Lab 6: Functions

Name: Rufus Petrie

This week's agenda: creating and updating functions; understanding argument and return structures; revisiting Shakespeare's plays; code refactoring.

Huber loss function

The Huber loss function (or just Huber function, for short) is defined as:

$$\psi(x) = \begin{cases} x^2 & \text{if } |x| \le 1 \\ 2|x| - 1 & \text{if } |x| > 1 \end{cases}$$

It is often used in place of the usual squared error loss for robust estimation. The sample average, $\bar{X}$ – which given a sample $X_1, \ldots, X_n$ minimizes the squared error loss $\sum_{i=1}^n (X_i - m)^2$ over all choices of m – can be inaccurate as an estimate of $\mathbb{E}(X)$ if the distribution of X is heavy-tailed. In such cases, minimizing Huber loss can give a better estimate. (Interested in hearing more? Come ask one of us, or ask your 401 or 402 Professor!)

Some simple function tasks

- **1a.** Write a function `huber()` that takes as an input a number x, and returns the Huber value $\psi(x)$, as defined above. Hint: the body of a function is just a block of R code, e.g., in this code you can use `if()` and `else()` statements. Check that `huber(1)` returns 1, and `huber(4)` returns 7.

```
huber = function(x){
  if(abs(x)<=1){
    return(x^2)
  }
  else{
    return(2*abs(x)-1)
  }
}

huber(1)
```

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

```
huber(4)
```

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

- **1b.** The Huber function can be modified so that the transition from quadratic to linear happens at an arbitrary cutoff value a, as in:

$$\psi_a(x) = \begin{cases} x^2 & \text{if } |x| \le a \\ 2a|x| - a^2 & \text{if } |x| > a \end{cases}$$

Starting with your solution code to the last question, update your `huber()` function so that it takes two arguments: x, a number at which to evaluate the loss, and a a number representing the cutoff value. It should now return $\psi_a(x)$, as defined above. Check that `huber(3, 2)` returns 8, and `huber(3, 4)` returns 9.

```
huber = function(x, a){
  if(abs(x)<=a){
    return(x^2)
  }
  else{
    return(2*a*abs(x)-a^2)
  }
}

huber(3,2)
```

```
## [1] 8
```

```
huber(3,4)
```

```
## [1] 9
```

- **1c.** Update your `huber()` function so that the default value of the cutoff a is 1. Check that `huber(3)` returns 5.

```
huber = function(x, a=1){
  if(abs(x)<=a){
    return(x^2)
  }
  else{
    return(2*a*abs(x)-a^2)
  }
}

huber(3)
```

```
## [1] 5
```

- **1d.** Check that `huber(a=1, x=3)` returns 5. Check that `huber(1, 3)` returns 1. Explain why these are different.

```
huber(a=1, x=3)
```

```
## [1] 5
```

```
huber(1, 3)
```

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

These calls return different results because the order of the inputs is flipped in the first call.

- **1e.** Vectorize your `huber()` function, so that the first input can actually be a vector of numbers, and what is returned is a vector whose elements give the Huber evaluated at each of these numbers. Hint: you might try using `ifelse()`, if you haven't already, to vectorize nicely. Check that `huber(x=1:6, a=3)` returns the vector of numbers (1, 4, 9, 15, 21, 27).

```
huber = function(x, a=1){
  return(ifelse(abs(x)<=a, x^2, 2*a*abs(x)-a^2))
}
```

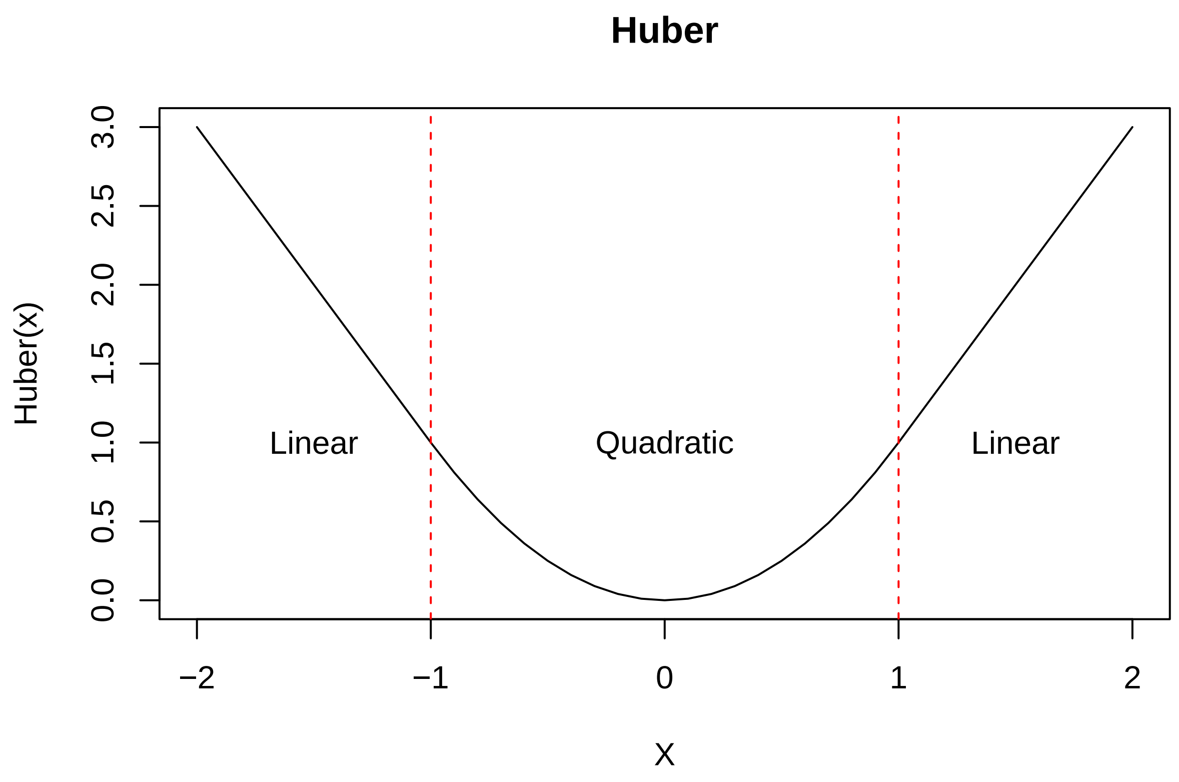
```r
huber(x=1:6, a=3)
```

```
## [1]  1  4  9 15 21 27
```

Plotting practice, side effects

- **2a.** Professor Tibs created in plot of the Huber function displayed at the top of this lab in R. Reproduce this plot with your own plotting code, and the `huber()` function you wrote above. The axes and title should be just the same, so should the Huber curve (in black), so should be the red dotted lines at the values -1 and 1, and so should the text "Linear", "Quadratic", "Linear".

```r
x <- seq(-2, 2, 0.1)
plot(x, huber(x), type="l",
     main = "Huber", xlab = "X", ylab = "Huber(x)")
abline(v=c(-1,1), col="red", lty=2)
text("Linear", x=-1.5, y=1)
text("Quadratic", x=0, y=1)
text("Linear", x=1.5, y=1)
```



- **2b.** Modify the `huber()` function so that, as a side effect, it prints the string "Invented by the great Swiss statistician Peter Huber!" to the console. Hint: use `cat()`. Call your function on an input of your choosing, to demonstrate this side effect.

```r
huber = function(x, a=1){
  cat("Invented by the great Swiss statistician Peter Huber!")
  return(ifelse(abs(x)<=a, x^2, 2*a*abs(x)-a^2))
}

huber(x=1:6, a=3)
```

```
## Invented by the great Swiss statistician Peter Huber!
```

```
## [1]  1  4  9 15 21 27
```

Exploring function environments

- **3a.** A modified version of the Huber function called `huber.mod()` is given below. You can see that we've defined the variable `x.squared` in the body of the function to be the square of the input argument `x`. In a separate line of code (outside of the function definition), define the variable `x.squared` to be equal to 999. Then call `huber.mod(x=3)`, and display the value of `x.squared`. What is its value? Is this affected by the function call `huber.mod(x=3)`? It shouldn't be! Reiterate this point with several more lines of code, in which you repeatedly define `x.squared` to be something different (even something nonnumeric, like a string), and then call `huber.mod(x=3)`, and demonstrate afterwards that the value of `x.squared` hasn't changed.

```
x <- 999
huber.mod = function(x, a=1) {
  x.squared = x^2
  ifelse(abs(x) <= a, x.squared, 2*a*abs(x)-a^2)
}
huber.mod(x=3)
```

```
## [1] 5
```

- **3b.** Similar to the last question, define the variable `a` to be equal to -59.6, then call `huber.mod(x=3, a=2)`, and show that the value of `a` after this function call is unchanged. And repeat a few times with different assignments for the variable `a`, to reiterate this point.

```
a <- -59.6
huber.mod(x=3, a=2)
```

```
## [1] 8
```

- **3c.** The previous two questions showed you that a function's body has its own environment in which locally defined variables, like those defined in the body itself, or those defined through inputs to the function, take priority over those defined outside of the function. However, when a variable referred to the body of a function is *not defined in the local environment*, the default is to look for it in the global environment (outside of the function).

  Below is a "sloppy" implementation of the Huber function called `huber.sloppy()`, in which the cutoff `a` is not passed as an argument to the function. In a separate line of code (outside of the function definition), define `a` to be equal to 1.5 and then call `huber.sloppy(x=3)`. What is the output? Explain. Repeat this a few times, by defining `a` and then calling `huber.sloppy(x=3)`, to show that the value of `a` does indeed affect the function's ouptut as expected. **Challenge**: try setting `a` equal to a string and calling `huber.sloppy(x=3)`; can you explain what is happening?

```
a=2
huber.sloppy = function(x) {
  ifelse(abs(x) <= a, x^2, 2*a*abs(x)-a^2)
}
huber.sloppy(x=3)
```

```
## [1] 8
```

- **3d.** At last, a difference between `=` and `<-`, explained! Some of you have asked about this. The equal sign `=` and assignment operator `<-` are often used interchangeably in R, and some people will often say that a choice between the two is mostly a matter of stylistic taste. This is not the full story. Indeed, `=` and `<-` behave very differently when used to set input arguments in a function call. As we showed above, setting, say, `a=5` as the input to `huber()` has no effect on the global assignment for `a`. However, replacing `a=5` with `a<-5` in the call to `huber()` is entirely different in terms of its effect on `a`. Demonstrate this, and explain what you are seeing in terms of global assignment.

```
a <- 1
huber(x <- 3, a <- 2)
```

```
## Invented by the great Swiss statistician Peter Huber!
```

```
## [1] 8
```

```
a
```

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

```
x
```

```
## [1] 3
```

Assignments passed into function calls will affect global variables.

- **3e.** The story now gets even more subtle. It turns out that the assignment operator `<-` allows us to define new global variables even when we are specifying inputs to a function. Pick a variable name that has not been defined yet in your workspace, say `b` (or something else, if this has already been used in your R Markdown document). Call `huber(x=3, b<-20)`, then display the value of `b`– this variable should now exist in the global enviroment, and it should be equal to 20! Also, can you explain the output of `huber(x=3, b<-20)`?

```
huber(x=3, b <- 20)
```

```
## Invented by the great Swiss statistician Peter Huber!
```

```
## [1] 9
```

X is under the threshold, so the output just equals $x^2 = 9$.

Shakespeare's complete works

Recall, as we saw in Week 4, that the complete works of William Shakespeare are available freely from Project Gutenberg. We've put this text file up at http://www.stat.cmu.edu/~ryantibs/statcomp/data/shakespeare.txt.

Getting lines of text play-by-play

- **4a.** Below is the `get.wordtab.from.url()` from lecture. Modify this function so that the string vectors `lines` and `words` are both included as named components in the returned list. For good practice, update the documentation in comments to reflect your changes. Then call this function on the URL for the Shakespeare's complete works (with the rest of the arguments at their default values) and save the result as `shakespeare.wordobj`. Using `head()`, display the first several elements of (definitely not all of!) the `lines`, `words`, and `wordtab` components of `shakespeare.wordobj`, just to check that the output makes sense to you.

```
# get.wordtab.from.url: get a word table from text on the web
# Inputs:
# - str.url: string, specifying URL of a web page
```

```r
# - split: string, specifying what to split on. Default is the regex pattern
#   "[[:space:]]|[[:punct:]]"
# - tolower: Boolean, TRUE if words should be converted to lower case before
#   the word table is computed. Default is TRUE
# - keep.nums: Boolean, TRUE if words containing numbers should be kept in the
#   word table. Default is FALSE
# Output: list, containing lines, words, word table, and some basic summaries

get.wordtab.from.url = function(str.url, split="[[:space:]]|[[:punct:]]",
                                tolower=TRUE, keep.nums=FALSE) {
  lines = readLines(str.url)
  text = paste(lines, collapse=" ")
  lines = lines[lines != ""]
  words = strsplit(text, split=split)[[1]]
  words = words[words != ""]

  # Convert to lower case, if we're asked to
  if (tolower) words = tolower(words)

  # Get rid of words with numbers, if we're asked to
  if (!keep.nums)
    words = grep("[0-9]", words, inv=TRUE, val=TRUE)

  # Compute the word table
  wordtab = table(words)

  return(list(wordtab=wordtab,
              lines = lines,
              words = words,
              number.unique.words=length(wordtab),
              number.total.words=sum(wordtab),
              longest.word=words[which.max(nchar(words))]))
}

shakespeare.wordobj <-
  get.wordtab.from.url("http://www.stat.cmu.edu/~ryantibs/statcomp/data/shakespeare.txt")

# Wordtab
head(shakespeare.wordobj$wordtab)
# Lines
head(shakespeare.wordobj$lines, 6)
# Words
head(shakespeare.wordobj$words, 6)
```

- **4b.** Go back and look Q5 of Lab 4, where you located Shakespeare's plays in the lines of text for Shakespeare's complete works. Set `shakespeare.lines = shakespeare.wordobj$lines`, and then rerun your solution code (or the rerun the official solution code, if you'd like) for Q5 of Lab 4 on the lines of text stored in `shakespeare.lines`. (Note: you don't actually need to rerun the code for Q5d or Q5e, since the code for Q5f will accomplish the same task only without encountering `NA`s). You should end up with two vectors `titles.start` and `titles.end`, containing the start and end positions of each of Shakespeare's plays in `shakespeare.lines`. Print out `titles.start` and `titles.end` to the console.

```r
shakespeare.lines = shakespeare.wordobj$lines
shakespeare.lines <- trimws(shakespeare.lines)
toc.start <- which(shakespeare.lines == "THE SONNETS")[1]
toc.end <- which(shakespeare.lines == "VENUS AND ADONIS")[1]

n <- toc.end - toc.start + 1
titles <- vector(length = n)
for(i in 1:n){
  titles[i] <- shakespeare.lines[toc.start+ i-1]
}

titles.start <- vector(length = n)
for(i in 1:n){
  titles.start[i] <-  grep(pattern=titles[i], x=shakespeare.lines)[2]
}

titles.end <- vector(length = n)
for(i in 1:(n-1)){
  titles.end[i] <- titles.start[i+1]-1
}
titles.end[n] <- length(shakespeare.lines)

titles.start
```

```
##  [1]     67   2378   5311   9142  11773  13703  17591  21386  26645  30390
## [11]  33615  36903  39958  43249  46413  49896  52681  55428  60108  62924
## [21]  65463  68320  71021  73767  75997  79470  83084  86328  89287  93443
## [31]  97536 101206 103641 106199 108939 113683 116176 118464 122683 126021
## [41] 126352 126557 126627 128535
```

```r
titles.end
```

```
##  [1]   2377   5310   9141  11772  13702  17590  21385  26644  30389  33614
## [11]  36902  39957  43248  46412  49895  52680  55427  60107  62923  65462
## [21]  68319  71020  73766  75996  79469  83083  86327  89286  93442  97535
## [31] 101205 103640 106198 108938 113682 116175 118463 122682 126020 126351
## [41] 126556 126626 128534 130095
```

- **4c.** Create a list `shakespeare.lines.by.play` of length equal to the number of Shakespeare's plays (a number you should have already computed in the solution to the last question). Using a `for()` loop, and relying on `titles.start` and `titles.end`, extract the appropriate subvector of `shakespeare.lines` for each of Shakespeare's plays, and store it as a component of `shakespeare.lines.by.play`. That is, `shakespeare.lines.by.play[[1]]` should contain the lines for Shakespeare's first play, `shakespeare.lines.by.play[[2]]` should contain the lines for Shakespeare's second play, and so on. Name the components of `shakespeare.lines.by.play` according to the titles of the plays.

```r
shakespeare.lines.by.play <- vector(mode = "list", length = n)
for(i in 1:n){
  shakespeare.lines.by.play[[i]] <- shakespeare.wordobj$lines[titles.start[i]:titles.end[i]]
names(shakespeare.lines.by.play) <- titles
}
```

- **4d.** Using one of the apply functions, along with `head()`, print the first 4 lines of each of Shakespeare's plays to the console (sorry graders . . .). This should only require one line of code.

```r
lapply(shakespeare.lines.by.play, FUN = head, n=4)
```

```
## $`THE SONNETS`
## [1] "THE SONNETS"
## [2] "                    1"
## [3] "From fairest creatures we desire increase,"
## [4] "That thereby beautyâ\200\231s rose might never die,"
## 
## $`ALLâ\200\231S WELL THAT ENDS WELL`
## [1] "ALLâ\200\231S WELL THAT ENDS WELL" "Dramatis Personae"
## [3] "  KING OF FRANCE"            "  THE DUKE OF FLORENCE"
## 
## $`THE TRAGEDY OF ANTONY AND CLEOPATRA`
## [1] "THE TRAGEDY OF ANTONY AND CLEOPATRA" "DRAMATIS PERSONAE"
## [3] "  MARK ANTONY,         Triumvirs"    "  OCTAVIUS CAESAR,         \""
## 
## $`AS YOU LIKE IT`
## [1] "AS YOU LIKE IT"
## [2] "DRAMATIS PERSONAE."
## [3] "  DUKE, living in exile"
## [4] "  FREDERICK, his brother, and usurper of his dominions"
## 
## $`THE COMEDY OF ERRORS`
## [1] "THE COMEDY OF ERRORS"           "DRAMATIS PERSONAE"
## [3] "SOLINUS, Duke of Ephesus"       "AEGEON, a merchant of Syracuse"
## 
## $`THE TRAGEDY OF CORIOLANUS`
## [1] "THE TRAGEDY OF CORIOLANUS"
## [2] "Dramatis Personae"
## [3] "  CAIUS MARCIUS, afterwards CAIUS MARCIUS CORIOLANUS"
## [4] "    Generals against the Volscians"
## 
## $CYMBELINE
## [1] "CYMBELINE"
## [2] "Dramatis Personae"
## [3] "  CYMBELINE, King of Britain"
## [4] "  CLOTEN, son to the Queen by a former husband"
## 
## $`THE TRAGEDY OF HAMLET, PRINCE OF DENMARK`
## [1] "THE TRAGEDY OF HAMLET, PRINCE OF DENMARK"
## [2] "by William Shakespeare"
## [3] "Contents"
## [4] "ACT I"
## 
## $`THE FIRST PART OF KING HENRY THE FOURTH`
## [1] "THE FIRST PART OF KING HENRY THE FOURTH"
## [2] "by William Shakespeare"
## [3] " Dramatis PersonÃ¦"
## [4] "KING HENRY the Fourth."
## 
## $`THE SECOND PART OF KING HENRY THE FOURTH`
## [1] "THE SECOND PART OF KING HENRY THE FOURTH"
## [2] "Dramatis Personae"
## [3] "  RUMOUR, the Presenter"
## [4] "  KING HENRY THE FOURTH"
##
## $`THE LIFE OF KING HENRY THE FIFTH`
## [1] "THE LIFE OF KING HENRY THE FIFTH" "DRAMATIS PERSONAE"
## [3] "  CHORUS"                         "  KING HENRY THE FIFTH"
##
## $`THE FIRST PART OF HENRY THE SIXTH`
## [1] "THE FIRST PART OF HENRY THE SIXTH"
## [2] "Dramatis Personae"
## [3] "  KING HENRY THE SIXTH"
## [4] "  DUKE OF GLOUCESTER, uncle to the King, and Protector"
##
## $`THE SECOND PART OF KING HENRY THE SIXTH`
## [1] "THE SECOND PART OF KING HENRY THE SIXTH"
## [2] "Dramatis Personae"
## [3] "  KING HENRY THE SIXTH"
## [4] "  HUMPHREY, DUKE OF GLOUCESTER, his uncle"
##
## $`THE THIRD PART OF KING HENRY THE SIXTH`
## [1] "THE THIRD PART OF KING HENRY THE SIXTH"
## [2] "DRAMATIS PERSONAE"
## [3] "  KING HENRY THE SIXTH"
## [4] "  EDWARD, PRINCE OF WALES, his son"
##
## $`KING HENRY THE EIGHTH`
## [1] "KING HENRY THE EIGHTH"
## [2] "DRAMATIS PERSONAE"
## [3] "  KING HENRY THE EIGHTH"
## [4] "  CARDINAL WOLSEY               CARDINAL CAMPEIUS"
##
## $`KING JOHN`
## [1] "KING JOHN"                 "DRAMATIS PERSONAE"
## [3] "    KING JOHN"             "    PRINCE HENRY, his son"
##
## $`THE TRAGEDY OF JULIUS CAESAR`
## [1] "THE TRAGEDY OF JULIUS CAESAR"
## [2] "Dramatis Personae"
## [3] "  JULIUS CAESAR, Roman statesman and general"
## [4] "  OCTAVIUS, Triumvir after Caesar's death, later Augustus Caesar,"
##
## $`THE TRAGEDY OF KING LEAR`
## [1] "THE TRAGEDY OF KING LEAR" "by William Shakespeare"
## [3] "Contents"                 "ACT I"
##
## $`LOVEâ\200\231S LABOURâ\200\231S LOST`
## [1] "LOVEâ\200\231S LABOURâ\200\231S LOST"
## [2] "Dramatis Personae."
## [3] "  FERDINAND, King of Navarre"
## [4] "  BEROWNE,    lord attending on the King"
##
## $`THE TRAGEDY OF MACBETH`
## [1] "THE TRAGEDY OF MACBETH"
## [2] "Dramatis Personae"
## [3] "  DUNCAN, King of Scotland"
## [4] "  MACBETH, Thane of Glamis and Cawdor, a general in the King's army"
##
## $`MEASURE FOR MEASURE`
## [1] "MEASURE FOR MEASURE"   "DRAMATIS PERSONAE"     "  VINCENTIO, the Duke"
## [4] "  ANGELO, the Deputy"
##
## $`THE MERCHANT OF VENICE`
## [1] "THE MERCHANT OF VENICE"
## [2] "DRAMATIS PERSONAE"
## [3] "  THE DUKE OF VENICE"
## [4] "  THE PRINCE OF MOROCCO, suitor to Portia"
##
## $`THE MERRY WIVES OF WINDSOR`
## [1] "THE MERRY WIVES OF WINDSOR"  "Dramatis Personae"
## [3] "  SIR JOHN FALSTAFF"         "  FENTON, a young gentleman"
##
## $`A MIDSUMMER NIGHTâ\200\231S DREAM`
## [1] "A MIDSUMMER NIGHTâ\200\231S DREAM" "DRAMATIS PERSONAE"
## [3] "  THESEUS, Duke of Athens"   "  EGEUS, father to Hermia"
##
## $`MUCH ADO ABOUT NOTHING`
## [1] " MUCH ADO ABOUT NOTHING"      "      by William Shakespeare"
## [3] "Contents"                     "ACT I"
##
## $`THE TRAGEDY OF OTHELLO, MOOR OF VENICE`
## [1] "THE TRAGEDY OF OTHELLO, MOOR OF VENICE"
## [2] "Dramatis Personae"
## [3] "  OTHELLO, the Moor, general of the Venetian forces"
## [4] "  DESDEMONA, his wife"
##
## $`PERICLES, PRINCE OF TYRE`
## [1] "PERICLES, PRINCE OF TYRE" "by William Shakespeare"
## [3] "Contents"                 "ACT I"
##
## $`KING RICHARD THE SECOND`
## [1] "KING RICHARD THE SECOND"
## [2] "DRAMATIS PERSONAE"
## [3] "  KING RICHARD THE SECOND"
## [4] "  JOHN OF GAUNT, Duke of Lancaster - uncle to the King"
##
## $`KING RICHARD THE THIRD`
## [1] "KING RICHARD THE THIRD" "Dramatis Personae"      "  EDWARD THE FOURTH"
## [4] "    Sons to the King"
##
## $`THE TRAGEDY OF ROMEO AND JULIET`
## [1] "THE TRAGEDY OF ROMEO AND JULIET" "by William Shakespeare"
## [3] "PERSONS REPRESENTED"             "Escalus, Prince of Verona."
##
## $`THE TAMING OF THE SHREW`
## [1] "THE TAMING OF THE SHREW" "by William Shakespeare"
## [3] "Contents"                "INDUCTION"
##
## $`THE TEMPEST`
## [1] "THE TEMPEST"              "DRAMATIS PERSONAE"
## [3] "  ALONSO, King of Naples" "  SEBASTIAN, his brother"
##
## $`THE LIFE OF TIMON OF ATHENS`
## [1] "THE LIFE OF TIMON OF ATHENS" "DRAMATIS PERSONAE"
## [3] "    TIMON of Athens"         "    LUCIUS"
##
## $`THE TRAGEDY OF TITUS ANDRONICUS`
## [1] "THE TRAGEDY OF TITUS ANDRONICUS"
## [2] "Dramatis Personae"
## [3] "  SATURNINUS, son to the late Emperor of Rome, afterwards Emperor"
## [4] "  BASSIANUS, brother to Saturninus"
##
## $`THE HISTORY OF TROILUS AND CRESSIDA`
## [1] "THE HISTORY OF TROILUS AND CRESSIDA" "by William Shakespeare"
## [3] "Contents"                            "ACT I"
##
## $`TWELFTH NIGHT; OR, WHAT YOU WILL`
## [1] "TWELFTH NIGHT; OR, WHAT YOU WILL" "DRAMATIS PERSONAE"
## [3] "  ORSINO, Duke of Illyria"        "  SEBASTIAN, brother of Viola"
##
## $`THE TWO GENTLEMEN OF VERONA`
## [1] "THE TWO GENTLEMEN OF VERONA"
## [2] "DRAMATIS PERSONAE"
## [3] "  DUKE OF MILAN, father to Silvia"
## [4] "  VALENTINE, one of the two gentlemen"
##
## $`THE TWO NOBLE KINSMEN`
## [1] "THE TWO NOBLE KINSMEN:"
## [2] "Presented at the Blackfriers by the Kings Maiesties servants, with"
## [3] "great applause:"
## [4] "Written by the memorable Worthies of their time;"
##
## $`THE WINTERâ\200\231S TALE`
## [1] "THE WINTERâ\200\231S TALE"
## [2] "Dramatis Personae"
## [3] "  LEONTES, King of Sicilia"
## [4] "  MAMILLIUS, his son, the young Prince of Sicilia"
##
## $`A LOVERâ\200\231S COMPLAINT`
## [1] "A LOVERâ\200\231S COMPLAINT"
## [2] " From off a hill whose concave womb reworded"
## [3] "  A plaintful story from a sist'ring vale,"
## [4] "  My spirits t'attend this double voice accorded,"
##
## $`THE PASSIONATE PILGRIM`
## [1] "THE PASSIONATE PILGRIM"
## [2] "by William Shakespeare"
## [3] "I."
## [4] "Did not the heavenly rhetoric of thine eye,"
##
## $`THE PHOENIX AND THE TURTLE`
## [1] "THE PHOENIX AND THE TURTLE"   "by William Shakespeare"
## [3] "Let the bird of loudest lay," "On the sole Arabian tree,"
##
```

```
## $`THE RAPE OF LUCRECE`
## [1] "THE RAPE OF LUCRECE"                 "by William Shakespeare"
## [3] "                 THE RAPE OF LUCRECE" "                       TO THE"
##
## $`VENUS AND ADONIS`
## [1] " VENUS AND ADONIS"
## [2] " by William Shakespeare"
## [3] "            _Vilia miretur vulgus; mihi flavus Apollo"
## [4] "            Pocula Castalia plena ministret aqua._"
```

Getting word tables play-by-play

- **5a.** Define a function `get.wordtab.from.lines()` to have the same argument structure as `get.wordtab.from.url()`, which recall you last updated in Q2a, except that the first argument of `get.wordtab.from.lines()` should be `lines`, a string vector passed by the user that contains lines of text to be processed. The body of `get.wordtab.from.lines()` should be the same as `get.wordtab.from.url()`, except that `lines` is passed and does not need to be computed using `readlines()`. The output of `get.wordtab.from.lines()` should be the same as `get.wordtab.from.url()`, except that `lines` does not need to be returned as a component. For good practice, incude documentation for your `get.wordtab.from.lines()` function in comments.

```r
get.wordtab.from.lines = function(lines, split="[[:space:]]|[[:punct:]]",
                                  tolower=TRUE, keep.nums=FALSE) {
  text = paste(lines, collapse=" ")
  lines = lines[lines != ""]
  words = strsplit(text, split=split)[[1]]
  words = words[words != ""]

  # Convert to lower case, if we're asked to
  if (tolower) words = tolower(words)

  # Get rid of words with numbers, if we're asked to
  if (!keep.nums)
    words = grep("[0-9]", words, inv=TRUE, val=TRUE)

  # Compute the word table
  wordtab = table(words)

  return(list(wordtab=wordtab,
              words = words,
              number.unique.words=length(wordtab),
              number.total.words=sum(wordtab),
              longest.word=words[which.max(nchar(words))]))
}
```

- **5b.** Using a `for()` loop or one of the apply functions (your choice here), run the `get.wordtab.from.lines()` function on each of the components of `shakespeare.lines.by.play`, (with the rest of the arguments at their default values). Save the result in a list called `shakespeare.wordobj.by.play`. That is, `shakespeare.wordobj.by.play[[1]]` should contain the result of calling this function on the lines for the first play, `shakespeare.wordobj.by.play[[2]]` should contain the result of calling this function on the lines for the second play, and so on.

```r
shakespeare.wordobj.by.play <- lapply(shakespeare.lines.by.play, FUN = get.wordtab.from.lines)
```

- **5c.** Using one of the apply functions, compute numeric vectors `shakespeare.total.words.by.play` and `shakespeare.unique.words.by.play`, that contain the number of total words and number of unique words, respectively, for each of Shakespeare's plays. Each vector should only require one line of code to compute. Hint: `"[["()` is actually a function that allows you to do extract a named component of a list; e.g., try `"[["(shakespeare.wordobj, "number.total.words")`, and you'll see this is the same as `shakespeare.wordobj[["number.total.words"]]`; you should take advantage of this functionality in your apply call. What are the 5 longest plays, in terms of total word count? The 5 shortest plays?

```
shakespeare.total.words.by.play <- lapply(shakespeare.wordobj.by.play, "[[", "number.total.words")
names(shakespeare.total.words.by.play) <- titles
shakespeare.unique.words.by.play <- lapply(shakespeare.wordobj.by.play, "[[", "number.unique.words")
names(shakespeare.unique.words.by.play) <- titles
shakespeare.total.words.by.play[order(unlist(shakespeare.total.words.by.play))][40:44]
```

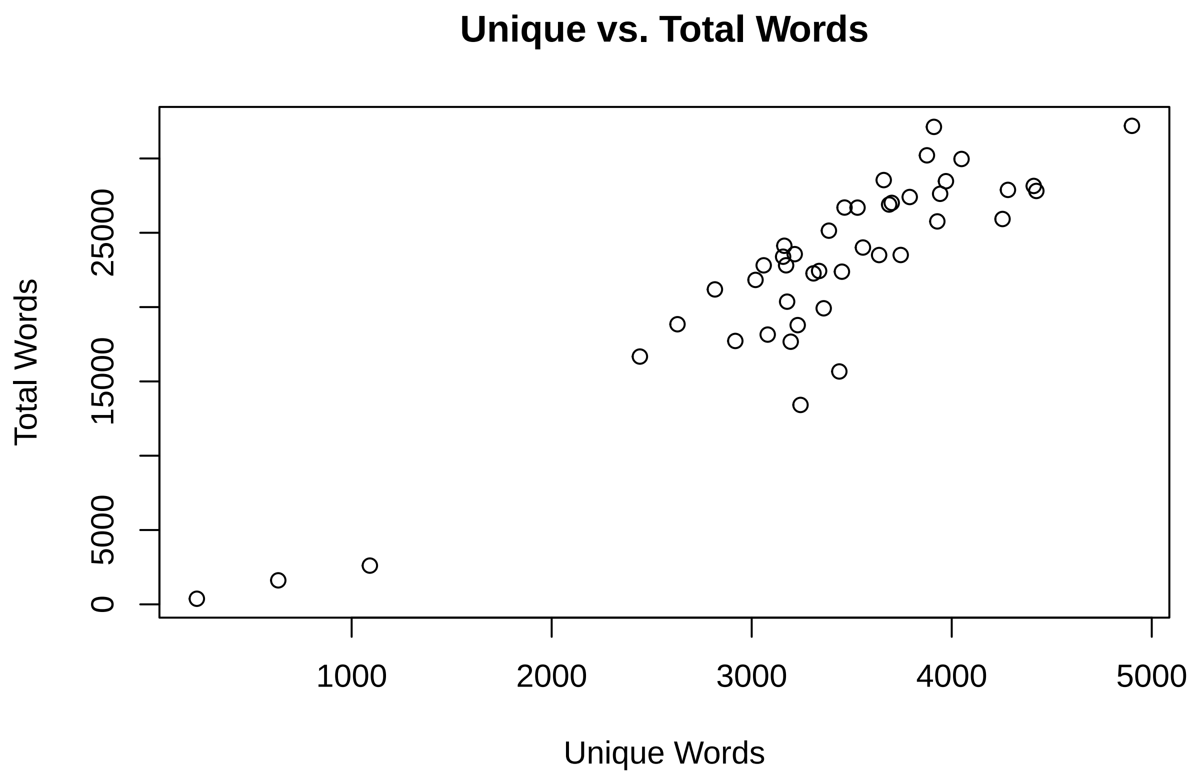
```
## $`THE TRAGEDY OF OTHELLO, MOOR OF VENICE`
## [1] 28545
##
## $CYMBELINE
## [1] 29963
##
## $`THE TRAGEDY OF CORIOLANUS`
## [1] 30210
##
## $`KING RICHARD THE THIRD`
## [1] 32123
##
## $`THE TRAGEDY OF HAMLET, PRINCE OF DENMARK`
## [1] 32193
```

```
shakespeare.total.words.by.play[order(unlist(shakespeare.total.words.by.play))][1:5]
```

```
## $`THE PHOENIX AND THE TURTLE`
## [1] 378
##
## $`THE PASSIONATE PILGRIM`
## [1] 1616
##
## $`A LOVERâ\200\231S COMPLAINT`
## [1] 2607
##
## $`VENUS AND ADONIS`
## [1] 13416
##
## $`THE RAPE OF LUCRECE`
## [1] 15668
```

- **5d.** Plot the number of unique words versus number of total words, across Shakeapeare's plays. Set the title and label the axes appropriately. Is there a consistent trend you notice?

```
plot(shakespeare.unique.words.by.play, shakespeare.total.words.by.play,
     main = "Unique vs. Total Words", xlab="Unique Words", ylab="Total Words")
```



There appears to be a constant, linear trend between unique and total words.

Refactoring the word table functions

- **6.** Look back at `get.wordtab.from.lines()` and `get.wordtab.from.url()`. Note that they overlap heavily, i.e., their bodies contain a lot of the same code. Redefine `get.wordtab.from.url()` so that it just calls `get.wordtab.from.lines()` in its body. Your new `get.wordtab.from.url()` function should have the same inputs as before, and produce the same output as before. So externally, nothing will have changed; we are just changing the internal structure of `get.wordtab.from.url()` to clean up our code base (specifically, to avoid code duplication in our case). This is an example of **code refactoring**.

  Call your new `get.wordtab.from.url()` function on the URL for Shakespeare's complete works, saving the result as `shakespeare.wordobj2`. Compare some of the components of `shakespeare.wordobj2` to those of `shakespeare.wordobj` (which was computed using the old function definition) to check that your new implementation works as it should.

```
get.wordtab.from.url <- function(str.url){
  return(get.wordtab.from.lines(readLines(str.url)))
}

shakespeare.wordobj2 <- get.wordtab.from.url("http://www.stat.cmu.edu/~ryantibs/statcomp/data/shakespea
```

- **Challenge.** Check using `all.equal()` whether `shakespeare.wordobj` and `shakespeare.wordobj2` are the same. Likely, this will not return `TRUE`. (If it does, then you've already solved this challenge question!) Modify your `get.wordtab.from.url()` function from the last question, so that it still calls get.wordtab.from.lines() to do the hard work, but produces an output exactly the same as the original shakespeare.wordobj object. Demonstrate your success by calling all.equal() once again.

```
# New function to also return lines
get.wordtab.from.lines = function(lines, split="[[:space:]]|[[:punct:]]",
                                  tolower=TRUE, keep.nums=FALSE) {
  text = paste(lines, collapse=" ")
  lines = lines[lines != ""]
  words = strsplit(text, split=split)[[1]]
  words = words[words != ""]

  # Convert to lower case, if we're asked to
  if (tolower) words = tolower(words)

  # Get rid of words with numbers, if we're asked to
  if (!keep.nums)
    words = grep("[0-9]", words, inv=TRUE, val=TRUE)

  # Compute the word table
  wordtab = table(words)

  return(list(wordtab=wordtab,
              lines = lines,
              words = words,
              number.unique.words=length(wordtab),
              number.total.words=sum(wordtab),
              longest.word=words[which.max(nchar(words))]))
}

get.wordtab.from.url <- function(str.url){
  return(get.wordtab.from.lines(readLines(str.url)))
}

shakespeare.wordobj2 <- get.wordtab.from.url("http://www.stat.cmu.edu/~ryantibs/statcomp/data/shakespea

all.equal(shakespeare.wordobj, shakespeare.wordobj2)
```

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

get.wordtab.from.lines did not return lines, so the output object was not originally equal.