Assignment 2

Saurabh Yelne

9/23/2018

Exercise 5.1

a. Create a list that contains, in this order, a sequence of 20 evenly spaced numbers between –4 and 4; a 3 × 3 matrix of the logical vector c(F,T,T,T,F,T,T,F,F) filled column-wise; a character vector with the two strings “don” and “quixote”; and a factor vector con- taining the observations c(“LOW”,“MED”,“LOW”,“MED”,“MED”,“HIGH”). Then, do the following:

```
foo <- list(seq(from=-
4,to=4,length=20),matrix(c(F,T,T,T,F,T,T,F,F),nrow=3,ncol=3),c("don","quixote
"),factor(x=c("LOW","MED","LOW","MED","MED","HIGH")))
foo
```

```
## [[1]]
##  [1] -4.0000000 -3.5789474 -3.1578947 -2.7368421 -2.3157895 -1.8947368
##  [7] -1.4736842 -1.0526316 -0.6315789 -0.2105263  0.2105263  0.6315789
## [13]  1.0526316  1.4736842  1.8947368  2.3157895  2.7368421  3.1578947
## [19]  3.5789474  4.0000000
##
## [[2]]
##       [,1]  [,2]  [,3]
## [1,] FALSE  TRUE  TRUE
## [2,]  TRUE FALSE FALSE
## [3,]  TRUE  TRUE FALSE
##
## [[3]]
## [1] "don"     "quixote"
##
## [[4]]
## [1] LOW  MED  LOW  MED  MED  HIGH
## Levels: HIGH LOW MED
```

i. Extract row elements 2 and 1 of columns 2 and 3, in that order, of the logical matrix

```
foo[[2]][2:1,2:3]
```

```
##       [,1]  [,2]
## [1,] FALSE FALSE
## [2,]  TRUE  TRUE
```

ii. Use sub to overwrite “quixote” with “Quixote” and “don” with “Don” inside the list. Then, using the newly overwritten list member, concatenate to the console screen the following statement exactly: “Windmills! ATTACK!” -Quixote/-

```
foo[[3]][1] <- sub(pattern="d",replacement="D",x=foo[[3]][1])
foo[[3]][2] <- sub(pattern="q",replacement="Q",x=foo[[3]][2])

cat(" ","\"Windmills! ATTACK!\"\n\"-\\",foo[[3]][1]," ",foo[[3]][2],"/-
",sep="")

##  "Windmills! ATTACK!"
## "-\Don Quixote/-
```

iii. Obtain all values from the sequence between −4 and 4 that are greater than 1

```
foo[[1]][foo[[1]]>1]

## [1] 1.052632 1.473684 1.894737 2.315789 2.736842 3.157895 3.578947
4.000000
```

iv. Using which, determine which indexes in the factor vector are assigned the “MED” level

```
which(x=foo[[4]]=="MED")

## [1] 2 4 5
```

b Create a new list with the factor vector from (a) as a component named “facs”; the numeric vector c(3,2.1,3.3,4,1.5,4.9) as a component named “nums”; and a nested list comprised of the first three members of the list from (a) (use list slicing to obtain this), named “oldlist”. Then, do the following:

```
bar <- list(facs=foo[[4]],nums=c(3,2.1,3.3,4,1.5,4.9),oldlist=foo[1:3])
bar

## $facs
## [1] LOW  MED  LOW  MED  MED  HIGH
## Levels: HIGH LOW MED
##
## $nums
## [1] 3.0 2.1 3.3 4.0 1.5 4.9
##
## $oldlist
## $oldlist[[1]]
##  [1] -4.0000000 -3.5789474 -3.1578947 -2.7368421 -2.3157895 -1.8947368
##  [7] -1.4736842 -1.0526316 -0.6315789 -0.2105263  0.2105263  0.6315789
## [13]  1.0526316  1.4736842  1.8947368  2.3157895  2.7368421  3.1578947
## [19]  3.5789474  4.0000000
##
## $oldlist[[2]]
##       [,1]  [,2]  [,3]
```

```
## [1,] FALSE  TRUE  TRUE
## [2,]  TRUE FALSE FALSE
## [3,]  TRUE  TRUE FALSE
##
## $oldlist[[3]]
## [1] "Don"     "Quixote"
```

i. Extract the elements of "facs" that correspond to elements of "nums" that are greater than or equal to 3

```
bar$facs[bar$nums>=3]

## [1] LOW  LOW  MED  HIGH
## Levels: HIGH LOW MED
```

ii. Add a new member to the list named "flags". This member should be a logical vector of length 6, obtained as a twofold repetition of the third column of the logical matrix in the "oldlist" component

```
bar$flags <- rep(x=bar$oldlist[[2]][,3],times=2)
bar$flags

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

iii. Use "flags" and the logical negation operator ! to extract the entries of "num" corresponding to FALSE

```
which(bar$flags != TRUE)

## [1] 2 3 5 6
```

iv. Overwrite the character string vector component of "oldlist" with the single character string "Don Quixote"

```
bar$oldlist[[3]] <- "Don Quixote"
bar$oldlist[[3]]

## [1] "Don Quixote"
```

Exercise 10.1

a. Create the following two vectors:

```
vec1 <- c(2,1,1,3,2,1,0)
vec2 <- c(3,8,2,2,0,0,0)
```

Without executing them, determine which of the following if statements would result in the string being printed to the console. Then confirm your answers in R.

i. if((vec1[1]+vec2[2])==10){ cat("Print me!") }

Ans. TRUE, this will print the "Print me!" to the console as 2+8 equals 10

ii. if(vec1[1]>=2&&vec2[1]>=2){ cat("Print me!") }

Ans. TRUE, this will print the "Print me!" to the console as both the conditons are true

iii. if(all((vec2-vec1)[c(2,6)]<7)){ cat("Print me!") }

Ans. FALSE, this will not print the "Print me!" to the console as the conditon used with all are false

iv. if(!is.na(vec2[3])){ cat("Print me!") }

Ans. TRUE, this will print the "Print me!" to the console as the 3rd element in vec2 is not NA

```
if((vec1[1]+vec2[2])==10){ cat("Print me1!") }

## Print me1!

if(all((vec2-vec1)[c(2,6)]<7)){ cat("Print me2!") }
if(!is.na(vec2[3])){ cat("Print me3!") }

## Print me3!

if(!is.na(vec2[3])){ cat("Print me4!") }

## Print me4!
```

So statements 1,3 and 4 are true

b. Using vec1 and vec2 from (a), write and execute a line of code that multiplies the corresponding elements of the two vectors together if their sum is greater than 3. Otherwise, the code should simply sum the two elements.

```
vec1

## [1] 2 1 1 3 2 1 0

vec2

## [1] 3 8 2 2 0 0 0

ifelse((vec1+vec2)>3,vec1 * vec2,vec1+vec2)

## [1] 6 8 3 6 2 1 0
```

c. In the editor, write R code that takes a square character matrix and checks if any of the character strings on the diagonal (top left to bottom right) begin with the letter g, lowercase or upper- case. If satisfied, these specific entries should be overwritten with the string "HERE". Otherwise, the entire matrix should be replaced with an identity matrix of the same dimensions. Then, try your code on the following matrices, checking the result each time:

i. mymat <- matrix(as.character(1:16),4,4)

```
mymat <- matrix(as.character(1:16),4,4)
if(any(substr(diag(mymat),1,1)=="g")|any(substr(diag(mymat),1,1)=="G")){
  i <- which(substr(diag(mymat),1,1)=="g"|substr(diag(mymat),1,1)=="G")
  diag(mymat)[i] <- "HERE"
} else {
  mymat <- diag(nrow(mymat))
}
mymat
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    0
## [2,]    0    1    0    0
## [3,]    0    0    1    0
## [4,]    0    0    0    1
```

ii. mymat <- matrix(c("DANDELION","Hyacinthus","Gerbera", "MARIGOLD","geranium","ligularia", "Pachysandra","SNAPDRAGON","GLADIOLUS"),3,3)

```
mymat <-
matrix(c("DANDELION","Hyacinthus","Gerbera","MARIGOLD","geranium","ligulria",
"Pachysandra","SNAPDRAGON","GLADIOLUS"),3,3)

if(any(substr(diag(mymat),1,1)=="g")|any(substr(diag(mymat),1,1)=="G")){
  i <- which(substr(diag(mymat),1,1)=="g"|substr(diag(mymat),1,1)=="G")
  diag(mymat)[i] <- "HERE"
} else {
  mymat <- diag(nrow(mymat))
}
mymat
```

```
##      [,1]         [,2]       [,3]
## [1,] "DANDELION"  "MARIGOLD" "Pachysandra"
## [2,] "Hyacinthus" "HERE"     "SNAPDRAGON"
## [3,] "Gerbera"    "ligulria" "HERE"
```

iii. mymat <- matrix(c("GREAT","exercises","right","here"),2,2, byrow=T) Hint: This requires some thought—you will find the functions diag from Section 3.2.1 and substr from Section 4.2.4 useful

```
mymat <- matrix(c("GREAT","exercises","right","here"),2,2, byrow=T)
if(any(substr(diag(mymat),1,1)=="g")|any(substr(diag(mymat),1,1)=="G")){
  i <- which(substr(diag(mymat),1,1)=="g"|substr(diag(mymat),1,1)=="G")
  diag(mymat)[i] <- "HERE"
} else {
  mymat <- diag(nrow(mymat))
}
mymat
```

```
##      [,1]    [,2]
## [1,] "HERE"  "exercises"
## [2,] "right" "here"
```

Exercise 10.2

a. Write an explicit stacked set of if statements that does the same thing as the integer version of the switch function illustrated earlier. Test it with mynum <- 3 and mynum <- 0, as in the text

```
mynum <- 3
if (mynum==1){
  foo <- 12
} else if(mynum==2){
  foo <- 34
} else if (mynum==3){
  foo <- 56
} else if (mynum==4){
  foo <- 78
} else{
  foo <- NA
}
foo
```

```
## [1] 56
```

```
mynum <- 0
if (mynum==1){
  foo <- 12
} else if(mynum==2){
  foo <- 34
} else if (mynum==3){
  foo <- 56
} else if (mynum==4){
  foo <- 78
} else{
  foo <- NA
}
foo
```

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

b. Suppose you are tasked with computing the precise dosage amounts of a certain drug in a collection of hypothetical scientific experiments. These amounts depend upon some pre-determined set of "dosage thresholds" (lowdose, meddose, and highdose), as well as a predetermined dose level factor vector named doselevel. Look at the following items (i–iv) to see the intended form of these objects. Then write a set of nested if statements that produce a new numeric vector called dosage, according to the following rules:

– First, if there are any instances of "High" in doselevel, perform the following operations: Check if lowdose is greater than or equal to 10. If so, overwrite lowdose with 10; otherwise, overwrite lowdose by itself divided by 2. Check if meddose is greater than or equal to 26. If so, overwrite meddose by 26. Check if highdose is less than 60. If so, overwrite highdose with 60; otherwise, overwrite highdose by itself multiplied by 1.5. Create a vector named dosage with the value of lowdose repeated (rep) to match the length of doselevel. Overwrite the elements in dosage corresponding to the index positions of instances of "Med" in doselevel by meddose. Overwrite the elements in dosage corresponding to the index positions of instances of "High" in doselevel by highdose. – Otherwise (in other words, if there are no instances of "High" in doselevel), perform the following operations:

Create a new version of doselevel, a factor vector with levels "Low" and "Med" only, and label these with "Small" and "Large", respectively (refer to Section 4.3 for details or see ?factor)

Check to see if lowdose is less than 15 AND meddose is less than 35. If so, overwrite lowdose by itself multiplied by 2 and overwrite meddose by itself plus highdose.

Create a vector named dosage, which is the value of lowdose repeated (rep) to match the length of doselevel. Overwrite the elements in dosage corresponding to the index positions of instances of "Large" in doselevel by meddose.

Now, confirm the following:

i. Given lowdose <- 12.5 meddose <- 25.3 highdose <- 58.1 doselevel <- factor(c("Low","High","High","High","Low","Med", "Med"),levels=c("Low","Med","High")) the result of dosage after running the nested if statements is as follows:

R> dosage [1] 10.0 60.0 60.0 60.0 10.0 25.3 25.3

```
lowdose <- 12.5
meddose <- 25.3
highdose <- 58.1
doselevel <-
```

```
factor(c("Low","High","High","High","Low","Med","Med"),levels=c("Low","Med","
High"))


if(any(doselevel=="High")){
  if(lowdose>=10){
    lowdose <- 10
  } else {
    lowdose <- lowdose/2
  }
  if(meddose>=26){
    meddose <- 26
  }
  if(highdose<60){
    highdose <- 60
  } else {
    highdose <- highdose*1.5
  }
  dosage <- rep(lowdose,length(doselevel))
  dosage[doselevel=="Med"] <- meddose
  dosage[doselevel=="High"] <- highdose
} else {
  doselevel <-
factor(doselevel,levels=c("Low","Med"),labels=c("Small","Large"))
  if(lowdose<15 && meddose<35){
    lowdose <- lowdose*2
    meddose <- meddose+highdose
  }
  dosage <- rep(lowdose,length(doselevel))
  dosage[doselevel=="Large"] <- meddose
}
dosage

## [1] 10.0 60.0 60.0 60.0 10.0 25.3 25.3
```

ii. Using the same lowdose, meddose, and highdose thresholds as in (i), given doselevel <- factor(c("Low","Low","Low","Med","Low","Med", "Med"),levels=c("Low","Med","High")) the result of dosage after running the nested if statements is as follows: R> dosage [1] 25.0 25.0 25.0 83.4 25.0 83.4 83.4 Also, doselevel has been overwritten as follows: R> doselevel [1] Small Small Small Large Small Large Large Levels: Small Large

```
lowdose <- 12.5
meddose <- 25.3
highdose <- 58.1
doselevel <-
factor(c("Low","Low","Low","Med","Low","Med","Med"),levels=c("Low","Med","Hig
h"))

if(any(doselevel=="High")){
```

```
  if(lowdose>=10){
    lowdose <- 10
  } else {
    lowdose <- lowdose/2
  }
  if(meddose>=26){
    meddose <- 26
  }
  if(highdose<60){
    highdose <- 60
  } else {
    highdose <- highdose*1.5
  }
  dosage <- rep(lowdose,length(doselevel))
  dosage[doselevel=="Med"] <- meddose
  dosage[doselevel=="High"] <- highdose
} else {
  doselevel <-
factor(doselevel,levels=c("Low","Med"),labels=c("Small","Large"))
  if(lowdose<15 && meddose<35){
    lowdose <- lowdose*2
    meddose <- meddose+highdose
  }
  dosage <- rep(lowdose,length(doselevel))
  dosage[doselevel=="Large"] <- meddose
}
dosage

## [1] 25.0 25.0 25.0 83.4 25.0 83.4 83.4

doselevel

## [1] Small Small Small Large Small Large Large
## Levels: Small Large
```

iii. Given lowdose <- 9 meddose <- 49 highdose <- 61 doselevel <- factor(c("Low","Med","Med"), levels=c("Low","Med","High")) the result of dosage after running the nested if statements is as follows: R> dosage [1] 9 49 49 Also, doselevel has been overwritten as follows: R> doselevel [1] Small Large Large Levels: Small Large

```
lowdose <- 9
meddose <- 49
highdose <- 61
doselevel <- factor(c("Low","Med","Med"),levels=c("Low","Med","High"))

if(any(doselevel=="High")){
  if(lowdose>=10){
    lowdose <- 10
  } else {
    lowdose <- lowdose/2
```

```
  }
  if(meddose>=26){
    meddose <- 26
  }
  if(highdose<60){
    highdose <- 60
  } else {
    highdose <- highdose*1.5
  }
  dosage <- rep(lowdose,length(doselevel))
  dosage[doselevel=="Med"] <- meddose
  dosage[doselevel=="High"] <- highdose
} else {
  doselevel <-
factor(doselevel,levels=c("Low","Med"),labels=c("Small","Large"))
  if(lowdose<15 && meddose<35){
    lowdose <- lowdose*2
    meddose <- meddose+highdose
  }
  dosage <- rep(lowdose,length(doselevel))
  dosage[doselevel=="Large"] <- meddose
}

dosage

## [1]  9 49 49

doselevel

## [1] Small Large Large
## Levels: Small Large
```

iv. Using the same lowdose, meddose, and highdose thresholds as (iii), as well as the same doselevel as (i), the result of dosage after running the nested if statements is as follows: R> dosage [1] 4.5 91.5 91.5 91.5 4.5 26.0 26.0

```
lowdose <- 9
meddose <- 49
highdose <- 61
doselevel <-
factor(c("Low","High","High","High","Low","Med","Med"),levels=c("Low","Med","
High"))
if(any(doselevel=="High")){
  if(lowdose>=10){
    lowdose <- 10
  } else {
    lowdose <- lowdose/2
  }
  if(meddose>=26){
    meddose <- 26
  }
```

```
  if(highdose<60){
    highdose <- 60
  } else {
    highdose <- highdose*1.5
  }
  dosage <- rep(lowdose,length(doselevel))
  dosage[doselevel=="Med"] <- meddose
  dosage[doselevel=="High"] <- highdose
} else {
  doselevel <-
factor(doselevel,levels=c("Low","Med"),labels=c("Small","Large"))
  if(lowdose<15 && meddose<35){
    lowdose <- lowdose*2
    meddose <- meddose+highdose
  }
  dosage <- rep(lowdose,length(doselevel))
  dosage[doselevel=="Large"] <- meddose
}

dosage
```

```
## [1]  4.5 91.5 91.5 91.5  4.5 26.0 26.0
```

c. Assume the object mynum will only ever be a single integer between 0 and 9. Use ifelse and switch to produce a command that takes in mynum and returns a matching character string for all possible values 0, 1, . . . , 9. Supplied with 3, for example, it should return “three”; supplied with 0, it should return “zero”

```
mynum <- 75
ifelse(mynum >= 0 & mynum <= 9,
       foo <- switch(as.character(mynum), "0"="zero",
                "1"="one", "2"="two","3"="three",
"4"="four","5"="five","6"="six","7"="seven","8"="eight","9"="nine"), foo <-
"0")
```

```
## [1] "0"
```

Exercise 10.3

a. In the interests of efficient coding, rewrite the nested loop example from this section, where the matrix foo was filled with the multiples of the elements of loopvec1 and loopvec2, using only a single for loop

```
loopvec1 <- 5:7
loopvec2 <- 9:6
```

```
foo <- matrix(NA,length(loopvec1),length(loopvec2))
for(i in 1:length(loopvec1)) for (j in 1:length(loopvec2)) {
  foo[i,j] <- loopvec1[i]*loopvec2[j]
}
foo

##      [,1] [,2] [,3] [,4]
## [1,]   45   40   35   30
## [2,]   54   48   42   36
## [3,]   63   56   49   42
```

b. In Section 10.1.5, you used the command switch(EXPR=mystring,Homer=12,Marge=34,Bart=56,Lisa=78,Maggie=90, NA) to return a number based on the supplied value of a single character string. This line won't work if mystring is a character vector. Write some code that will take a character vector and return a vector of the appropriate numeric values. Test it on the following vector: c("Peter","Homer","Lois","Stewie","Maggie","Bart")

```
mystring<-c("Peter","Homer","Lois","Stewie","Maggie","Bart")
out<- rep(NA, length(mystring))
for(i in seq_along(mystring)){
  out[i]<-
switch(EXPR=mystring[i],Homer=12,Marge=34,Bart=56,Lisa=78,Maggie=90, NA)
}
out

## [1] NA 12 NA NA 90 56
```

c. Suppose you have a list named mylist that can contain other lists as members, but assume those "member lists" cannot themselves contain lists. Write nested loops that can search any possible mylist defined in this way and count how many matrices are present. Hint: Simply set up a counter before commencing the loops that is incremented each time a matrix is found, regardless of whether it is a straightforward member of mylist or it is a member of a member list of mylist.

i. That the answer is 4 if you have the following: mylist <- list(aa=c(3.4,1),bb=matrix(1:4,2,2), cc=matrix(c(T,T,F,T,F,F),3,2),dd="string here", ee=list(c("hello","you"),matrix(c("hello", "there"))), ff=matrix(c("red","green","blue","yellow")))

```
mylist <- list(aa=c(3.4,1),bb=matrix(1:4,2,2),
cc=matrix(c(T,T,F,T,F,F),3,2),dd="string here",
```

```
ee=list(c("hello","you"),matrix(c("hello", "there"))),
ff=matrix(c("red","green","blue","yellow")))

counter <- 0
for(i in 1:length(mylist)){
  member <- mylist[[i]]
  if(is.matrix(member)){
    counter <- counter+1
  } else if(is.list(member)){
    for(j in 1:length(member)){
      if(is.matrix(member[[j]])){
        counter <- counter+1
      }
    }
  }
}
counter

## [1] 4
```

ii. That the answer is 0 if you have the following: mylist <- list("tricked you",as.vector(matrix(1:6,3,2)))

```
mylist <- list("tricked you",as.vector(matrix(1:6,3,2)))

counter <- 0
for(i in 1:length(mylist)){
  member <- mylist[[i]]
  if(is.matrix(member)){
    counter <- counter+1
  } else if(is.list(member)){
    for(j in 1:length(member)){
      if(is.matrix(member[[j]])){
        counter <- counter+1
      }
    }
  }
}
counter

## [1] 0
```

iii. That the answer is 2 if you have the following: mylist <- list(list(1,2,3),list(c(3,2),2), list(c(1,2),matrix(c(1,2))), rbind(1:10,100:91))

```
mylist <- list(list(1,2,3),list(c(3,2),2), list(c(1,2),matrix(c(1,2))),
          rbind(1:10,100:91))

counter <- 0
for(i in 1:length(mylist)){
  member <- mylist[[i]]
  if(is.matrix(member)){
```

```
    counter <- counter+1
  } else if(is.list(member)){
    for(j in 1:length(member)){
      if(is.matrix(member[[j]])){
        counter <- counter+1
      }
    }
  }
}
counter
```

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

Exercise 10.4

a. Based on the most recent example of storing identity matrices in a list, determine what the resulting mylist would look like for each of the following possible mynumbers vectors, without executing anything:

i. mynumbers <- c(2,2,2,2,5,2)

Ans. The list will contain the identity matrix corresponding to all the element in the mynumbers vector as the condition will be true for all the elements

```
mylist <- list()
counter <- 1
mynumbers <- c(2,2,2,2,5,2)
mycondition <- mynumbers[counter]<=5

while(mycondition){
  mylist[[counter]] <- diag(mynumbers[counter])
  counter <- counter+1
  if(counter<=length(mynumbers)){
    mycondition <- mynumbers[counter]<=5
    }else {
      mycondition <- FALSE
    }
}
mylist
```

```
## [[1]]
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
##
## [[2]]
##      [,1] [,2]
```

```
## [1,]    1    0
## [2,]    0    1
##
## [[3]]
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
##
## [[4]]
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
##
## [[5]]
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    0    0    0    0
## [2,]    0    1    0    0    0
## [3,]    0    0    1    0    0
## [4,]    0    0    0    1    0
## [5,]    0    0    0    0    1
##
## [[6]]
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

ii. mynumbers <- 2:20

Ans. The list will contain the identity matrix of first 4 elements namely 2,3,4 and 5

```
mylist <- list()
counter <- 1
mynumbers <- 2:20
mycondition <- mynumbers[counter]<=5

while(mycondition){
  mylist[[counter]] <- diag(mynumbers[counter])
  counter <- counter+1
  if(counter<=length(mynumbers)){
    mycondition <- mynumbers[counter]<=5
    }else {
      mycondition <- FALSE
    }
}
mylist

## [[1]]
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
##
```

```
## [[2]]
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
##
## [[3]]
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    0
## [2,]    0    1    0    0
## [3,]    0    0    1    0
## [4,]    0    0    0    1
##
## [[4]]
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    0    0    0    0
## [2,]    0    1    0    0    0
## [3,]    0    0    1    0    0
## [4,]    0    0    0    1    0
## [5,]    0    0    0    0    1
```

iii. mynumbers <- c(10,1,10,1,2)

Ans. The list will be empty as first element gives a FALSE condition and the while loop is never executed

```
mylist <- list()
counter <- 1
mynumbers <- c(10,1,10,1,2)
mycondition <- mynumbers[counter]<=5

while(mycondition){
  mylist[[counter]] <- diag(mynumbers[counter])
  counter <- counter+1
  if(counter<=length(mynumbers)){
    mycondition <- mynumbers[counter]<=5
    }else {
      mycondition <- FALSE
    }
}
mylist

## list()
```

Then, confirm your answers in R (note you'll also have to reset the initial values of mylist, counter, and mycondition each time, just as in the text).

b. Write a while loop that computes and stores as a new object the factorial of any non-negative integer mynum by decrementing mynum by 1 at each repetition of the braced code

Using your loop, confirm the following:

i. That the result of using mynum <- 5 is 120

```
fac <- 5
n <- fac-1
while(n>=1){
  fac<-fac*n
   n <- n-1
}
if(fac<=1 && fac==1||fac==0){
  fac <- 1
}
print(fac)
```

```
## [1] 120
```

ii. That using mynum <- 12 yields 479001600

```
fac<- 12
n <- fac-1
while(n>=1){
  fac<-fac*n
   n <- n-1
}
if(fac<=1 && fac==1||fac==0){
  fac <- 1
}
print(fac)
```

```
## [1] 479001600
```

iii. That having mynum <- 0 correctly returns 1

```
fac <- 0
n <- fac-1
while(n>=1){
  fac<-fac*n
   n <- n-1
}
if(fac<=1 && fac==1||fac==0){
  fac <- 1
}

print(fac)
```

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

c. Consider the following code, where the operations in the braced area of the while loop have been omitted:

mystring <- “R fever” index <- 1 ecount <- 0 result <- mystring while(ecount<2 && index<=nchar(mystring)){ $# several omitted operations #$ } result Your task is to complete the code in the braced area so it inspects mystring character by character until it reaches the second instance of the letter e or the end of the string, whichever comes first. The result object should be the entire character string if there is no second e or the character string made up of all the characters up to, but not including, the sec- ond e if there is one. For example, mystring <- “R fever” should provide result as “R fev”. This must be achieved by following these operations in the braces:

1. Use substr (Section 4.2.4) to extract the single character of mystring at position index.
2. Use a check for equality to determine whether this single- character string is either “e” OR “E”. If so, increase ecount by 1.
3. Next, perform a separate check to see whether ecount is equal to 2. If so, use substr to set result equal to the characters between 1 and index-1 inclusive.
4. Increment index by 1. Test your code—ensure the previous result for mystring <- “R fever”.

```
mystring <- "R fever"
index <- 1
ecount <- 0
result <- mystring
while(ecount<2 && index<=nchar(mystring)){
  store_e <- substr(mystring,index,index)
  if(store_e=="e"||store_e=="E"){
    ecount <- ecount+1
  }
  if(ecount==2){
    result <- substr(mystring,1,index-1)
  }
  index <- index + 1
}
result

## [1] "R fev"
```

Furthermore, confirm the following: – – – – Using mystring <- “beautiful” provides result as “beautiful”

```
mystring <- "beautiful"
index <- 1
ecount <- 0
result <- mystring
while(ecount<2 && index<=nchar(mystring)){
  store_e <- substr(mystring,index,index)
  if(store_e=="e"||store_e=="E"){
    ecount <- ecount+1
```

```
  }
  if(ecount==2){
    result <- substr(mystring,1,index-1)
  }
  index <- index + 1
}
result
```

```
## [1] "beautiful"
```

Using mystring <- “ECCENTRIC” provides result as “ECC”

```
mystring <- "ECCENTRIC"
index <- 1
ecount <- 0
result <- mystring
while(ecount<2 && index<=nchar(mystring)){
  store_e <- substr(mystring,index,index)
  if(store_e=="e"||store_e=="E"){
    ecount <- ecount+1
  }
  if(ecount==2){
    result <- substr(mystring,1,index-1)
  }
  index <- index + 1
}
result
```

```
## [1] "ECC"
```

Using mystring <- “ElAbOrAte” provides result as “ElAbOrAt”

```
mystring <- "ElAbOrAte"
index <- 1
ecount <- 0
result <- mystring
while(ecount<2 && index<=nchar(mystring)){
  store_e <- substr(mystring,index,index)
  if(store_e=="e"||store_e=="E"){
    ecount <- ecount+1
  }
  if(ecount==2){
    result <- substr(mystring,1,index-1)
  }
  index <- index + 1
}
result
```

```
## [1] "ElAbOrAt"
```

Using mystring <- “eeeeek!” provides result as “e”

```
mystring <- "eeeeek!"
index <- 1
ecount <- 0
result <- mystring
while(ecount<2 && index<=nchar(mystring)){
  store_e <- substr(mystring,index,index)
  if(store_e=="e"||store_e=="E"){
    ecount <- ecount+1
  }
  if(ecount==2){
    result <- substr(mystring,1,index-1)
  }
  index <- index + 1
}
result
```

```
## [1] "e"
```

Exercise 10.5

a. Continuing on from the most recent example in the text, write an implicit loop that calculates the product of all the column elements of the matrix returned by the call to apply(foo,1,sort,decreasing=TRUE)

```
foo <- matrix(1:12,4,3)
apply(apply(foo,1,sort,decreasing=TRUE),2,prod)
```

```
## [1]  45 120 231 384
```

```
foo
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

b. Convert the following for loop to an implicit loop that does exactly the same thing:

matlist <- list(matrix(c(T,F,T,T),2,2), matrix(c("a","c","b","z","p","q"),3,2), matrix(1:8,2,4))
matlist for(i in 1:length(matlist)){ matlist[[i]] <- t(matlist[[i]]) } matlist

```
matlist <- list(matrix(c(T,F,T,T),2,2),
matrix(c("a","c","b","z","p","q"),3,2),
matrix(1:8,2,4))
matlist<-lapply(matlist, t)
matlist
```

```
## [[1]]
##      [,1]  [,2]
## [1,] TRUE FALSE
## [2,] TRUE  TRUE
##
## [[2]]
##      [,1] [,2] [,3]
## [1,] "a"  "c"  "b"
## [2,] "z"  "p"  "q"
##
## [[3]]
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]    5    6
## [4,]    7    8
```