Problem Set 3

Alexander Brandt
SID: 24092167

September 30 2015

1 Debugging Reading

I chose to read option ii) as I am a huge fan of the Software Carpentry Foundation, as well as Titus Brown's work in general.

My question/comment was "After reading the Software Carpentry Foundation's paper on 'Best Practices for Scientific Computing', I found myself wondering about section 9, which says 'Document design and purpose, not mechanics.' When is code written in such a way that it is self-evident to an outside reader/developer? What are the hallmarks of code written in this style? I think the suggestion has good intentions, but often discerning the difficulty level associated block of code is challenging for developers. One person's 'easy' is often another person's 'hard,' and so in this way applying the principle of "the onerous should always be on the author to convince his or her peers of that" seems to be to be a recipe for disaster in the hands of some very well-meaning scientific programmers that I know that have forgotten how difficult it can be to work within a certain language, leading to less commentary and comprehension, not more. For example, would explaining a choice of data structure in a language (a dict vs. a list in Python, for example) be violating this principle?"

2 An Analysis of the Presidental Election Debates

For the debate analysis, I attempted to create a modular, function solution to each subproblem and then use various apply or lapply methods. To begin, we construct a series of URL's by pulling from the appropriate tags in the main debate transcript webpage. From there, we select the correct years and look for the "first" debate with a regular expression. We now apply the debate_summary method to each link, and print that information to the screen with some information about the year the debate took place.

The debate_summary function uses three different helper methods. The first,

create _debate_text, pulls the information from each page out with calls to the correct HTML tag's values. These are split into parts using a grep that finds the speakers's name (in the format SPEAKER) and then maintains the structure while adding a newline. This is returned to split_block which loops through these lines of text and adds them to a list, indexed by the speaker's name.

Finally, the summary_stats function builds a list, indexed by the various values asked for in the problem set, which allows us to quickly/cleanly navigate the pertinent information.

```
library(stringr)
library(XML)
library(curl)
library(RCurl)

## Loading required package:  bitops

# Create debate takes a url and extracts all of the
# text from between the dev[id = 'content-sm'] tags
# and then splits the block of text on the speaker
# information (here given as "SPEAKER: ").  Note it
# preserves the speaker's name using the \\1 value.
create_debate_text <- function(file_url)
{
  # Get rid of br tags for all non-2012 debates
  temp <- gsub("<br/>","\n",
               paste(readLines(curl(file_url),
               warn = FALSE),collapse=""))
  # Get ride of p tags for the 2012 debate
  temp <- gsub("<p>","\n",temp)
  temp <- gsub("<*p>","\n",temp)

  xml_handle <- htmlParse(temp, asText=TRUE)
  v <- xpathSApply(xml_handle,
                   "//div[@id = 'content-sm']",xmlValue)
  text_data <- lapply(v,str_replace_all,
                      "([A-Z]+:)","\n\n\\1")
  return(unlist(text_data))
}

# Split block takes a large block of text as a string
# (with "\n" still preserved).  It then skips any
# pre-speaking information, like the location, etc.
# and it truncates the debate at the END (or until the
# text runs out).
split_block <- function (list_of_strings_solid) {
```

```
current_name = ""
current_block = ""
my_list = list()
# Breaks on newlines, preserves general format of the
# webpage
list_of_strings <- unlist(strsplit(list_of_strings_solid,"\n"))
# XML example class notes -- different HTML features
for (i in 1:length(list_of_strings)) {
  if (((toupper(list_of_strings[[i]]) == list_of_strings[[i]]) &&
       !grepl("^\\([A-Z]+\\)$",list_of_strings[[i]]))) {
    # This is a "caps line" which doesn't contain useful
    # information about the text of the debate.
    # Mostly filler.  Move to next UNLESS it is
    # of the form "(EVENT)" (i.e. "(APPLAUSE)")
    next
  }
  # Helps prevent bad formatting on the part of the website
  if (grepl("END",list_of_strings[[i]])
      || grepl("Transcription",list_of_strings[[i]])) { break }
  # Uses the name of the speaker as the index
  name <- str_match(list_of_strings[[i]],regex("([A-Z]+):"))[,2]
  if ((!is.na(name)) && name != current_name) {
    if (current_name != "") {
      my_list[[current_name]] <- c(my_list[[current_name]],
      str_replace_all(current_block,paste(current_name,": ",sep=""),
      ""))
    }
    # Resets the name, resets the block
    current_name <- name
    current_block <- ""
    # ... and maybe add a new block to the list?
  }
  # If there is a name, append the speaking block to the list
  if (length(current_name) != 0) {
    if (current_block != "") {
      current_block <- paste(current_block,
                             list_of_strings[[i]], sep=" ")
    }
    else {
      current_block <- list_of_strings[[i]]
    }
  }
}
# Gets any "hanging" information within the block buffer
my_list[[current_name]] <- c(my_list[[current_name]],
```

```
  str_replace_all(current_block,
  paste(current_name,": ",sep=""),""))
  return(my_list)
}

# Constructs summary information about the debate based
# on a debate block
summary_stats <- function(debate_block)
{
   return_vals = list()
   # QUESTION 2C) gets "events" like (APPLAUSE) as they
   # occur, and saves the freqency of the event before
   # sanitizing the text
   return_vals$events <- table(
    str_extract_all(
      paste(debate_block,collapse=" "),
      "\\([A-Za-z]+\\)"))
   debate_block <- lapply(debate_block,
                               str_replace_all,
                               "\\([A-Za-z]+\\)","")

   # The various REGEX patterns we are interested in.
   # Note that I take capital letters to generally be an
   # unambiguous start to the word (vs. a word like "we"
   # which could show up as a suffix).  I assuemd that
   # "we" and "war" and "freedom" should be case insensitive
     patterns = c("I[^a-zA-Z]",
                  "[^a-zA-Z](W|w)e[^a-zA-Z]",
                  "America(n)?[^a-zA-Z]",
                  "[^a-zA-Z]democra(cy|tic)[^a-zA-Z]",
                  "[^a-zA-Z]republic[^a-zA-Z]",
                  "Democrat(ic)?[^a-zA-Z]",
                  "Republican[^a-zA-Z]",
                  "[^a-zA-Z](F|f)ree(dom)?[^a-zA-Z]",
                  "[^a-zA-Z](W|w)ar[^a-zA-Z]",
                  "God(?! bless)[^a-zA-Z]",
                  "(Jesus|Christ|Christian)[^a-zA-Z]",
                  "God bless[^a-zA-Z]")

  to_analyze <- paste(debate_block,collapse=" ")
  # Count the number of each buzzword
  word_counts <- lapply(patterns, function(me) str_count(to_analyze,me))
  # Present more cleanly as a dataframe with nice names
  word_counts <- as.data.frame(word_counts)
  names(word_counts) <- c("I","we","America{n}",
```

```
                                "democra{cy,tic}","republic",
                                "Democrat{,ic}","Republican",
                                "free{,dom}","war",
                                "God (only)","God Bless",
                                "{Jesus, Christ, Christian}")
  # QUESTION 2D) and 2E)
  # A regular expression to get words and sentences.  I treated
  # written numbers ("400,000") as one word, when possible.
  words <- str_extract_all(to_analyze,
  '(([:alpha:]+(\'([:alpha:]+)?)?)|([:digit:]+(,([:digit:]+)?)?))')
  sentence <- str_extract_all(to_analyze,
  "([:alpha:])(|[:alpha:]|[:space:]|[:digit:]|'|,|-|\\$|/|\\\"|(\\.[A-Za-z]))*((\\.\\.\\.)|\
  #return_vals$my_words <- words
  #return_vals$my_sentences <- sentence
  return_vals$mean_word_length <- mean(rapply(words,nchar))
  return_vals$num_character_from_words <- sum(rapply(words,nchar))
  return_vals$number_of_words <- length(unlist(words))
  #return_vals$buzzwords <- word_counts
  #return_vals$db <- debate_block
  # Return all of the pertinent information for later printing
  return(return_vals)
}

debate_summary <- function(file_url)
{
  # QUESTION 2B) SOLUTION
  text_data <- create_debate_text(file_url)
  debate_blocks <- split_block(text_data)
  debate_statistics <- lapply(debate_blocks,summary_stats)

  return(debate_statistics)
}
# QUESTION 2A) solution
# Grabs the html links for the debates using the a href= tag,
# and then subsets based on the descriptor for the correct year
# and the first debate.
menu_url="http://www.debates.org/index.php?page=debate-transcripts"
menu_xml_handle <- htmlParse(menu_url)
menu_nodes <- getNodeSet(menu_xml_handle,"//a[@href]")
all_debate_links <- xpathSApply(menu_xml_handle,
                                "//a[@href]", xmlGetAttr, 'href')
years <- c("2012","2008","2004","2000","1996")
year_reg <- paste("(",paste(paste(years,collapse="|"),").+(First)"
                            ,sep="")
                  ,sep="")
```

```r
# Uses a logical grep to subset all of the debate links into just
# the ones we want
my_debate_links <- all_debate_links[grepl(
  year_reg,
  sapply(menu_nodes,xmlValue))]
debate_blocks_list = list()

# Using a simple for loop to print with a nice index for the years.
# We do it with an lapply but it is nice to have the debate year
# printed out with the summary stats.  I thought that would be fine
# for output.

debate_blocks_statistics <- lapply(my_debate_links,debate_summary)
```

```
## Warning:  closing unused connection 5 (http://www.debates.org/index.php?page=october-3-20
## Warning:  closing unused connection 5 (http://www.debates.org/index.php?page=2008-debate-
## Warning:  closing unused connection 5 (http://www.debates.org/index.php?page=september-30
## Warning:  closing unused connection 5 (http://www.debates.org/index.php?page=october-3-20
## Warning:  closing unused connection 5 (http://www.debates.org/index.php?page=october-6-19
```

```r
names(debate_blocks_statistics) <- years
print(debate_blocks_statistics)
```

```
## $`2012`
## $`2012`$LEHRER
## $`2012`$LEHRER$events
##
##  (APPLAUSE) (CROSSTALK) (inaudible)
##           1          10           4
##
## $`2012`$LEHRER$mean_word_length
## [1] 4.399606
##
## $`2012`$LEHRER$num_character_from_words
## [1] 6705
##
## $`2012`$LEHRER$number_of_words
## [1] 1524
##
##
## $`2012`$OBAMA
## $`2012`$OBAMA$events
##
## (CROSSTALK)  (LAUGHTER)
##           4           3
##
```

```
## $`2012`$OBAMA$mean_word_length
## [1] 4.450814
##
## $`2012`$OBAMA$num_character_from_words
## [1] 32531
##
## $`2012`$OBAMA$number_of_words
## [1] 7309
##
##
## $`2012`$ROMNEY
## $`2012`$ROMNEY$events
##
## (CROSSTALK) (inaudible)  (LAUGHTER)
##          11           2           1
##
## $`2012`$ROMNEY$mean_word_length
## [1] 4.322593
##
## $`2012`$ROMNEY$num_character_from_words
## [1] 33807
##
## $`2012`$ROMNEY$number_of_words
## [1] 7821
##
##
##
## $`2008`
## $`2008`$LEHRER
## $`2008`$LEHRER$events
##
##  (APPLAUSE) (CROSSTALK)  (LAUGHTER)
##           2           4           1
##
## $`2008`$LEHRER$mean_word_length
## [1] 4.326471
##
## $`2008`$LEHRER$num_character_from_words
## [1] 5884
##
## $`2008`$LEHRER$number_of_words
## [1] 1360
##
##
## $`2008`$OBAMA
```

```
## $`2008`$OBAMA$events
##
## (CROSSTALK)        (ph)
##           3           1
##
## $`2008`$OBAMA$mean_word_length
## [1] 4.368359
##
## $`2008`$OBAMA$num_character_from_words
## [1] 33383
##
## $`2008`$OBAMA$number_of_words
## [1] 7642
##
##
## $`2008`$MCCAIN
## $`2008`$MCCAIN$events
##
## (CROSSTALK)  (LAUGHTER)        (ph)       (sic)
##           1           1           1           1
##
## $`2008`$MCCAIN$mean_word_length
## [1] 4.412685
##
## $`2008`$MCCAIN$num_character_from_words
## [1] 31586
##
## $`2008`$MCCAIN$number_of_words
## [1] 7158
##
##
##
## $`2004`
## $`2004`$LEHRER
## $`2004`$LEHRER$events
##
## (APPLAUSE)
##          2
##
## $`2004`$LEHRER$mean_word_length
## [1] 4.715942
##
## $`2004`$LEHRER$num_character_from_words
## [1] 6508
##
```

```
## $`2004`$LEHRER$number_of_words
## [1] 1380
##
##
## $`2004`$KERRY
## $`2004`$KERRY$events
##
## (LAUGHTER)
##          2
##
## $`2004`$KERRY$mean_word_length
## [1] 4.291059
##
## $`2004`$KERRY$num_character_from_words
## [1] 30621
##
## $`2004`$KERRY$number_of_words
## [1] 7136
##
##
## $`2004`$BUSH
## $`2004`$BUSH$events
##
## (LAUGHTER)
##          1
##
## $`2004`$BUSH$mean_word_length
## [1] 4.319169
##
## $`2004`$BUSH$num_character_from_words
## [1] 27444
##
## $`2004`$BUSH$number_of_words
## [1] 6354
##
##
##
## $`2000`
## $`2000`$MODERATOR
## $`2000`$MODERATOR$events
##
## (Applause) (APPLAUSE)
##          1          1
##
## $`2000`$MODERATOR$mean_word_length
```

```
## [1] 4.558824
##
## $`2000`$MODERATOR$num_character_from_words
## [1] 7750
##
## $`2000`$MODERATOR$number_of_words
## [1] 1700
##
##
## $`2000`$GORE
## $`2000`$GORE$events
## < table of extent 0 >
##
## $`2000`$GORE$mean_word_length
## [1] 4.339315
##
## $`2000`$GORE$num_character_from_words
## [1] 31434
##
## $`2000`$GORE$number_of_words
## [1] 7244
##
##
## $`2000`$BUSH
## $`2000`$BUSH$events
## < table of extent 0 >
##
## $`2000`$BUSH$mean_word_length
## [1] 4.3035
##
## $`2000`$BUSH$num_character_from_words
## [1] 32216
##
## $`2000`$BUSH$number_of_words
## [1] 7486
##
##
##
## $`1996`
## $`1996`$LEHRER
## $`1996`$LEHRER$events
## < table of extent 0 >
##
## $`1996`$LEHRER$mean_word_length
## [1] 4.546569
```

```
##
## $`1996`$LEHRER$num_character_from_words
## [1] 5565
##
## $`1996`$LEHRER$number_of_words
## [1] 1224
##
##
## $`1996`$CLINTON
## $`1996`$CLINTON$events
## < table of extent 0 >
##
## $`1996`$CLINTON$mean_word_length
## [1] 4.377442
##
## $`1996`$CLINTON$num_character_from_words
## [1] 32485
##
## $`1996`$CLINTON$number_of_words
## [1] 7421
##
##
## $`1996`$DOLE
## $`1996`$DOLE$events
##
## (ph)
##    3
##
## $`1996`$DOLE$mean_word_length
## [1] 4.308138
##
## $`1996`$DOLE$num_character_from_words
## [1] 34939
##
## $`1996`$DOLE$number_of_words
## [1] 8110
```

Comments for e/f:

Not surprisingly, useage of the word "war" which spikes around the 2004 election, and is still pretty high in 2008 too. This probably is probably correlated with the national awareness and interest in U.S. military intervention in the Middle East.

Ancedotally, from this small subset, number of words used doesn't seem to be a good indicator of who wins the election.

3 Random Walks and OOP

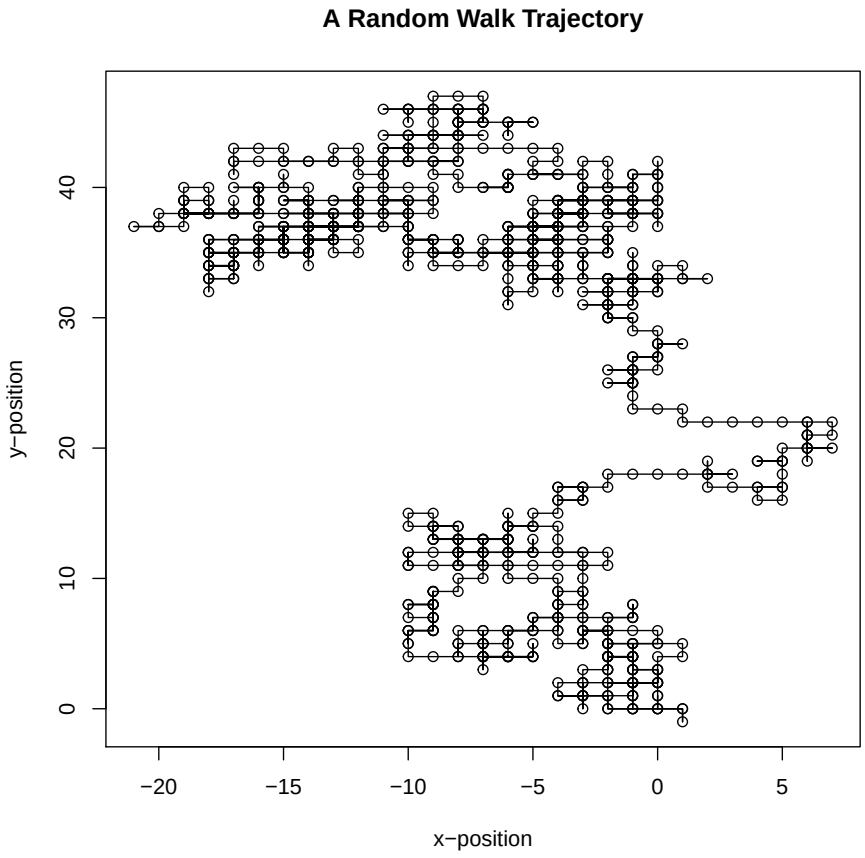
Here we build a new class "rw" which makes a call to simulate() to create a random walk trajectory. Various new methods/functions are texted in accordance with the problem set's requests and then evaluated at the end of the file.

```
# While I didn't work with any other students on this question, I found
# the S4 classes a little complicated, and consulted print/web sources
# pretty extensively.  As always, I attempted to do so in a way that
# forced my own understanding (no copy + paste), but these were great
# resources that I didn't want to omit from my write-up.
#
# Citations:
# https://cran.r-project.org/doc/contrib/Genolini-S4tutorialV0-5en.pdf
# http://www.cyclismo.org/tutorial/R/s4Classes.html
# http://practicalcomputing.org/node/80
#
# We begin by defining rw, which holds our various slots
rw <- setClass(
  "rw",
  # The basic slots associated with our class
  slots = c(
    start = "numeric",
    steps   = "numeric",
    trajectory_recording = "logical",
    .trajectory = "matrix"),

  # Now we declare our default values
  prototype=list(
    start = c(0,0),
    #steps = 10,
    trajectory_recording = TRUE
  ),
  # Look for things that might be amiss
  validity=function(object)
  {
    # We need our steps to be greater than 0...
    if(object@steps<1) {
      return("Please enter
             a positive number of steps.")
    }
    # and integers...
    if(as.integer(object@steps)!=object@steps) {
      return("Please enter
             an integer valued number of steps.")
    }
```

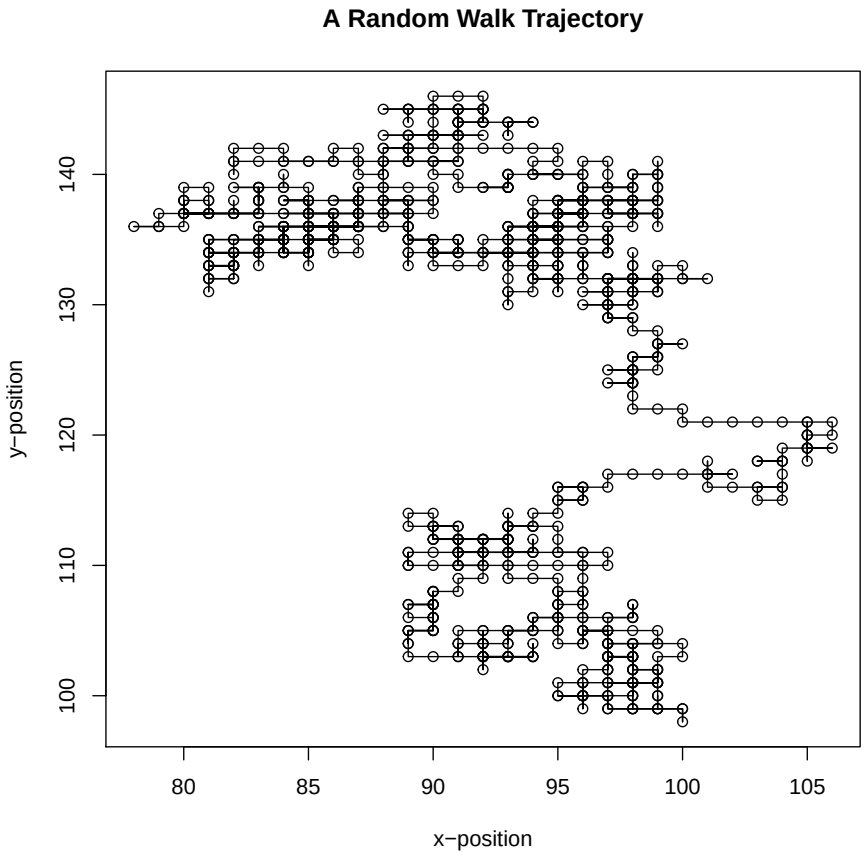
```
    # and we should only be working in 2D.
    if(length(object@start)!=2) {
      return("This program is
             only written for 2D (for now!).")
    }
    return(TRUE)
  }
)

# Define the method to replace our start position
setGeneric("start<-", function(self, value) standardGeneric("start<-"))

## [1] "start<-"

setReplaceMethod("start",
  "rw",
  function(self,value) {
  self@start <- value
  self
  }
)

## [1] "start<-"

# Define the rw[i th] <- replacement method
setMethod(
  f="[",
  signature="rw",
  definition=function(x,i,drop){
    mypath=slot(x,".trajectory");
    xs=sum(mypath[1:i,1]);
    ys=sum(mypath[1:i,2]);
    return(c(x@start[1]+xs, x@start[2]+ys))
  }
)

## [1] "["

# Define our plotting function
setMethod(
  f="plot",
  signature="rw",
  definition=function(x){
    mypath=slot(x,".trajectory");
    # here we use "cumsum" to get the cumulitive
    # displacement vector at each point in time
    # which we add to our start
```

```
    xs=cumsum(mypath[,1]);
    ys=cumsum(mypath[,2]);
    plot(x@start[2]+ys,x@start[1]+xs, type='o',
         xlab='x-position', ylab='y-position',
         main='A Random Walk Trajectory');
  }
)

## Creating a generic function for 'plot' from package 'graphics' in
the global environment

## [1] "plot"

# Some details about the walk.  We will print out
# the entire trajectory if the user has specificed
# that they would like to with the optional argument
setMethod(
  f="print",
  signature="rw",
  definition=function(x){
    print("Starting position:")
    print(slot(x,"start"))
    print("After this many steps...:")
    print(slot(x,"steps"))
    print("We arrive at:")
    print(x[slot(x,"steps")])
    if (slot(x,"trajectory_recording"))
    { print("List of positions visited in order:")
      for (i in 1:slot(x,"steps")) {
        print(x[i]) }}
  }
)

## Creating a generic function for 'print' from package 'base' in the
global environment

## [1] "print"

# The simulate method initializes the walk
setGeneric("simulate",
           function(x){standardGeneric("simulate")})

## Creating a new generic function for 'simulate' in the global environment

## [1] "simulate"
```

```r
setMethod(
  f="simulate",
  signature="rw",
  definition=function(x){
    # Set our random seed for reproduceable results.
    set.seed(0)
    slot(x,".trajectory") <-
      matrix(c(0, 1, -1, 0, 1, 0, 0, -1),
             nrow=4,
             ncol=2)[sample(4,size=slot(x,"steps"),
             replace=TRUE),];
    return(x)
  }
)

## [1] "simulate"

# Testing the replacement class
my_walk <- new("rw", start=c(1,1),steps=1000,trajectory_recording=FALSE)
# We initialize the walk with the simulate() method -- I don't love this.
# But to circumvent, I would need to use the "assign" function, which
# the S4 manual cautions against!
# (https://cran.r-project.org/doc/contrib/Genolini-S4tutorialV0-5en.pdf)
my_walk <- simulate(my_walk)
# Show the 50th position
my_walk[50]

## [1]  1 -3

# Plot before we move the start
plot(my_walk)
```




```
# Change the start position
start(my_walk)<-c(100,100)
# Plot AFTER we move the start
# ...Hopefully they are the same shape, just shifted!
plot(my_walk)
```




```
# print the summary statistics
print(my_walk)

## [1] "Starting position:"
## [1] 100 100
## [1] "After this many steps...:"
## [1] 1000
## [1] "We arrive at:"
## [1] 144  94
```