Lecture 10

Input / Output

GEOG 489

SPRING 2020

Quiz 4

Generate a vector of X that has 100 random numbers from the normal distribution (mean = 1, sd = 1). Generate a vector of Y that has 200 random numbers from the uniform distribution (min = 0, max = 2)

Then generate a list Z with two elements X and Y, and use lapply() to count the number of values between 0.5 and 1.5 for each element.

```
X <- rnorm(100, mean = 1, sd = 1)
Y <- runif(200, min = 0, max = 2)
Z <- list(X,Y)
countnumber <- function(x){

  n <- length(x[x > 0.5 & x < 1.5])
  return(n)
}
lapply(Z, countnumber)
```

Input / Output

The first step in any GIS project is often getting data into your analysis, and the last step is often going to be getting results out of your analysis.

We will talk about GIS input/output (I/O) soon, but we are going to start with the basics.

Input / Output

Prepare the text files:

1) Setting working directory

```
setwd("Z:/Teaching/GISProgramming/Test/")
```

2) Create some text files in the working directory

```
cat("123", "4 5", "6", file="z1.txt", sep="\n")

cat("123", "4.2 5", "6", file="z2.txt", sep="\n")

cat("abc", "de f", "g", file="z3.txt", sep="\n")

cat("abc", "123 6", "y", file="z4.txt", sep="\n")
```

3) Check to see if the file is in the "working directory“

```
myfiles <- dir()
myfiles
```

Input / Output

1) Scan() function – read txt file

scan() reads data from a file into a vector or list

scan() uses whitespaces as delimiting characters. A whitespace includes blanks, returns, and horizontal tabs.

```
z1 <- scan("z1.txt")                    # numeric value
z2 <- scan("z2.txt")                    # numeric value
z3 <- scan("z3.txt",what="")            # character
z4 <- scan("z4.txt",what="")            # character
```

Input / Output

1) Scan() function – read from keyboard

File name: blank value

```
v <- scan("")
# Type:
# 12 5 13
# 3 4 5
# 8
# and then hit return to end it.
v
```

Input / Output

2) readline() function – read a single line from keyboard

```
w <- readline()
# type abc de f
w

# We can use readline with a prompt so the user knows what to do:
inits <- readline("type your initials:")
# Type your initials
inits
```

Input / Output

3) read.table() function – read a file into a data frame with the proper heading

(1) read a txt file

```
# read.table assumes whitespace delimitations.
cat("name age", "John 25", "Mary 28", "Jim 19", file="ztable1.txt",
sep="\n")
z1 <- read.table("ztable1.txt",header=TRUE)
z1
class(z1)
names(z1)
```

Input / Output

3) read.table() function – read a file into a data frame with the proper heading

(1) read a csv file

```
# Use comma delimitation to read csv file

z2 <- read.table("ztable2.csv",sep=",",header=TRUE)

# R conveniently provides a "wrapper" for CSV files to make this a bit easier:

z2 <- read.csv("ztable2.csv") # Notice that it assumes header=TRUE
```

Input / Output

4) write.table() function – write to a file

```
z1 <- read.table("ztable1.txt",header=TRUE)

# A quick way to write this file out to a new file is:
write.table(z1, file="ztablenew.txt")

# write to a file without row or column names
write.table(z1, file="ztablenew_nocolrow.txt",
        row.names=FALSE, col.names=FALSE)
```

Input / Output

5) cat() function – print multiple objects to the screen

```
# need use a newline character (\n) to go to the next line
print("abc")
cat("abc","abc",10)

x <- 1:3
# Say we want to make a more useful statement:
cat("Vector x is",x,"\n")
# We can modify what goes between each element sent to cat:
cat("Vector x is",x,sep="\n")
```

Input / Output

5) cat() function – print multiple objects to the screen

```
x <- 1:3
# or use no space at all:
cat("Vector x is",x,"\n",sep="")
# or use exclamation:
cat("Vector x is",x,"\n",sep="!")
# or have different separators for each element:
x <- c(5,12,13,8,88)
separators <- c(".",".",".","\n","\n")
cat(x,sep=separators)
```

Input / Output

5) cat() function – write to a file

```
# Calling cat initially will create a file if it doesn't exist:
cat("def\n", file="cattest.txt") # Don't forget the newline character

# If we want to add a new line to the file, set the append parameter to TRUE:
cat("de\n", file="cattest.txt", append=TRUE)
```

Introduction to connections

To interact with files, R provides many tools for fine-tuned control, but we need to work within the concept of connections.

This allows us to read or write in pieces, rather than all-at-once.

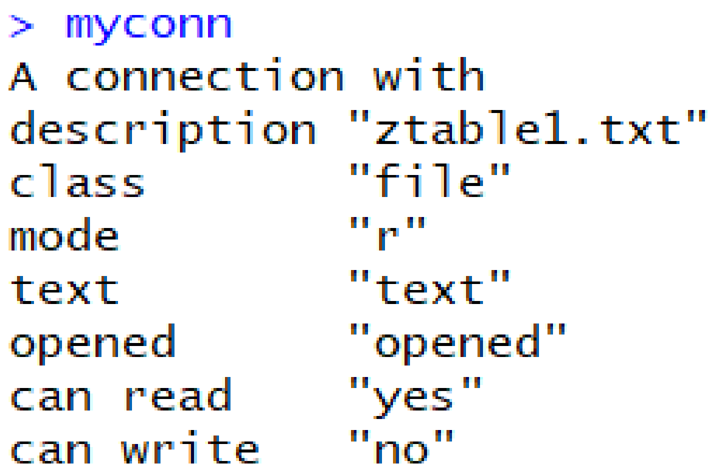
For big files, this is an important feature so we don't run out of memory

Introduction to connections

We can create a connection to a file in different ways, but a common way to connect to a file on disk is using **file():**

# open the file ztable1.txt in read-only mode ("r")

myconn <- file("ztable1.txt","r")



```
> myconn
A connection with
description "ztable1.txt"
class       "file"
mode        "r"
text        "text"
opened      "opened"
can read    "yes"
can write   "no"
```

Introduction to connections

We can create a connection to a file in different ways, but a common way to connect to a file on disk is using **file():**

# myconn is the CONNECTION to the file, not the contents. We can now use this connection to access the file

# Notice that the line that is read increments as we call the function over and over again. n=1 refers to "read one line".

readLines(myconn,n=1)

# remember to close() the connection once you are done with it

close(myconn)

Introduction to connections

We can create a connection to a file in different ways, but a common way to connect to a file on disk is using **file():**

```
# write the file

myconn <- file("writelines_test.txt","w")

# the "w" indicates the file is opened for writing.

writeLines(c("abc","de","f"),myconn)

close(myconn)
```

Access files on remote machine via URLs

Some I/O functions can directly access URLs, including read.table(), read.csv(), and scan().

```
# First pull up a web browser and confirm this URL works:

# http://archive.ics.uci.edu/ml/machine-learning-databases/echocardiogram/echocardiogram.data

uci <- "http://archive.ics.uci.edu/ml/machine-learning-databases/echocardiogram/echocardiogram.data"

ecc <- read.csv(uci)
```

Getting file and directory information

Getting file information is important to perform "batch processing" in GIS.

Briefly, batch processing is repeating the same data analysis to multiple files.

For instance, say we wanted to reproject a bunch of vector files that we've stored in a directory. We would need to know, first, the names of the vector files so we can open them, reproject them, and write the output to disk.

Getting file and directory information

1) Get the file information - file.info()

```
?file.info # Returns file information from the OS

file.info("writelines_test.txt")
```

2) List all files in a directory

```
dir()
# For batch processing, we might loop through this:
mydir <- dir()
for (i in seq(mydir))
{
    print(mydir[i])
}
```

Getting file and directory information

3) List files with certain file extension

dir(pattern=".txt")   # file with .txt extension

4) Check if a file exists

file.exists("writelines_test.txt“)

5) Get and set working directory info

# get the working directory – getwd()
getwd()
# change the working directory – setwd()
setwd()