") }
20    1048576
18    262144
16    65536
14    16384
12    4096
10    1024
8    256
6    64
4    16
2    4
```

Apply this loop to some data

R code

```
my.data <- attitude
nvar <- NCOL(attitude)
for (i in 1:nvar) {print(mean(my.data[,i]))}
```

```
 my.data <- attitude
> nvar <- NCOL(attitude)
> for (i in 1:nvar) {print(mean(my.data[,i]))}
[1] 64.63333
[1] 66.6
[1] 53.13333
[1] 56.36667
[1] 64.63333
[1] 74.76667
[1] 42.93333
```

Somewhat more elegant

R code

```
my.data <- attitude
nvar <- NCOL(my.data)
for (i in 1:nvar) {cat(colnames(my.data)[i],"  ",
        mean(my.data[,i]),"\n")}
```

```
for (i in 1:nvar) {cat(colnames(my.data)[i],"  ",mean(my.data[,i]),"\n")}
rating    64.63333
complaints    66.6
privileges    53.13333
learning    56.36667
raises    64.63333
critical    74.76667
advance    42.93333
```

Even better

R code

```
my.data <- attitude
nvar <- NCOL(my.data)
for (i in 1:nvar) {cat(colnames(my.data)[i],"  ",
      mean(my.data[,i]),"\n")}
```

```
for (i in 1:nvar) {cat(colnames(my.data)[i],"  ",round(mean(my.data[,i]),2),"\n")}
rating    64.63
complaints    66.6
privileges    53.13
learning    56.37
raises    64.63
critical    74.77
advance    42.93
```

Clean up the loop by using a data frame

R code

```
my.data <- attitude
nvar <- NCOL(my.data)
df <- data.frame(var = 1:nvar, mean=rep(NA,nvar))
for(i in 1:nvar) {df[i,1] <- colnames(my.data)[i]
df [i, 2] <- mean(my.data[,i])
}
df
```

```
df
         var     mean
1     rating 64.63333
2 complaints 66.60000
3 privileges 53.13333
4   learning 56.36667
5     raises 64.63333
6   critical 74.76667
7    advance 42.93333
```

Try this on a different data set

R code

```
my.data <- sat.act
nvar <- NCOL(my.data)
df <- data.frame(var = 1:nvar, mean=rep(NA,nvar))
for(i in 1:nvar) {df[i,1] <- colnames(my.data)[i]
df [i, 2] <- mean(my.data[,i])
}
df
```

```
df
        var       mean
1    gender   1.647143
2 education   3.164286
3       age  25.594286
4       ACT  28.547143
5      SATV 612.234286
6      SATQ         NA      <- What happened here
```

remove missing data

R code

```
my.data <- sat.act
nvar <- NCOL(my.data)
df <- data.frame(var = 1:nvar, mean=rep(NA,nvar))
for(i in 1:nvar) {df[i,1] <- colnames(my.data)[i]
df [i, 2] <- mean(my.data[,i], na.rm=TRUE)}
df
```

```
my.data <- sat.act
> nvar <- NCOL(my.data)
> df <- data.frame(var = 1:nvar, mean=rep(NA,nvar))
> for(i in 1:nvar) {df[i,1] <- colnames(my.data)[i]
+ df [i, 2] <- mean(my.data[,i], na.rm=TRUE)}
> df
        var       mean
1    gender   1.647143
2 education   3.164286
3       age  25.594286
4       ACT  28.547143
5      SATV 612.234286
6      SATQ 610.216885
```

Improve this loop by adding more output

R code

```
my.data <- sat.act
nvar <- NCOL(my.data)
df <- data.frame(varname = 1:nvar, cases=rep(NA,nvar),
     mean=rep(NA,nvar))

for(i in 1:nvar) {df[i,1] <- colnames(my.data)[i]
df[i,2] <- sum( !is.na(my.data[,i] ))
df [i, 3] <- mean(my.data[,i], na.rm=TRUE)}
df
```

```
my.data <- sat.act
 my.data <- sat.act
> nvar <- NCOL(my.data)
> df <- data.frame(varname = 1:nvar, cases=rep(NA,nvar),
    mean=rep(NA,nvar))
>
> for(i in 1:nvar) {df[i,1] <- colnames(my.data)[i]
+ df[i,2] <- sum( !is.na(my.data[,i] ))
+ df [i, 3] <- mean(my.data[,i], na.rm=TRUE)}
> df
    varname cases       mean
1    gender   700   1.647143
2 education   700   3.164286
3       age   700  25.594286
4       ACT   700  28.547143
5      SATV   700 612.234286
6      SATQ   687 610.216885
```

Improve this loop by adding more output

R code

```
my.data <- sat.act
digits <- 2
nvar <- NCOL(my.data)
df <- data.frame(varname = 1:nvar, cases=rep(NA,nvar),
     mean=rep(NA,nvar), sd = rep(NA,nvar))

for(i in 1:nvar) {df[i,1] <- colnames(my.data)[i]
df[i,2] <- sum( !is.na(my.data[,i] ))
df [i, 3] <- round(mean(my.data[,i], na.rm=TRUE), digits=digits)
df[i,4] <- round(sd(my.data[,i] , na.rm = TRUE), digits = digits)}
df
```

```
my.data <- sat.act
> digits <- 2
> nvar <- NCOL(my.data)
> df <- data.frame(varname = 1:nvar, cases=rep(NA,nvar),
+      mean=rep(NA,nvar), sd = rep(NA,nvar))
>
> for(i in 1:nvar) {df[i,1] <- colnames(my.data)[i]
+ df[i,2] <- sum( !is.na(my.data[,i] ))
+ df [i, 3] <- round(mean(my.data[,i], na.rm=TRUE), digits=digits)
+ df[i,4] <- round(sd(my.data[,i] , na.rm = TRUE), digits = digits)}
> df
    varname cases   mean     sd
1    gender   700   1.65   0.48
2 education   700   3.16   1.43
3       age   700  25.59   9.50
```

Bundle this loop into a baby function

R code

```
my.describe <- function(my.data, digits=2)  {
nvar <- NCOL(my.data)
df <- data.frame(varname = 1:nvar, cases=rep(NA,nvar),
     mean=rep(NA,nvar), sd = rep(NA,nvar))

for(i in 1:nvar) {df[i,1] <- colnames(my.data)[i]
df[i,2] <- sum( !is.na(my.data[,i] ))
df [i, 3] <- round(mean(my.data[,i], na.rm=TRUE), digits=digits)
df[i,4] <- round(sd(my.data[,i] , na.rm = TRUE), digits = digits)}
df }  #the variable on the last line is automatically returned.

my.describe(bfi[1:10])
```

```
my.describe(bfi[1:10])
   varname cases mean   sd
1       A1  2784 2.41 1.41
2       A2  2773 4.80 1.17
3       A3  2774 4.60 1.30
4       A4  2781 4.70 1.48
5       A5  2784 4.56 1.26
6       C1  2779 4.50 1.24
7       C2  2776 4.37 1.32
8       C3  2780 4.30 1.29
9       C4  2774 2.55 1.38
10      C5  2784 3.30 1.63
```

apply does loops automatically

1. Because so much of data analysis is looping over variables or subjects, R has a way of making this easier
2. apply Returns a vector or array or list of values obtained by applying a function to margins of an array or matrix.
3. apply(X, MARGIN, FUN, ..., simplify = TRUE)

R code

```
my.data <- sat.act
apply(my.data, 2, mean)    #does not work for missing values
apply(my.data, 2, function(x)  mean(x, na.rm=TRUE))
```

```
my.data <- sat.act
> apply(my.data, 2, mean)
    gender  education        age        ACT       SATV       SATQ
  1.647143   3.164286  25.594286  28.547143 612.234286         NA
> apply(my.data, 2, function(x)  mean(x, na.rm=TRUE))
    gender  education        age        ACT       SATV       SATQ
  1.647143   3.164286  25.594286  28.547143 612.234286 610.216885
```

apply can be applied to rows as well as columns

R code

```
apply(attitude, 1,mean)
#compare to
rowMeans(attitude)
```

```
apply(attitude, 1,mean)
 [1] 51.57143 59.28571 69.71429 55.57143 68.85714 46.85714 56.00000 61.14286 68.28571 57.5714
[11] 55.28571 55.28571 53.28571 64.28571 68.85714 63.28571 73.28571 65.71429 63.42857 60.5714
[21] 42.42857 60.85714 57.28571 47.57143 54.42857 72.57143 70.28571 52.28571 74.00000 63.2857
> #compare to
> rowMeans(attitude)
 [1] 51.57143 59.28571 69.71429 55.57143 68.85714 46.85714 56.00000 61.14286 68.28571 57.5714
[11] 55.28571 55.28571 53.28571 64.28571 68.85714 63.28571 73.28571 65.71429 63.42857 60.5714
[21] 42.42857 60.85714 57.28571 47.57143 54.42857 72.57143 70.28571 52.28571 74.00000 63.2857
```

Generalizations of apply

1. lapply returns a list of the same length as X, each element of which is the result of applying FUN to the corresponding element of X.
2. sapply is a user-friendly version and wrapper of lapply by default returning a vector, matrix or, if simplify = "array", an array if appropriate, by applying simplify2array(). sapply(x, f, simplify = FALSE, USE.NAMES = FALSE) is the same as lapply(x, f).
3. mapply is a multivariate version of sapply. mapply applies FUN to the first elements of each ... argument, the second elements, the third elements, and so on. Arguments are recycled if necessary.

while , repeat

1. do a certain operation until some condition is met
2. WARNING, make sure that condition can be met, or you will have an infinite loop!
3. might be some iteration until some condition is close enough to zero
4. Think back to our square root function we could quit if the error is very small (will never achieve zero because of accuracy effects).
5. anything less than $|2.220446e-16|$ is treated as 0

while error > .Machine$double.eps

R code

```
iterative.sqrt <- function(X,guess) { if(missing(guess)) guess <- 1
error <-  10
 guess <- 1 #a dumb guess

  iterate <- function(X,guess) {
low <- guess
high <- X/guess
guess <- ((high+low)/2)
 guess}

while(abs(error ) > .Machine$double.eps) {
 answer <- iterate(X, guess)
 error <- answer - guess
 guess <- answer }

cat("The square root of", X ," is ", answer)
}
```

```
iterative.sqrt(9)
The square root of 9  is  3
> iterative.sqrt(17)
The square root of 17  is  4.123106
```

Recursion allows a function to operate on itself

1. Classic example is that of N!
2. N! = N * (N-1)!
3. 0! = 1
4. Note that it needs a way to stop

R code

```
Nfactorial <- function(N){
if (N==0) {1} else {N* Nfactorial(N-1)}
}
```

```
 for(i in 1:10) cat( i, "  ", Nfactorial(i) ,"\n")
 for(i in 1:10) cat( i, "  ", Nfactorial(i) ,"\n")
1    1
2    2
3    6
4    24
5    120
6    720
7    5040
8    40320
9    362880
10    3628800
```