PS3 - STAT 243

Alexander Fred Ojala
Student ID: 26958060
afo@berkeley.edu

Collaborators: Guillame Baquiast and Milos Atz

September 30th 2015

1 Problem 1

1.1 1. a-d)

The question submitted to the Github repository was the following:

When writing modular code in R and splitting up different tasks into functions do you store the functions as stand-alone R scripts in your working directory (e.g. myFun1.R, myFun2.R etc.) and then source the function scripts into your main script? Or would you recommend writing several functions - specific for just that project - in one script (e.g. allFun.R) and then only source that one? Or do you usually define functions within the main script along with the other code? My guess is that a mix of these strategies should be used; if so what are your general guidelines for what situations / function types are most suitable for each strategy?

2 Problem 2

2.1 2. a)

First the necessary URLs to the debates of interest was extracted by parsing the html-info from the main html site with all links and then extracting the html node set. After that we could apply XML-commands in R associated with the XML package to extract title and urls of that website. The result was stored in a data frame and the years of interest was defined and sorted in the data frame with the function *url_title_extract* (see Github repository if the function should be examined).

```r
## Initial setup ----
URL<-"http://www.debates.org/index.php?page=debate-transcripts" #URL to look for speeches
doc<-htmlParse(URL)
listOfANodes <- getNodeSet(doc, "//a[@href]") #get all href a tags
links<-sapply(listOfANodes, xmlGetAttr, "href") #get links from href comments
titles<-sapply(listOfANodes, xmlValue) #get titles from xmlValue
rm(listOfANodes)

yrs<-c(1996, 2000, 2004, 2008, 2012)

urlTitleDF<-data.frame(matrix(NA,2,length(yrs))) #create data frame for URL and titles
row.names(urlTitleDF)<-c("url","title")

for (i in 1:length(yrs)) {
  urlTitleDF[i]<-url_title_extract(yrs[i],links,titles)
}

names(urlTitleDF)<-as.character(yrs)

urlTitleDF

##                                                                            1996
## url   http://www.debates.org/index.php?page=october-6-1996-debate-transcript
## title            October 6, 1996: The First Clinton-Dole Presidential Debate
##                                                                     2000
## url   http://www.debates.org/index.php?page=october-3-2000-transcript
## title        October 3, 2000: The First Gore-Bush Presidential Debate
##                                                                               2004
## url   http://www.debates.org/index.php?page=september-30-2004-debate-transcript
## title               September 30, 2004: The First Bush-Kerry Presidential Debate
##                                                                    2008
## url     http://www.debates.org/index.php?page=2008-debate-transcript
## title September 26, 2008: The First McCain-Obama Presidential Debate
##                                                                            2012
## url   http://www.debates.org/index.php?page=october-3-2012-debate-transcript
## title            October 3, 2012: The First Obama-Romney Presidential Debate
```

2.2 2. b)

To extract the body of any debate the function txt_extract was implemented, it can be seen below. The same procedure as in a) was used except here the node tag of interest was *p*. To handle special formating for the years two if statements took care of the differences (2012 was very different from the other years when it came html code e.g. no br:s for line breaks and in 2000 Lehrer was called Moderator in the transcript). In order to only extract the body of the text and no other content a pattern was observed. Every debate began with (LEHRER): and ended with *Thank you* in the last sentence combined with either *God Bless* or *APPLAUSE* (that we should keep in order to present later). Therefore a stanard regular expression could be used in order to extract the text body of all debates as one long vetor. We also have to be sure that we are using greedy matching (standard).

After that to print the text nicely (and have one speaker say one thing at a time). A pattern was observed that every speaker's chunk started with *(NAME):*. This was used with a string split where an odd character was first added before every speaker. After that the text was cleaned up from weird characters that was not latin-1 or UTF-8 (see the last lines below).

```
#Function to extract body of a txt
txt_extract = function(url,title,print=FALSE) {
  doc<-htmlParse(url)
  listOfANodes <- getNodeSet(doc, "//p")
  txt<-unlist(sapply(listOfANodes, xmlValue))
  rm(listOfANodes)
  if(!(length(grep("2012",url))==1)) {
    #All debates are stored the same except 2012 & to avoid copies of 2008 debate
    txt<-txt[2]
  }
```

```r
  txt<-paste(txt,collapse=" ")
  if(length(grep("2000",url)==1)) {
    txt<-gsub("MODERATOR:","LEHRER:",txt)
  }

  #same regex works for all debates, greedy matching
  regex=c("(LEHRER):.*(Thank)\\s(yo)[(A-Z)|\\s|,|(a-z)]+\\.(\\s(God)[(A-Z)|
          (a-z)|\\s]+\\.)?(\\(APPLAUSE\\))?")

  txt<-str_extract(txt,regex)

  #add odd character to every speaker, and then string split
  txt<-unlist(strsplit(gsub("([A-Z]+:)","~\\1",txt),"~"))

  txt<-txt[2:length(txt)] # remove first row, empty
  txt<-gsub("Ã¡$.","",txt) #remove weird characters (similar to Ã¡£. cannot be displayed with knitr)!
  txt<-gsub('\"',"'",txt) #change quotation standard in the text
  return(txt)
}
```

Below is an example of printing (with cat()) the first five speaker chunks in the debate of 2012.
N.B. Sorry for the formatting of the output, but it saves space on the paper and the environment will be happy! (knitr cannot handle automatic line breaks for cat).

```r
#Function to extract body of a txt
txt<-txt_extract(urlTitleDF$"2012"[1],urlTitleDF$"2012"[2])
title=urlTitleDF$"2012"[2]
cat(c(title,txt[1:5]),sep="\n\n")

## October 3, 2012: The First Obama-Romney Presidential Debate
##
## LEHRER: Good evening from the Magness Arena at the University of Denver in Denver, Colorado. I'm Jim
##
## LEHRER: This debate and the next three -- two presidential, one vice presidential -- are sponsored by
##
## LEHRER: You have two minutes. Each of you have two minutes to start. A coin toss has determined, Mr.
##
## OBAMA: Well, thank you very much, Jim, for this opportunity. I want to thank Governor Romney and the
##
## LEHRER: Governor Romney, two minutes.
```

2.3 2. c-f)

Now the fun starts with the analysis! The text was manipulated in order to exract statistics and data from the debates. First a data frame was created that is going to contain all the information we are looking for. All the rows, where every value is going to be presented was named and the number of columns was specified (which is going to represent every candidate + the moderator for every year of the debate). It is done with the R code below:

```r
#Function to extract body of a txt
dStat<-data.frame(matrix(NA,17,length(yrs)*3)) #values for each candidate for each debate
row.names(dStat)<-c("APPLAUSE","LAUGHTER","nWords","nChar","avgLength",
                    "I","we","America{,n}","democra{cy,tic}","republic",
                    "Democrat{,ic}","Republican","free{,dom}","war",
                    "God","God Bless","{Jesus, Christ, Christian}")

candidateYear<-rep(NA,length(yrs)*3) #pre-allocate array with year+speaker info
```

Then the processing of the data begins, where one debate/year is processed with a bunch of functions at a time.
To do the analysis we make use of three implemented functions not shown before, namely: *speaker_stat, speaker_chunks and count_stat*. The functions are presented below, and after those are presented the evaluation of the code together with the results are shown directly below.

First the statistics for LAUGHTER and APPLAUSE are extracted for every speaker, by first identifying speakers in the debate, then extracting the regex pattern and counting the occurences and then placing the results in a data frame.

```r
# Extract stat for each speaker with count of laughs and applause
speaker_stat = function (txt) {
  speakers=unlist(unique(str_extract_all(txt,"[A-Z]+:"))) #extract speakers

  nonSpoken=c("\\(APPLAUSE\\)|\\(Applause\\)","\\(LAUGHTER\\)")
  nonSpokenCount<-data.frame(matrix(0,length(nonSpoken),length(speakers)),stringsAsFactors=FALSE)

  names(nonSpokenCount)<-gsub(":","",speakers) #assign name to cols

  row.names(nonSpokenCount)<-c("APPLAUSE","LAUGHTER")

  for (i in 1:length(speakers)) {
    for (j in 1:length(nonSpoken)) {

      #extract data for APPLAUSE and LAUGHTER add up in the df
      nonSpokenCount[j,i]<-sum(str_count(grep(speakers[i],txt,value=TRUE),nonSpoken[j]))

    }
  }

  return(nonSpokenCount)
}
```

The function to extract the chunks of text (and to clean the text up) for every speaker, so that it is returned as a list with list entries for every candidate. And every list entry has arrays of chunks spoken by that candidate. It takes the text as one vector as input and then splits it up for every speaker, by adding a special character. Also weird and unrecognizable characters were removed (not latin-1 or other encodings) as well as (LAUGHTER), (APPLAUSE), (sic), (inaudible) and (ph)

```
# Extract chunks of text for every speaker
# Input txt: One long character vector of txt for each debate

speaker_chunks = function(txt) {

  # find all speakers in the debate
  speakers=unlist(unique(str_extract_all(txt,"[A-Z]+:")))

  #assign list entry for every speaker
  spokenChunks <- lapply(rep(NA,length(speakers)), function(i) i)

  names(spokenChunks)<-gsub(":","",speakers) #correct header

  for (i in 1:length(speakers)) {
    #remove (LAUGHTER), (APPLAUSE), (inaudible), (ph), (sic) etc
    txt2<-gsub("\\([A-Za-z]+\\)","",txt)

    # extract a chunk of text every time a speaker starts to say something
    spokenChunks[[i]]<-gsub(paste(speakers[i]," ",sep=""),"",grep(speakers[i],txt2,value=TRUE))
  }
  return(spokenChunks)
}
```

The last function to present for problem 2 is *count_stat*. The inputs are the dataframe to be filled with info, the correct indicies for the column (ci) for the statistics to be stored and the chunks of text in the list for the debate (chunkList). This is a pretty hard coded and not that elegant solution, but it gets the work done better than any other implementation that I tried (that was for more slicker, e.g. counting word occurences in only the word vector without specifying elimiter).

N.B. 1: The regular expressions might look like a mess, but after several iterations over all the debates of interest the regex:es below gave the best result on the full text (if the strings where extracted one could compare to the chunks). I could have also implemented a *paste* solution to combine strings for all the different types of delimiters of a certain word and the strings of interest (many times the patterns are reoccuring, especially for the phrases). However, this would only add on to the confusion as there were some special cases so even more references to different regex:es would be bad. Also see the long comments below for the regex:es to see the reasoning behind. It also important to note that the full text was used because R could not match the exact results in the *words* results for the debates. (I did not get the same result and the *carrot* and *dollarsign* operators did not work to specify just that exact word.) Also to extract the data from the long *chunkList* further improved my regex skills in order to find patterns (but it could definitely be more elegant coding if I got it to work with the words list).

N.B. 2: When counting the number of chars I chose to include e.g. ', *dollarsign* and *percent* (sorry that I have to write the characters like this, but my knitr file will not compile if I write *dollarsign* even when it is backslashed or double backslashed) as characters . I count them as characters in the words / abbreviations. That could make my results deviate somewhat from others.

```
#Count Statistics and Phrase Occurences

count_stat = function(df,ci,chunkList) {

# REGEX-SENTENCE: Matches all possible abbreviations in the beginning with
# [A-Z0-9]([a-z])?([0.9]+)?(\\.)+([A-Za-z])?([0-9]+)?(\\.)?
# OR
# Al word/digit character with dots combined with:
# [[[:alnum:]]](((\\.)+([[:alnum:]])+))?
# OR
# All delimiters a sentence can have w/o stopping \\s|[\\,\\'-]|;|&|\\£|\\%
# And end with a plus to select as many of them as possible
# Last sentences end in one of the following characters ([\\.?!\\:]|--) .
#-- is there because 1 sentence ended like that otherwise - is included in OR above

  regexSentence<-c("([A-Z0-9]([a-z])?([0.9]+)?(\\.)+([A-Za-z])?
                   ([0-9]+)?(\\.)?|[[[:alnum:]]](((\\.)+([[:alnum:]])+))?|
                   \\s|[\\,\\'-]|;|&|\\$|\\%)+([\\.?!\\:]|--)")


  # REGEX-WORDS: Matches all digits and characters from the sentence vector
  # with or without sepcial characters combining words
  # and also handles abbreviatons.
  regexWords<-"([[:alnum:]]+([\\'\\-\\$\\%]+)?((\\.)?[[:alnum:]]+)?)"

  sentence<-str_extract_all(chunkList,regexSentence)
  words<-str_extract_all(sentence,regexWords)

  df[3,cI]<-nWords<-unlist(lapply(words,length)) #count words
  df[4,cI]<-nChar<-unlist(lapply(lapply(words,nchar),sum)) #count characters
  df[5,cI]<-as.character(round(nChar/nWords,digits=3)) #average character length


  # Phrases and specific words
  # All sentences begin with capital letter, otherwise it should be
  # a space and a lower case character
  # all specific delimiters that can be after a word ends it

  #Extract I
  df[6,ci]<-str_count(chunkList,"((\\s)?I(\\s|\\.|\\?|!|,|:|;|'))")
  #Extract we
  df[7,ci]<-str_count(chunkList,"((We|\\s(we))(\\s|\\.|\\?|!|,|:|;|'))")
  #Extract America/American ... etc
  df[8,ci]<-str_count(chunkList,"(America)(\\s|\\.|\\?|!|,|:|;|')|
                      (American)(\\s|\\.|\\?|!|,|:|;|')")
  df[9,ci]<-str_count(chunkList,"(Democracy|\\s(democracy))(\\s|\\.|\\?|!|,|:|;|')
                      |(Democratic|\\s(democratic))(\\s|\\.|\\?|!|,|:|;|')")
  df[10,ci]<-str_count(chunkList,"(Republic|\\s(republic))(\\s|\\.|\\?|!|,|:|;|')")
  df[11,ci]<-str_count(chunkList,"(Democrat)(\\s|\\.|\\?|!|,|:|;|')
                       |(Democratic)(\\s|\\.|\\?|!|,|:|;|')")
  df[12,ci]<-str_count(chunkList,"(Republican)(\\s|\\.|\\?|!|,|:|;|')")
  df[13,ci]<-str_count(chunkList,"(Free|\\s(free))(\\s|\\.|\\?|!|,|:|;|')
                       |(Freedom|\\s(freedom))(\\s|\\.|\\?|!|,|:|;|')")
  df[14,ci]<-str_count(chunkList,"(War|\\s(war))(\\s|\\.|\\?|!|,|:|;|')")
  #God, but not bless
  df[15,ci]<-str_count(chunkList,"(God)(\\s|\\.|\\?|!|,|:|;|')[^([Bb]less)]")
  df[16,ci]<-str_count(chunkList,"((God)\\s([Bb]less))(\\s|\\.|\\?|!|,|:|;|')")
  df[17,ci]<-str_count(chunkList,"(Jesus|Christ|Christian)(\\s|\\.|\\?|!|,|:|;|')")
  return(df)
}
```

Lastly, we put made the analysis for every year of interest. Extract the information and it in the predefined data frame: *dStat*, as defined above. The result of the analysis is presented below and after that a brief comment about the results are also given.

```
for (i in 1:length(urlTitleDF)) {
  cI<-(1:3)+3*(i-1) #index of every speaker chunk in the df

  txt<-txt_extract(urlTitleDF[[i]][1],urlTitleDF[[i]][2]) # extract text

  dStat[1:2,cI]<-speaker_stat(txt) #extract LAUGHTER and APPLAUSE data

  chunkList<-speaker_chunks(txt) #cleans up the text

  #name the cols of the data frame correctly with name + year
  candidateYear[cI]<-paste(names(chunkList),yrs[i],sep=" ")

  #extract statistics about words, chars and phrases
  dStat<-count_stat(dStat,cI,chunkList)

}

names(dStat)<-candidateYear

#Only show candidate data, remove moderator in cols (1,4,7,10,13)
dStatCandidates<-dStat[c(2,3,5,6,8,9,11,12,14,15)]

dStatCandidates

##                              CLINTON 1996 DOLE 1996 GORE 2000 BUSH 2000
## APPLAUSE                                0         0         0         0
## LAUGHTER                                0         0         0         0
## nWords                               7630      8048      7207      7451
## nChar                               33671     34994     31482     32240
## avgLength                           4.413     4.348     4.368     4.327
## I                                     243       275       229       213
## we                                    159       166        96       109
## America{,n}                            34        42        13        19
## democra{cy,tic}                         2         5         1         0
## republic                                0         0         0         0
## Democrat{,ic}                           1         5         1         0
## Republican                              7        11         1         1
## free{,dom}                              3         1         0         0
## war                                     7         7         7         4
## God                                     0         0         0         0
## God Bless                               0         1         0         0
## {Jesus, Christ, Christian}              0         0         0         0
##                              KERRY 2004 BUSH 2004 OBAMA 2008 MCCAIN 2008
## APPLAUSE                              0         0          0           0
## LAUGHTER                              2         1          0           1
## nWords                              6885      6173       7441        6957
## nChar                              30420     27541      33345       31495
## avgLength                          4.418     4.462      4.481       4.527
## I                                    197       179        145         213
## we                                   150       170        267         167
## America{,n}                           42        24         13          18
## democra{cy,tic}                        0         0          0           0
## republic                               0         0          0           0
## Democrat{,ic}                          0         0          0           0
## Republican                             1         0          2           2
## free{,dom}                             2         9          0           3
## war                                   37        24         13           9
## God                                    0         1          0           0
## God Bless                              1         0          0           0
## {Jesus, Christ, Christian}             0         0          0           0
##                            OBAMA 2012 ROMNEY 2012
## APPLAUSE                               0           0
## LAUGHTER                               3           1
## nWords                              7278        7785
## nChar                              32531       33850
## avgLength                           4.47       4.348
## I                                    119         217
## we                                   182         124
## America{,n}                           18          34
## democra{cy,tic}                        3           1
## republic                               0           0
## Democrat{,ic}                          3           1
## Republican                             5           5
## free{,dom}                             2           1
## war                                    3           0
## God                                    0           0
## God Bless                              0           0
## {Jesus, Christ, Christian}             0           0
```

When it comes to analysis and comments on the result we see a clear peak in the use of war and freedom in the presidential debate in 2004 was much more frequent than in any other year. This is probably because it is in the aftermath of 9/11 (and Bush might have won since he uses freedom a lot more than Kerry).

When it comes to average word length McCain is at top and Bush 2000 at the bottom (even though Bush increased his reportaire of longer words in 2004).

Also it was surprising that candidates did not refer to Jesus, Christ or Christian. And that there was only one occurence of God bless. From my viewpoint and prejudices (European, Scaninvian, Swedish), I truly thought that the presidential debates would be filled with references to Christianity.

3 Problem 3

3.1 3 a) - b)

The function *r_walk* was implemented for the random walk. It takes the number of steps, the kind of random walk (full or only final position as a boolean) and also I added on the start positon of the random walk with the origin as default. An example of the output can be seen below for its two different inputs. The function checks so that the input values are correct from the user and the logic of the defensive programming should be self-explanatory from the stop-messages. Also the function is optimized in a vectorized way. I also added a starting position alternative to the function, that also is checked with if statements (defensive programming) that it was a vector only containing two integer values.

```r
r_walk = function(steps,full=TRUE,start=c(0,0)) {

  #checks
  if(!is.numeric(steps)) stop("The Number of steps must be numeric")
  if(!is.logical(full)) stop("Must be logical. Either TRUE for full walk or
                             FALSE for final position")
  if(!isTRUE(steps == floor(steps) & steps>=1)) stop("steps must be a positive integer")

  if(!is.double(start)) stop("start must be a vector with two integers")
  if(!isTRUE(length(start)==2)) stop("start must be a vector with two integers")
  if(!isTRUE(start[2] == floor(start[2])) & !isTRUE(start[2] == floor(start[2]))) {
    stop("start must be a vector with two integer values")
  }

  walk<-matrix(start,2,steps-1)

  #draw probabilities and assign each step with prob 0.25
    prob<-runif(steps-1)

  walk[,which(prob<=0.25)]=walk[,which(prob<=0.25)]+c(1,0)
  walk[,which(0.25<prob & prob<=0.5)]=walk[,which(0.25<prob & prob<=0.5)]+c(-1,0)
  walk[,which(0.5<prob & prob<=0.75)]=walk[,which(0.5<prob & prob<=0.75)]+c(0,1)
  walk[,which(0.75<prob & prob<=1)]=  walk[,which(0.75<prob & prob<=1)]+c(0,-1)

  realWalk<-matrix(c(c(start[1],cumsum(walk[1,])),c(start[2],cumsum(walk[2,]))),2,
                   steps,byrow = TRUE)


  if(full) {
    return(realWalk)
  }
  else {
    return(realWalk[,steps])
  }

}
```

The output that this function produces can be seen below:

```r
r_walk(26)

##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## [1,]    0    0   -1   -2   -2   -2   -1   -1   -1    -1    -1     0     1
## [2,]    0   -1   -1   -1    0   -1   -1   -2   -3    -2    -1    -1    -1
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24]
## [1,]     2     2     1     1     0     0     0    -1    -1    -1     0
## [2,]    -1     0     0    -1    -1     0    -1    -1    -2    -3    -3
##      [,25] [,26]
## [1,]     0     1
## [2,]    -2    -2
```

```r
r_walk(2000,full=FALSE)

## [1] -10 -61
```

3.2 3 c)

The function was embedded in S3 object-oriented method so that we created a class and methods and attributes for the objects of the class. The constructor makes use of the random walk function's functionality, but adds on methods to work on that class of objects.
Below you can see the implementation where the class *rw* is implemented along with the function, so to create a random walk we specify the number of steps (and if we want origin, that can also be changed later with the replacement method). It resturns a list and assigns it the rw-class. Where the user has the following attributes for the object: steps (number of steps), walk (the random walk as a row matrix), start (start position), final (final position).

A print method was implemented that sums up the different, relevant aspects of the walk, namely steps, start and final in a nice printout with cat that explains the result to the user.

Also a *bracket* operator was assigned to show the i:th step of the random walk, here defensive programming was implemented so that the user can only see steps that are relevant for this random walk.

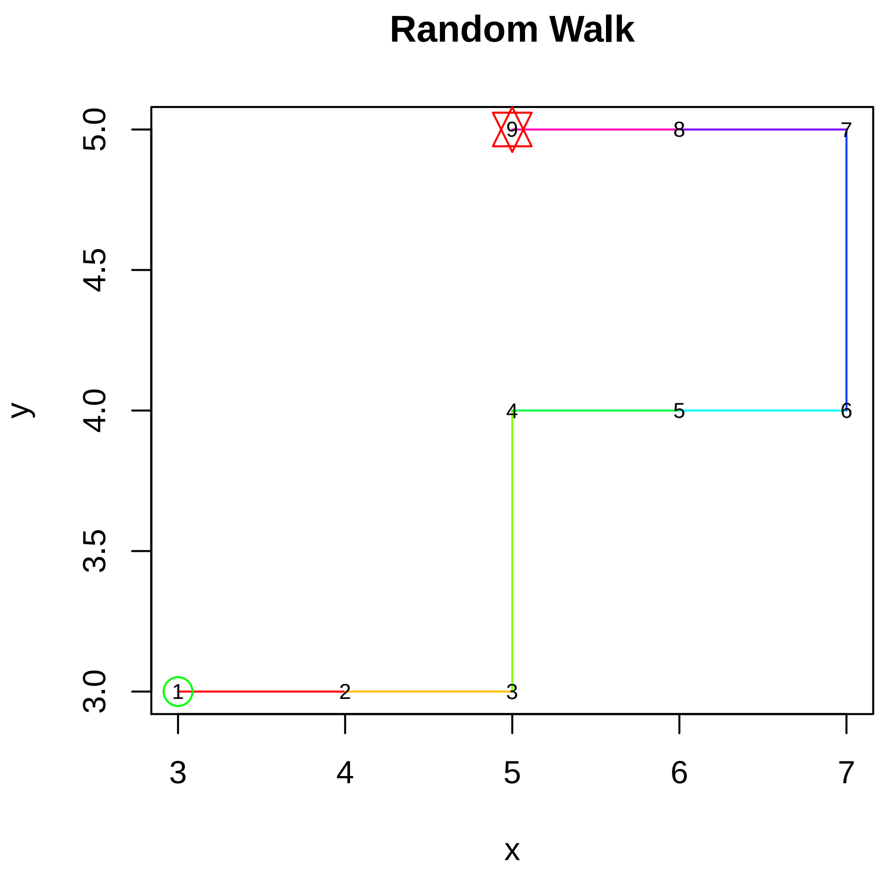
The replacement method for the start position first checks the old start position and then assigns the new one to the full walk, the start and the final. Defensive programming is also implemented here.

The plotting method was defined to for every step draw a line from two positions that is of a different color from the last one. Also the number of the step is printed to the coordinates of the step. As well as that the beginning of the random walk as well as the end are marked with special characters.

```r
# CONSTRUCTOR

rw = function(steps,start=c(0,0)) {

  realWalk<-r_walk(steps,start=start) # Create the ranom walk
  #r_walk will check inputs

  value<-list(steps=steps, walk=realWalk, start=start,final=realWalk[,steps])

  class(value)<-"rw"
  value # print info
}


# PRINT METHOD

print.rw <- function(rw) {
  cat("Random Walk of ", rw$steps, " of steps\n")
  cat("Start position (x y)_start = ", rw$start[1],rw$start[2], "\n")
  cat("Final position (x y)_final = ", rw$final[1],rw$final[2],"\n")
}

# `[` operator see i:th step of rw

`[.rw` <- function (rw,i)  {
  if(!is.numeric(i)) stop("i must be a positive integer")
  if(!isTRUE(i == floor(i) & i>=1 & i<=rw$steps)) stop("i must be a positive integer
                                                    between 1 and rw steps")
  y<-rw$walk[,i]   # find position i in the random walk
  return (y)
}


# REPLACEMENT METHOD replace/change start position

`start<-` <- function(x, ...) UseMethod("start<-")
`start<-.rw` <- function(rw,value){
  #checks
  if(!is.double(value)) stop("start must be a vector with two integers")
  if(!isTRUE(length(value)==2)) stop("start must be a vector with two integers")
  if(!isTRUE(value[2] == floor(value[2])) & !isTRUE(value[2] == floor(value[2]))) {
      stop("start must be a vector with two integer values")
  }

  old <- rw$start   # old starting position
  rw$start <- value # set new start position
  rw$walk <- rw$walk + rw$start - old # add to the walk
  rw$final <- rw$final + rw$start - old # add to info about final
  return(rw)
}


# PLOTTING METHOD

plot.rw <- function(rw) {
  walk<-rw$walk #define the walk
  plot(0,type="n",xlab="x",ylab="y",main="Random Walk",col=1:10,
       xlim=range(walk[1,]),ylim=range(walk[2,])) # initialize plot

  #Add one segment (a line from 1 coordinate to the next) for every other element
  segments(head(walk[1,], -1)
           , head(walk[2,], -1)
           , tail(walk[1,], -1)
           , tail(walk[2,], -1)
           , col = rainbow(ncol(walk) -1)  # color differently for every step
  )
  text(walk[1,1:rw$steps],walk[2,1:rw$steps],(1:rw$steps), cex=0.55) # print numbers on steps

  points(walk[1,1],walk[2,1],pch=1,col="green", cex = 2) # mark beginning with green circle
  points(walk[1,rw$steps],walk[2,rw$steps],pch=11,col="red", cex = 2) # mark end with red star
}
```

If we construct a random walk and run all the different methods and show all the outputs that we have defined for this class of objects, it could be summarized with the evaluation below:

```
rwTest<-rw(9)
rwTest

## Random Walk of  9  of steps
## Start position (x y)_start =  0 0
## Final position (x y)_final =  2 2

names(rwTest)

## [1] "steps" "walk"  "start" "final"

rwTest$walk

##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
## [1,]    0    1    2    2    3    4    4    3    2
## [2,]    0    0    0    1    1    1    2    2    2

rwTest[6]

## [1] 4 1

start(rwTest)<-c(3,3)
rwTest$walk

##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
## [1,]    3    4    5    5    6    7    7    6    5
## [2,]    3    3    3    4    4    4    5    5    5

plot(rwTest)
```


Random Walk
y
x
3.0
3.5
4.0
4.5
5.0
3
4
5
6
7
1
2
3
4
5
6
7
8
9