STAT243 Problem Set2

Name: Chih Hui Wang SID: 26955255

September 28, 2015

1. These comments and questions are from the article written by Milman and Perez.

Comments

The article breaks down the computational research cycle into difference steps which make me understand the problem it mentioned. For example, I learned Git last year; however, without participating in an open project, I haven't use the most powerful part of it. In the chapter3, the authors illustrate several ideas such as version control and testing to tell us how important those tools or practice and how they will influence the workflow of computational research. Not only can they make the big project run successfully, but also benefit some daily routines. The authors take the open source project, IPython, as an example to show how we can utilize tools and procedure to improve the computation scientific work. It makes their points more clear and convincing to reader.

Questions

(1). Will it be better if we have a united and compulsory for computational work to obey?

(2). Is there any suggested learning path for students to acquire skills for each tool?

(3). What may be the drawback for using the tools or practices mentioned in article?

2. (a) I use the function in package **XML** to get the first of three debates' links for 1996, 2000, 2004, 2008, 2012. I pull out the node with html tag <a herf ...> and get their name and attributes by **xmlValue** and **xmlGetAttr**. Then, I can use **grep** function to search the word "First" in name and each year in link to find out those links' positions and store them into variable **link_want**.

```
library(XML)

Warning:  package 'XML' was built under R version 3.2.2
Loading required package:  methods

#Get the html
html <- htmlParse("http://www.debates.org/index.php?page=debate-transcripts")

#Take out the node with link
node <- getNodeSet(html, "//a[@href]")

#Retrieve the Link
weblink <- sapply(node, xmlGetAttr, "href")
#Pull out name of the link
name <- sapply(node, xmlValue)

#All the link whose name contain First
first_debate_link <- weblink[grep("First", name)]
#Get the year we want
link_want <- first_debate_link[grep("1996|2000|2004|2008|2012", first_debate_link)]

link_want
```

```
[1] "http://www.debates.org/index.php?page=october-3-2012-debate-transcript"
[2] "http://www.debates.org/index.php?page=2008-debate-transcript"
[3] "http://www.debates.org/index.php?page=september-30-2004-debate-transcript"
[4] "http://www.debates.org/index.php?page=october-3-2000-transcript"
[5] "http://www.debates.org/index.php?page=october-6-1996-debate-transcript"
```

(b) To print the debate script, first we have to get the texts in each link and then output them by cat function. However, the text for 2012 is tagged by <p> while others are tagged by
. Therefore, I take different procedures for each situation. For 2012, the way to get the text is similar to the one in (a). For other years, I first use **readLines** to get all the html file and then **grep** by

 to get texts. For both method, I remove all the irrelevant word in the beginning and end.

```
#(b)
debate_printout <- function(year){
  #Get the link of the specific year
  link <- link_want[grep(year, link_want)]
  if(year == 2012){
    html_year <- htmlParse(link)
    #Get the spoken text
    script <- xpathSApply(html_year, "//p", xmlValue)
    #Remove the irrelevant words in the beginning and end
    script <- script[grep(":", script)[1]:(length(script) - 2)]
  }else{
    #Get the all html
    script <- readLines(link)

    #Get the spoken context
    script <- script[grep("<br/><br/>", script)]
    #Sepearate the text by <br/><br/> to get the sentences
    script <- unlist(strsplit(script, "<br/><br/>"))

    #Remove irrelevant html tag
    script <- script[-grep("</?p>|<br/>", script)]
    #Remove irrelevant words in the beginning
    script <- script[grep(":", script)[1]:length(script)]
  }

  #Replace words like (Laughter), (crosstalk)...
  script <- gsub("\\([[:alpha:]]+\\)", "", script)

  #Remove the ""
  script <- script[script != ""]

  #Ouput the result
  cat(script, sep="\n")
}
```

(c) To make those texts in a row into same chunk, first I have to obtain the script. I do a little adjustment from the function in (a). To get a better script, I seperate the 2004(for the introduction of speakers) and 2008(for the repeated text) to do some processes. The output will give user the debate text from the website without any other unnecessary words in it.

```r
#(c)
debate_script <- function(year){
  #Get the link of the specific year
  link <- link_want[grep(year, link_want)]

  if(year == 2012){
    html_year <- htmlParse(link)
    #Get the spoken text
    script <- xpathSApply(html_year, "//p", xmlValue)

    #Remove the irrelevant words in the beginning and end
    script <- script[grep(":", script)[2]:(length(script) - 2)]
  }else{
    #Get the all html
    script <- readLines(link)

    #Get the spoken context
    script <- script[grep("<br/><br/>", script)]
    #Sepearate the text by <br/><br/> to get the sentences
    script <- unlist(strsplit(script, "<br/><br/>"))

    #Remove irrelevant html tag
    script <- script[-grep("</?p>", script)]
    #Remove irrelevant words in the beginning
    script <- script[grep(":", script)[1]:length(script)]

    if(year == 2004){
      script <- script[grep(":", script)[2]:length(script)]
    }else if(year == 2008){
    script <- script[grep(":", script)[2]:(grep("END", script)[1] - 1)]
    }
  }

  #Remove the transcription
  script[grep("Transcription", script)] <- ""

  #Remove the ""
  script <- script[script != ""]

  #result
  script
}
#Script for each year
script_2012 <- debate_script(2012)
script_2008 <- debate_script(2008)
```

```
Warning in readLines(link):  incomplete final line found on 'http://www.debates.org/index.php?page=20
```

```r
script_2004 <- debate_script(2004)
```

```
Warning in readLines(link):  incomplete final line found on 'http://www.debates.org/index.php?page=se
```

```r
script_2000 <- debate_script(2000)
```

```
Warning in readLines(link):  incomplete final line found on 'http://www.debates.org/index.php?page=oc
```

```
script_1996 <- debate_script(1996)
```

```
Warning in readLines(link):  incomplete final line found on 'http://www.debates.org/index.php?page=oc
```

With the script, I can then do some processes to make those text spoken in a row and by the same people into one chunk. My strategy is that I find out the speakers of debates, make a vector with same length as texts and then use them as a reference to decide whether chunks should be merge or not. After finishing the merge chunk steps, I start count the number of applause and laughter in each chunk and also get the speakers name by pull out the first word after splitting by ":". In the end, I remove those unspoken text and output the text as well as the summary as a list.

```
#Process to make chunk text
text_chunk <- function(script){
  #Figure out the sentence with colon
  withcolon <- grep(":", script)

  #Count the number of the words befroe colon
  tcolon <- table(sapply(strsplit(script[withcolon], "\\:"), "[[", 1))

  #Get the name
  name <- names(tcolon[tcolon > 2])
  first <- name[1]; second <- name[2]; third <- name[3]

  #Determine which chunk should each sentence belong to
  chunk_n <- rep(0, length(script))
  chunk_n[grep(paste(first, ":", sep=""), script)] <- 1
  chunk_n[grep(paste(second, ":", sep=""), script)] <- 2
  chunk_n[grep(paste(third, ":", sep=""), script)] <- 3

  #Merge two chunk with same speaker
  script_character <- rep("0", sum(diff(chunk_n) != 0))

  #Initial Value
  j <- 1
  script_character[1] <- script[1]
  l <- chunk_n[1]
  for(i in 2:length(chunk_n)){
    if(chunk_n[i] != 0){
      if(chunk_n[i] == l){
        sentence <- gsub(paste(name, ":", sep="", collapse="|"), "", script[i])
        script_character[j] <- paste(script_character[j], sentence)
      }else{
        j <- j + 1
        sentence <- script[i]
        script_character[j] <- sentence
        l <- chunk_n[i]
      }
    }else{
      sentence <- script[i]
      script_character[j] <- paste(script_character[j], sentence)
    }
  }

  #Remove unused space
```

```
  chunk <- script_character[script_character != "0"]

  #Count the word LAUGHTER and APPLAUSE
  laughter <- sapply(strsplit(chunk[grep("\\(LAUGHTER\\)", chunk)], ":"), "[[", 1)
  applause <- sapply(strsplit(chunk[grep("\\(APPLAUSE\\)", chunk)], ":"), "[[", 1)

  summary_data <- data.frame(
  one=c(sum(laughter == first), sum(applause == first)),
  two=c(sum(laughter == second), sum(applause == second)),
  three=c(sum(laughter == third), sum(applause == third))
  )

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

  #Remove the non-spoken text
  chunk <- gsub("\\([[:alpha:]]+\\)", "", chunk)

  #Output
  output <- list()
  output$Spoken_text <- chunk
  output$summary <- summary_data
  output
}

#Chunk text for each year
chunk_2012 <- text_chunk(script_2012)
chunk_2008 <- text_chunk(script_2008)
chunk_2004 <- text_chunk(script_2004)
chunk_2000 <- text_chunk(script_2000)
chunk_1996 <- text_chunk(script_1996)

#script
head(chunk_2004$Spoken_text[[1]])

[1] "<br/>LEHRER: Good evening from the University of Miami Convocation Center in Coral Gables, Flori

#summary
chunk_2004$summary

          BUSH KERRY LEHRER
LAUGHTER     1     2      0
APPLAUSE     0     0      1
```

(d) To deal with the questions after (d), I think it will be more appropriate to stroe all speaker's words into each chunk, which means that the data will have three chunk(three character vector in a list). Therefore, I rewrite the function in (c) to make a speaker's words merge only into one chunk. With the same strategy of using the reference, for those words speaker by first person, I will store it into one chunk. The output will be a list.

```
#(d)
text_chunk_combine <- function(script){
  #Figure out the sentence with colon
  withcolon <- grep(":", script)
```

```
  #Count the number of the words befroe colon
  tcolon <- table(sapply(strsplit(script[withcolon], "\\:"), "[[", 1))

  #Get the name
  name <- names(tcolon[tcolon > 2])
  first <- name[1]; second <- name[2]; third <- name[3]

  #Determine which chunk should each sentence belong to
  chunk_n <- rep(0, length(script))
  chunk_n[grep(paste(first, ":", sep=""), script)] <- 1
  chunk_n[grep(paste(second, ":", sep=""), script)] <- 2
  chunk_n[grep(paste(third, ":", sep=""), script)] <- 3

  #Skip first one for speakers
  for(i in 2:length(chunk_n)){
    if(chunk_n[i] == 0){
      chunk_n[i] <- chunk_n[i - 1]
    }
  }

  #Put them into list as well as remove the redundant words
  chunk <- list(
  first=gsub(paste(first, ": ", sep=""), "", script[chunk_n == 1]),
  second=gsub(paste(second, ": ", sep=""), "", script[chunk_n == 2]),
  thrid=gsub(paste(third, ": ", sep=""), "", script[chunk_n == 3])
  )
  names(chunk) <- name

  #Combine all the sentences into one string
  chunk <- lapply(chunk, function(x) paste(x, collapse=" "))
  chunk <- lapply(chunk, function(x) gsub("<br/>", "", x))

  #Remove the non-spoken text
  chunk <- lapply(chunk, function(x) gsub("\\([[:alpha:]]+\\)", "", x))
  chunk
}

chunk_2012_combine <- text_chunk_combine(script_2012)
chunk_2008_combine <- text_chunk_combine(script_2008)
chunk_2004_combine <- text_chunk_combine(script_2004)
chunk_2000_combine <- text_chunk_combine(script_2000)
chunk_1996_combine <- text_chunk_combine(script_1996)
```

Then I split three character vectors in the list into sentence. I replace with period, question mark, exclaimation into period, question mark, exclaimation@. By doing so, I can use @ as a seperator to split each sentence. However, there will be some cases which should not be splitted into sentence like Mr.@. After replace those words back into origin like Mr., I use the split to get the sentence. To unlist what we get from the function and remove the empty and space, we will have a character vector which each element is a sentence.

```
#one element per sentence
sentence_data <- function(chunk){
  #Deal with period, question mark, exclaimation
```

```
  chunk_process1 <- unlist(lapply(chunk, gsub, pattern="\\. ", replacement=".@ "))
  chunk_process1 <- gsub(pattern="\\? ", replacement="?@ ", chunk_process1)
  chunk_process1 <- gsub(pattern="\\! ", replacement="!@ ", chunk_process1)

  #Change Mr back
  chunk_process2 <- gsub(pattern="Mr.@", replacement="Mr.", chunk_process1)

  #Split the Sentence
  strsplit(chunk_process2, "@ ")
}

sentence_2012 <- unlist(sentence_data(chunk_2012_combine))
#Remove space and empty
sentence_2012 <- sentence_2012[! sentence_2012 %in% c("", " ")]

sentence_2008 <- unlist(sentence_data(chunk_2008_combine))
sentence_2008 <- sentence_2008[! sentence_2008 %in% c("", " ")]

sentence_2004 <- unlist(sentence_data(chunk_2004_combine))
sentence_2004 <- sentence_2004[! sentence_2004 %in% c("", " ")]

sentence_2000 <- unlist(sentence_data(chunk_2000_combine))
sentence_2000 <- sentence_2000[! sentence_2000 %in% c("", " ")]

sentence_1996 <- unlist(sentence_data(chunk_1996_combine))
sentence_1996 <- sentence_2012[! sentence_1996 %in% c("", " ")]

head(sentence_2012)
```

```
"I'm Jim Lehrer of the \"PBS NewsHour,\" and I welcome you to the first of the 2012 presidential deba

                                                                                                  "Thi

                                                                                "There will be six

                                                                                     "Thousands
```

To get a character vector which its element is one word, I first remove all the punctuation and then split word by space to get words. Befroe outputting the result, I remove those element containing nothing. By unlisting the output of the function, we can get the words vector.

```
word_data <- function(chunk){
  chunk <- unlist(lapply(chunk, gsub, pattern="\\.|,|\\?|\\-", replacement=""))
  chunk_word <- strsplit(chunk, " ")
  chunk_word <- lapply(chunk_word, function(x) x[x != ""])
  chunk_word
}

word_2012 <- unlist(word_data(chunk_2012_combine))
```

```
word_2008 <- unlist(word_data(chunk_2008_combine))
word_2004 <- unlist(word_data(chunk_2004_combine))
word_2000 <- unlist(word_data(chunk_2000_combine))
word_1996 <- unlist(word_data(chunk_1996_combine))

head(word_2012)

  LEHRER1   LEHRER2   LEHRER3   LEHRER4   LEHRER5   LEHRER6
   "Good" "evening"    "from"     "the" "Magness"   "Arena"
```

(e) To count the average word length, I utilize the function **word_data** to get the words by each speaker(a list). Then I use the **nchar** and **mean** function to compute the words length.

All of the speakers' average word length are around 4.5. The word length of moderator, Lehrer, haven't change a lot. The word length of candidate tend to be more shorter than the moderator.

```
#(e)
ave_word_count <- function(chunk){
  chunk_word <- word_data(chunk)
  round(sapply(sapply(chunk_word, nchar), mean), 4)
}

ave_word_count(chunk_2012_combine)

LEHRER  OBAMA ROMNEY
4.4238 4.4826 4.3570

ave_word_count(chunk_2008_combine)

LEHRER MCCAIN  OBAMA
4.3545 4.4434 4.3944

ave_word_count(chunk_2004_combine)

  BUSH  KERRY LEHRER
4.3508 4.3183 4.7961

ave_word_count(chunk_2000_combine)

     BUSH      GORE MODERATOR
   4.3499    4.3793    4.6281

ave_word_count(chunk_1996_combine)

CLINTON    DOLE  LEHRER
 4.4077  4.3315  4.5893
```

(f) To find pattern in the word of each speaker, my function have two argument, **chunk** and **p**. **p** let users to input any pattern in a regular expression way they want to detect. To deal with the problem of God and God bless, I allow users to input a vector which its length is 3. The first and second element are the phrase they want to find, while the thrid element is to decide which kind of match they want. If the thrid argument is First, then the function will find the first element only(not followed by second element). If the argument is not First, then it will output the occurrence of matching the phrase(in this case, it will find the number of occurrence for God bless). With the function, I can use **sapply** to do the operation for each speakers' word. For "I", I search "I" as well as "I"' to catch the word like "I'll". For "We", I search "We", "We"', "we", "we'. For the rest of words, I just input the word on the problem without any processing.

The winner of the presidential election use more "I" and "free, freedom" in the debate. The word "war" happens many time in 2004 and 2008(by Obama). For other word, I did not detect any interesting pattern.

```
#(f)
count_word <- function(chunk, p){
  chunk_word <- word_data(chunk)
  if(length(p) == 1){
    sapply(sapply(chunk_word, function(x) grep(pattern=p, x)), length)
  }else if (length(p) == 3){
      if(p[3] == "First"){
        f <- function(x){
        in_one <- grep(p[1], x)
        in_one[x[in_one + 1] != p[2]]
        }
      }else{
        f <- function(x){
        in_second <- grep(p[2], x)
        in_second[x[in_second - 1] == p[1]]
      }
    }
    sapply(sapply(chunk_word, f), length)
  }
}

#I, we, America{,n}, democra{cy,tic}, republic
#Democrat{,ic}, Republican, free{,dom}, war
#God[not including God bless], God Bless
#{Jesus, Christ, Christian}
p <- list("^I$|I'", "^We$|We'|^we$|we'", "American|America",
       "democracy|democratic", "republic",
       "Democrat|Democratic", "Republican",
       c("God", "bless", "First"), c("God", "bless", "Second"),
       "free|freedom", "^war$", "Jesus|Christ|Chritian")
data.name <- c("I", "We", "America{,n}",
               "democracy{,ic}", "republic",
               "Democrat{,ic}", "Republican",
               "God[not including God bless]", "God bless",
               "free{,dom}", "war", "Jesus|Christ|Chritian")

#1996
table_1996 <- sapply(p, count_word, chunk=chunk_1996_combine)
colnames(table_1996) <- data.name
table_1996

          I  We America{,n} democracy{,ic} republic Democrat{,ic}
CLINTON 235 150          36              5        0             1
DOLE    272 166          50              0        0            12
LEHRER   13  15           1              0        0             1
        Republican God[not including God bless] God bless free{,dom} war
CLINTON         10                            0         0          8   2
DOLE            12                            0         1          1   2
LEHRER           2                            0         0          0   0
        Jesus|Christ|Chritian
```

```
CLINTON                     0
DOLE                        0
LEHRER                      0
```

```
#2000
table_2000 <- sapply(p, count_word, chunk=chunk_2000_combine)
colnames(table_2000) <- data.name
table_2000
```

```
            I  We America{,n} democracy{,ic} republic Democrat{,ic}
BUSH      213 109          26              1        0            12
GORE      228  96          16              1        0             2
MODERATOR  14  20           0              1        1             1
          Republican God[not including God bless] God bless free{,dom} war
BUSH               9                             0         0          4   4
GORE               2                             0         0          1   3
MODERATOR          1                             0         0          0   0
          Jesus|Christ|Chritian
BUSH                        0
GORE                        0
MODERATOR                   0
```

```
#2004
table_2004 <- sapply(p, count_word, chunk=chunk_2004_combine)
colnames(table_2004) <- data.name
table_2004
```

```
         I  We America{,n} democracy{,ic} republic Democrat{,ic}
BUSH   177 170          24              4        0             0
KERRY  195 148          46              2        0             0
LEHRER   9   5           3              1        0             1
       Republican God[not including God bless] God bless free{,dom} war
BUSH            0                             1         0         36  24
KERRY           1                             0         1          3  36
LEHRER          1                             0         0          0   1
       Jesus|Christ|Chritian
BUSH                       0
KERRY                      0
LEHRER                     0
```

```
#2008
table_2008 <- sapply(p, count_word, chunk=chunk_2008_combine)
colnames(table_2008) <- data.name
table_2008
```

```
         I  We America{,n} democracy{,ic} republic Democrat{,ic}
LEHRER  15  15           0              0        0             1
MCCAIN 211 167          24              1        0             3
OBAMA  145 267          16              1        0             0
       Republican God[not including God bless] God bless free{,dom} war
LEHRER          1                             0         0          1   0
MCCAIN          7                             0         0          4   5
OBAMA           3                             0         0          5  12
       Jesus|Christ|Chritian
LEHRER                     0
```

```
MCCAIN                    2
OBAMA                     0

#2012
table_2012 <- sapply(p, count_word, chunk=chunk_2012_combine)
colnames(table_2012) <- data.name
table_2012

         I  We America{,n} democracy{,ic} republic Democrat{,ic}
LEHRER  21  27           3              0        0             1
OBAMA  118 181          24              0        0             8
ROMNEY 214 124          39              1        0             7
       Republican God[not including God bless] God bless free{,dom} war
LEHRER          1                            0         0          0   0
OBAMA           9                            0         0          3   2
ROMNEY          9                            0         0          7   0
       Jesus|Christ|Chritian
LEHRER                    0
OBAMA                     0
ROMNEY                    0
```

3. (a) and (b) For the beginning of the function, I use several **if** statement to check whether the argument is correctly inputed. Then, I create a movement matrix which contain up, down, right, left in each row by two-coordinate. To repeat the sample process, the **replicate** is useful which give me n samples and I can use them to pull out the movement from my matrix. Finally, the function **cumsum** computes the coordinates of each step. With some name processing and output checking, the function will output the whole process by a matrix with n + 1 rows and 2 columns.

```
#Problem 3 (a)
myWalk <- function(n, full_path=TRUE){
  #Check whether n is an integer
  if((n %% 1) != 0) stop("Please input an integer")
  #Check whether n is positive
  if(n <= 0) stop("The number of step is negative. Please input a POSITIVE number")
  #Check the full_path input
  if(!is.logical(full_path)) stop("For the argument full_path, you should input TRUE/FALSE")

  #Setup
  up <- c(0, 1); down <- c(0, -1); right <- c(1, 0); left <- c(-1, 0)
  m <- rbind(up, down, right, left); colnames(m) <- c("x1", "x2")

  #Repeat the sample n times and get the row index
  #Same as sample(1:4, n, replace=TRUE)
  process <- replicate(n, sample(1:4, 1))

  #Sum cumulatively by column
  path <- cbind(cumsum(m[process, 1]),
                cumsum(m[process, 2]))
  #Change names
  colnames(path) <- c("x1", "x2")
  rownames(path) <- paste("Step", 1:n, sep=" ")

  #Decide Output
  if(full_path){
```

```
    path
  }else{
    path[n, ]
  }
}

#Demo
myWalk(10)

        x1 x2
Step 1  -1  0
Step 2  -1  1
Step 3  -2  1
Step 4  -3  1
Step 5  -3  2
Step 6  -4  2
Step 7  -3  2
Step 8  -2  2
Step 9  -1  2
Step 10 -1  1
```

(c) The function is almost the same as the one in (a). I just add an argument **Start** which gives the location of origin. To be more meanful, I change my element when I do sample from 1~4 to character "up", "down","right","left". In the last part, I create a list to store all the information of the process and assign it into **obj** and set its class to **'rw'**.

```
#(c)
myWalk <- function(n, full_path=TRUE, Start=c(0, 0)){
  #Check whether n is an integer
  if((n %% 1) != 0) stop("Please input an integer")
  #Check whether n is positive
  if(n <= 0) stop("The number of step is negative. Please input a POSITIVE number")
  #Check the full_path input
  if(!is.logical(full_path)) stop("For the argument full_path, you should input TRUE/FALSE")

  #Start
  path <- matrix(0, nrow=n + 1, ncol=2)
  colnames(path) <- c("x1", "x2")
  path[1, ] <- Start

  #Setup
  up <- c(0, 1); down <- c(0, -1); right <- c(1, 0); left <- c(-1, 0)
  m <- rbind(up, down, right, left); colnames(m) <- c("x1", "x2")

  #Repeat the sample n times and get the row index
  process <- replicate(n, sample(c("up", "down", "right", "left"), 1))

  #Sum cumulatively by column
  path[-1, ] <- cbind(Start[1] + cumsum(m[process, 1]),
                      Start[2] + cumsum(m[process, 2]))

  #Change row  names
  rownames(path) <- c("Start", paste("Step", 1:n, sep=" "))
  #Create object
```

```
  obj <- list(Start=Start,
              Process=process,
              End=path[n + 1, ],
              Path=path)
  class(obj) <- "rw"
  obj
}

#Demo
walk1 <- myWalk(10)
walk1

$Start
[1] 0 0

$Process
 [1] "left"  "down"  "right" "up"    "down"  "down"  "right" "left"
 [9] "right" "left"

$End
x1 x2
 0 -2

$Path
        x1 x2
Start    0  0
Step 1  -1  0
Step 2  -1 -1
Step 3   0 -1
Step 4   0  0
Step 5   0 -1
Step 6   0 -2
Step 7   1 -2
Step 8   0 -2
Step 9   1 -2
Step 10  0 -2

attr(,"class")
[1] "rw"
```

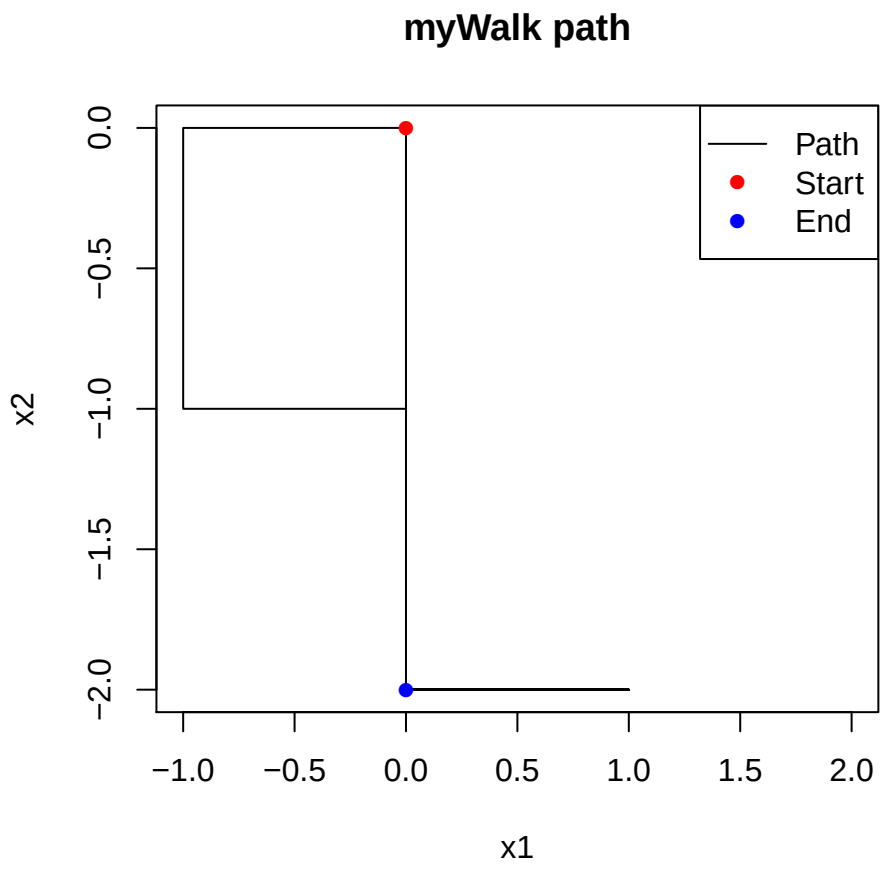
The print method for class rw will give a brief sentence to let user know where the process start and end. Also, it will give the summary table for occurence of the 4 movement and detailed process in abbreviation of the first letter for movement.

```
#print
print.rw <- function(object){
  cat("Origin: (", object$Start[1], ",", object$Start[2], ")", "\t", sep="")
  cat("End: (", object$End[1], ",", object$End[2], ")", "\n", sep="")
  #Summary
  tb <- table(object$Process)
  cat("The whole process is summary by following table", "\n")
  print(tb)
  cat("\n", "Detailed process:", paste(substr(object$Process, 1, 1), collapse=""), "\n")
  cat(" ", "u(Up), d(Down), r(Right), l(Left)")
```

```
}

walk1

Origin: (0,0) End: (0,-2)
The whole process is summary by following table

 down  left right    up
    3     3     3     1

 Detailed process: ldruddrlrl
  u(Up), d(Down), r(Right), l(Left)
```

In the plot method, the function will output the path out with Start pointed red and End pointed blue. Also, I add a legend in the plot.

```
#plot
plot.rw <- function(object){
  #Make plot
  with(object, plot(Path, type="l", main="myWalk path",
                    xlab="x1", ylab="x2",
                    xlim=c(min(Path[, 1]), max(Path[, 1]) + 1)))
  #Add point
  with(object, points(Start[1], Start[2], col="Red", pch=16))
  with(object, points(End[1], End[2], col="Blue", pch=16))

  #Add legend
  legend("topright", legend=c("Path", "Start", "End"),
         lty=c(1, NA, NA), pch=c(NA, 16, 16),
         col=c("black", "red", "blue"))
}

plot(walk1)
```



The '[' operator will give you the position of ith step with origin as 0 and 1st step as 1.

```
#[
`[.rw` <- function(object, i){
  object$Path[i + 1, ]
}

walk1[1]; walk1[3]

x1 x2
-1  0
x1 x2
 0 -1
```

The **start** function will change all thing in the object in class rw by the origin which the user specifies.

```
#Start
start.rw <- function(object, x){
  #Change Start
  object$Start <- x

  #Change Path
  object$Path <- object$Path + x
```

```
  #Update N
  object$End <- object$Path[dim(object$Path)[1], ]

  object
}

start(walk1, c(0, 5))
```

```
Origin: (0,5) End: (0,3)
The whole process is summary by following table

 down  left right    up
    3     3     3     1

 Detailed process: ldruddrlrl
  u(Up), d(Down), r(Right), l(Left)
```