Logical, ifs, loops

UO

1 feb, 2015

For this tutorial we will use the data of reproductive traits in lizards on different islands (found in the website)

```
# set the working directory
setwd("~/Dropbox/Rworkshop/2014/Databases")
# read the data into R
data<-read.csv("island_type_final2.csv",header=T)
```

Logical operators

We will start by geting familiarized with the logical class and logical operators in R.

Logical (or Boolean) data types have two values: *TRUE* or *FALSE*. They are created by comparison of objects, using the following operators:

equal: == not equal: != greater/less than: > < greater/less than or equal: >= <=

```
x=1;y=2;
z=x>y
z
```

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

```
z=x<=y
z
```

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

```
z=x==y
z
```

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

Standard logical operators are "&" (and), "|" (or) and "!" (negation).

```
u = TRUE; v = FALSE
u & v          # u AND v
```

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

```
u | v          # u OR v
```

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

```
!u              # negation of u
```

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

What happens with objects containing multiple logical values?

```
U=c(TRUE,TRUE,FALSE)
!U
```

```
## [1] FALSE FALSE  TRUE
```

```
V=1:6 # 1 to 6
V
```

```
## [1] 1 2 3 4 5 6
```

```
V>2
```

```
## [1] FALSE FALSE  TRUE  TRUE  TRUE  TRUE
```

```
V<=4
```

```
## [1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE
```

```
(V>2)&(V<=4)
```

```
## [1] FALSE FALSE  TRUE  TRUE FALSE FALSE
```

```
(V>2)|(V<2)
```

```
## [1]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
```

if statements

if statements execute a chunk of code in a desired scope (within the {} brackets) if given a TRUE value.

```
x=2>1
if(x){
  print('Nothing but the truth!')
}
```

```
## [1] "Nothing but the truth!"
```

another example

```
x=2
y=4
if(x>y){
  print("y is always better")
}
```

in this case nothing gets printed. However, if we change the symbol direction we'll see it get printed

```
x=2
y=4
if(x<y){
  print("y is always better")
}
```

```
## [1] "y is always better"
```

They can be complimented by else statements:

NOTE: the 'else' statement must be in the same line as the the curly brackets ending the previous 'if' statement!

```
Y=1>2
if(Y){
  print('Nothing but the truth!')
}else {
  print("Lies! It's all Lies!")
}
```

```
## [1] "Lies! It's all Lies!"
```

We can also use 'else if'

```
Grade=75
if(Grade>=95){
  print('Excellent!')
}else if (Grade>=85){
  print("Pretty good")
} else {
  print("You are stupid")

}
```

```
## [1] "You are stupid"
```

Loops-for, while, break and next

for loops for loops execute a chunk of code within their scope, as long as some variable is part of a predesignated sequence.

```
for (variable in 1:5){

  print(variable)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

They can be more abstract as well:

```
sequence=levels(data$species)[1:5]
sequence
```

```
## [1] "Afroablepharus_annobonensis" "Ailuronyx_seychellensis"
## [3] "Ailuronyx_tachyscopaeus"     "Ailuronyx_trachygaster"
## [5] "Algyroides_fitzingeri"
```

```
for (i in sequence){
  print (i)
}
```

```
## [1] "Afroablepharus_annobonensis"
## [1] "Ailuronyx_seychellensis"
## [1] "Ailuronyx_tachyscopaeus"
## [1] "Ailuronyx_trachygaster"
## [1] "Algyroides_fitzingeri"
```

This type of loops are very useful. They can also be used for calculations of subset data and many other things Lets try an example where we break the data to families and calculate the mean clutch size and body mass for each family. The results of this calculation we'll put in a new table

First we'll create the vector with all the family names

```
family<-levels(data$family)
```

then we'll create the output table. We can create a matrix, array or list and then convert it to a data frame. Here I use a matrix

```
# create the matrix and convert it to a data frame
family.data<-as.data.frame(matrix(NA,ncol=3,nrow=length(family)))
# add names to columns
names(family.data)<-c("family","mean_clutch","mean_mass")
```

Now we'll work the steps to create the loop: First we create all functions as if we want the mean clutch size and mean body mass only for the first family

1. subset the data to the first family
2. put the name of the family in the first cell of the first column
3. calculate the mean clutch size for that family and save it to the first cell of the second column
4. Calculate the mean body mass of the family and save it to the first cell of the third column
5. Generalize the code: repeat the process as the amount of the families we have

```
# step 1
new.data<-subset(data,family=="Agamidae")
# step 2
family.data[1,1]<-"Agamidae"
# step 3
family.data[1,2]<-mean(new.data$clutch,na.rm=T)
# step 4
family.data[1,3]<-mean(new.data$mass)
```

Step 5: Now we want to generalize the functions so that we can put them in a for loop The first thing we'll do is to change the name of the family to the place of a family in our 'family' vector And we'll do the same for the name that goes into our new data frame Everything else stays almost the same we just change the number of rows to i

```
# start the loop fro 1 to the length of the family vector
for(i in 1:length(family)){
  # create a subset of the data from the large dataset using the name of the family in position i in th
  new.data<-subset(data,family==family[i])
  # put the name of the family in position i in the family vector to the cell in position i in the firs
  family.data[i,1]<-family[i]
  #calculate the mean clutch size. Because there are NA's in the clutch size we need to make sure to re
  family.data[i,2]<-mean(new.data$clutch,na.rm=T)
  # calculate the mean body mass for the i family and save it to cell i in the third column
  family.data[i,3]<-mean(new.data$mass)
}
```

Everything that we showed in this loop is done today by the ddply function in the plyr package but it should give you enough understanding on how for loops work and how to build them.

while loops while loops execute a chunk of code within their scope, while some logical conditions is satisfied.

Note that if your condition will not change to FALSE, your loop will run infinitely!

```
i=0
while(i<5){#very similar to for loop
  i=i+1
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

```
X=TRUE
i=0
while(X){
  i=i+1
  print(i)
  if(i==5){
    X=FALSE
  }
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

'break' and 'next' break and next are used to exit the loops and go to the next value, respectively:

```
for (i in 1:10){
    if(i>7){
    break
  }

  if((floor(i/2)-i/2)==0)#check for even integers
    {
    next
    }
  print(i)

}
```

```
## [1] 1
## [1] 3
## [1] 5
## [1] 7
```

A complex example examples

The next function orders the species of each archipelago by mass and takes the lightestspecies that account for at least half of the overall species mass, then saves those species into a list.

```
specieslist=list()
for( i in 1:length(levels(data$Archipelago))){
  sub1=subset(data,Archipelago==levels(data$Archipelago)[i])
  mass=0
  count=0
  ArchiVec=c()
  while(mass<=0.5){
    count=count+1
    mass=mass+sort(exp(sub1$mass))[count]/sum(exp(sub1$mass))
    ArchiVec[count]=as.character(sub1$species[order(sub1$mass)[count]])
  }
  specieslist[[i]]=ArchiVec


}
names(specieslist)=levels(data$Archipelago)
specieslist[1:3]
```

```
## $Andaman_and_Nicobar_Islands
## [1] "Lipinia_macrotympanum"   "Phelsuma_andamanense"
## [3] "Ptychozoon_nicobarensis"
##
## $Aru_Islands
## [1] "Varanus_beccarii"
##
## $Bahamas
## [1] "Aristelliger_barbouri"     "Leiocephalus_inaguae"
```

```
## [3] "Leiocephalus_psammodromus" "Cyclura_rileyi"
## [5] "Cyclura_carinata"
```

NOTE: if you are running a long loop, try to initialize the needed vectors instead of letting them grow within the loop- this is MUCH quicker. If you do not know the size you will need, but can bound ir from above, it will probably still be more efficient to allocate the guesstimated vector size in advance.

```
#Allocating in advance
start.time <- Sys.time()
bar = seq(1,200000, by=2)
bar.squared = rep(NA, 200000)

for (i in 1:length(bar) ) {
bar.squared[i] = bar[i]^2
}

#get rid of excess NAs
bar.squared = bar.squared[!is.na(bar.squared)]

end.time <- Sys.time()
time.taken <- end.time - start.time
time.taken
```

```
## Time difference of 0.26612 secs
```

```
#Letting the object grow while looping

start.time <- Sys.time()
bar = seq(1, 200000, by=2)
bar.squared = NULL

for (i in 1:length(bar) ) {
bar.squared[i] = bar[i]^2
}
end.time <- Sys.time()
time.taken <- end.time - start.time
time.taken
```

```
## Time difference of 24.1313 secs
```