# Loops and Functions

## If/else conditional statements

### Set up

An if/else conditional statement is set up as follows: "if (condition) {Execute some code} else {execute some other code}". First, R will check if the condition you've specified is met. If so, it will execute the first chunk of code. Otherwise (else), it will execute the second chunk of code. NOTE: unlike other languages, the *else* statement must be linked to the *if* (you cannot have a "CR/LF" between them as in many compiled languages). In the examples that follow, the curly braces are used to establish the link.

### Credit card company example

Please note that the following example is not representative of an actual credit card application process.

Suppose that a credit card company exclusively uses credit scores (a value between 300-850 that supposedly measures how likely you are to pay your bills on time) to determine whether or not a potential application is approved (note: for simplicity, we will assume that people with credit scores between 300 and 850 are all equally likely to apply). The company uses the following system: People with credit scores less than 500 are denied and people with credit scores greater than or equal to 500 are approved.

For any given credit score, we can use an if/else statement to determine whether the application will be approved or denied.

```r
# First we randomly sample a credit score for a potential applicant
creditscore <- sample(seq(300, 850, by = 1), size = 1)

# If the credit score is less than 500, the credit card is not approved
if (creditscore < 500) {
    creditcard = "not approved"

} else {
    # otherwise...
    creditcard = "approved"
}

# This line prints out the outcome of our credit card application
paste("Your credit card application was", creditcard)
```

```
## [1] "Your credit card application was approved"
```

Now suppose that, once you are approved, your credit limit is determined by your credit score. There are two possible credit limits for applicants, so there are actually 3 possible outcomes for any given application: If the credit score is less than 500, the application is denied; if the credit score is in the range 500-699, the card is approved with a credit limit of 2000 dollars; and if the credit score is in the range 700-850, the card is approved with a credit limit of 10,000 dollars. We can use an "else if" statement in order to check another condition before the "else" statement:

```r
# First we randomly sample a credit score for a potential applicant
creditscore <- sample(seq(300, 850, by = 1), size = 1)

# If the credit score is less than 500, the credit card is not approved
if (creditscore < 500) {
    creditcard = "not approved"

} else if (creditscore < 700) {
    # otherwise if it is less than 700...
    creditcard = "approved (credit limit $2000)"

} else {
    # otherwise...
    creditcard = "approved (credit limit $10,000)"
}

# This line prints out the outcome of our credit card application
paste("Your credit card application was", creditcard)
```

```
## [1] "Your credit card application was not approved"
```

## Functions

Now suppose that the credit card company wants to create an online system where people can input their credit scores and quickly determine their approval status and credit limit. Instead of changing the value of "creditscore" every time and re-running the code, we can create a function that takes in a credit score and spits out the application result. This way, any time we want to determine the result of a new application, we only have to run one line of code! In general, anytime you have to copy and paste code *more than twice* (even if you are tweaking inputs slightly), that code is probably a good candidate for a function.

Functions are created using 'function()' and are stored as R obejcts in your global environemnt. They are R objects of class "function".

Functions follow the following structure:

myFunction <- function( arguments ){
execute some interesting code #this last line is used to return objects by default }

Using the above credit card application process, we can create a function that takes in a numerical credit score (x) and determines the result of the application. We will use the same conditions as above.

```
# name the function and set the arguments of the function note: you could
# include multiple inputs separated by commas
cc_app_function <- function(creditscore) {

    # If the credit score is less than 500, the credit card is not approved
    if (creditscore < 500) {
        creditcard = "not approved"

    } else if (creditscore < 700) {
        # otherwise if it is less than 700...
        creditcard = "approved (credit limit $2000)"

    } else {
        # otherwise...
        creditcard = "approved (credit limit $10,000)"
    }

    return(creditcard)  # this tells R that the function will return the value of 'creditcard'
}
```

Now let's make sure the function works. What is the result of my credit card application if my credit score is 300? 650? 790?

```r
cc_app_function(300)
```

```
## [1] "not approved"
```

```r
cc_app_function(650)
```

```
## [1] "approved (credit limit $2000)"
```

```r
cc_app_function(790)
```

```
## [1] "approved (credit limit $10,000)"
```

Now suppose that the credit card company wants to add a more personal critereon to the application process: some applicants have the opportunity to speak with a representative on the phone. The company representative can automatically approve (with a 10,000 dollar limit) any applicant that they speak to, regardless of their credit score, by giving their application a stamp of approval. The company wants to add an input to the above function, indicating whether or not the application received a stamp of approval:

```r
# add another input called stamp. Set stamp=FALSE as a default
cc_app_function2 <- function(creditscore, stamp=FALSE) {

  #first check if the application has a stamp of approval
  #note: we also could have just written "stamp" in the conditional
  #because "stamp" and "stamp==TRUE" both evaluate to the whether or not stamp is true.
  if(stamp==TRUE){
    creditcard = "approved (credit limit $10,000)"
  }

  else { #Otherwise (i.e., if stamp==FALSE)...
    # If the credit score is less than 500, the credit card is not approved
    if (creditscore < 500){
    creditcard = "not approved"

    } else if(creditscore < 700) { # otherwise if it is less than 700...
      creditcard = "approved (credit limit $2000)"

    } else { # otherwise...
      creditcard = "approved (credit limit $10,000)"
    }
  }

  return(creditcard) # this tells R that the function will return the value of "creditcard"
}
```

Note that, because stamp=FALSE by default in the header to the function, we do not have to specify "stamp" in the function input unless we want to set stamp=TRUE. Therefore, we can use this new function exactly as before (we just have one new option if we want to use it):

```r
cc_app_function2(300)
```

```
## [1] "not approved"
```

```r
cc_app_function2(300, stamp = TRUE)
```

```
## [1] "approved (credit limit $10,000)"
```

```r
cc_app_function2(650)
```

```
## [1] "approved (credit limit $2000)"
```

```r
cc_app_function2(790)
```

```
## [1] "approved (credit limit $10,000)"
```

## For loops

Suppose that we want to repeat this process many times, for many potential scores (which could be saved in a vector). One way to do this is by using a for loop to iterate through various scores and determine the outcome for each (which can also be saved in a vector).

First, lets create a fake dataset of potential credit card applicants:

```r
#this makes the results of this chunk reproducible (the sequence of random numbers will be identical up
set.seed(333)

#take a sample of 10 random credit scores and approval stamps
creditscores <- sample(seq(300, 850, by=1), size = 10, replace=TRUE)
stamps <- sample(c(TRUE, FALSE), size=10, replace=TRUE)

#create a data frame called data
data <- data.frame(score=creditscores, stamp=stamps)

#inspect data
data
```

```
##    score stamp
## 1    724  TRUE
## 2    610  TRUE
## 3    594 FALSE
## 4    615  TRUE
## 5    301 FALSE
## 6    366  TRUE
## 7    514 FALSE
## 8    658 FALSE
## 9    339 FALSE
## 10   460  TRUE
```

Now, let's create a vector of application results for each of these simulated applicants.

Note that, if we want to save results in a vector, it is good practice to initialize a vector of the proper length before starting to fill it.

In each iteration of the loop below, the result of each credit card application (corresponding to a row of the dataset above) is calculated and stored in a vector called app_results. We can then inspect a table of the results.

```r
# save sample size (the number of rows in the dataset)
samp_size <- nrow(data)

# initialize a vector of the same length called app_results
app_results <- rep(NA, samp_size)

# iterate through each row of 'data' and save the application result
for (i in 1:samp_size) {
    app_results[i] <- cc_app_function2(data$score[i], data$stamp[i])
}
```

```r
# inspect a table of the results
table(app_results)
```

```
## app_results
## approved (credit limit $10,000)   approved (credit limit $2000)
##                               5                               3
##                    not approved
##                               2
```

## While loops

Suppose that the credit card company can only approve 50 credit card applications with a credit limit of 10,000. We can use a while loop to estimate how many credit card applications will be considered before 50 high limit cards are approved. A while loop will continue to run as long as a given condition is met. NOTE: use of a for loop would be inelegant here as we do not know when it is to terminate; while loops are preferred in this context. The set up is: while(condition) {run some code}.

```r
# create two counter variables: total_apps: keeps track of the total number
# of applications that are reviewed high_limit_apps: keeps track of the
# total number of high-limit applications that are approved
total_apps <- 0
high_limit_apps <- 0

# while the number of high limit approvals is <=50, continue accepting
# applications
while (high_limit_apps <= 50) {

    # simulate a random credit score and result of 'stamp'
    creditscore <- sample(seq(300, 850, by = 1), size = 1)
    stamp <- sample(c(TRUE, FALSE), size = 1)

    # find out the application result
    app_result <- cc_app_function2(creditscore, stamp)

    # increase the total apps counter by 1
    total_apps <- total_apps + 1

    # if a high limit credit card is approved, increase the high limit counter
    # by 1
    if (app_result == "approved (credit limit $10,000)") {
        high_limit_apps <- high_limit_apps + 1
    }
}

# print the total number of applications that were reviewed
total_apps
```

```
## [1] 77
```

## Practice Problems

1. Suppose you are playing a game: You pick three numbers in the range 1-10. Then, a card is randomly drawn from a 10-card deck (each card has a number between 1 and 10 written on it). If the randomly drawn card matches any of the three numbers you chose, you win. Otherwise, you lose. Write a function that takes one input: a vector of the three numbers that you chose. The function should have one output: a character string indicating whether you won or lost the game. Write the function in the space below.
2. Use a for loop to repeat the following process 100 times: randomly choose 3 numbers from 1-10 to play the game with (all values are equally likely) and record whether you won or lost the game in a vector of length 100. Print a table of your wins and losses.
3. Create a function that takes in a vector of data and returns (one of) the mode(s) of the data. (Hint: the supplementary R code for the math/stats review packet includes some code to find the mode of a vector). Challenge problem: write a function that returns a vector of all modes in a dataset (if there is more than one mode). Hint: the which() function might be of use.
4. Use the mode() function that you just created to find the mode of each column in the simulated dataset (use a for loop to accomplish this). Instead of saving the results in a vector, print each of the results using the print() function (note: if you don't use print() within a loop or function, nothing will print out when you run the code or call the function).

```
dat1 <- data.frame(X1 = c(1,1,2,3,4,rep(5,5),2,rep(7,7)),
                   X2 = c(3,9,18,27,27,20,40,1,1,1,3,4,1,rep(1,5)),
                   X3 = c(rep(4,4), rep(2,7), rep(10,7)))
```

## Answers

1. Suppose you are playing a game: You pick three numbers in the range 1-10. Then, a card is randomly drawn from a 10-card deck (each card has a number between 1 and 10 written on it). If the randomly drawn card matches any of the three numbers you chose, you win. Otherwise, you lose. Write a function that takes one input: a vector of the three numbers that you chose. The function should have one output: a character string indicating whether you won or lost the game. Write the function in the space below.

```
game <- function(numbers){
  card <- sample(1:10, size=1)


  if(card==numbers[1]|card==numbers[2]|card==numbers[3]){ #check if you won
  #another option: if(card %in% numbers)

    result <- "You won!"

  } else {

    result <- "You lost"

  }

  return(result)
}
```

2. Use a for loop to repeat the following process 100 times: randomly choose 3 numbers from 1-10 to play the game with (all values are equally likely) and record whether you won or lost the game in a vector of length 100. Print a table of your wins and losses.

```
game_results <- rep(NA,100)

for(i in 1:100){
  numbers <- sample(1:10, size=3, replace=FALSE)
  game_results[i] <- game(numbers)
}

table(game_results)
```

```
## game_results
## You lost You won!
##       70       30
```

3. Create a function that takes in a vector of data and returns (one of) the mode(s) of the data. (Hint: the supplementary R code for the math/stats review packet includes some code to find the mode of a vector). Challenge problem: write a function that returns a vector of all modes in a dataset (if there is more than one mode). Hint: the which() function might be of use.

```
mode <- function(vec) {
 result <- as.numeric(names(table(vec))[which.max(table(vec))])
 return(result)
}

#this one returns all modes:
#this is a good example of being careful not to use "off the shelf" solutions to problems:
#which.max(x) and which(x==max(x)) have different behavior.  Try them!
```

```r
all_modes <- function(vec) {
 result <- as.numeric(names(table(X1))[which(table(X1)==max(table(X1)))])
 return(result)
}
```

4. Use the mode() function that you just created to find the mode of each column in the simulated dataset (use a for loop to accomplish this). Instead of saving the results in a vector, print each of the results using the print() function (note: if you don't use print() within a loop or function, nothing will print out when you run the code or call the function).

```r
dat1 <- data.frame(X1 = c(1,1,2,3,4,rep(5,5),2,rep(7,7)),
                   X2 = c(3,9,18,27,27,20,40,1,1,1,3,4,1,rep(1,5)),
                   X3 = c(rep(4,4), rep(2,7), rep(10,7)))

for(i in 1:ncol(dat1)){
  print(mode(dat1[,i]))
}
```

```
## [1] 7
## [1] 1
## [1] 2
```