File Handling

Writing Data

```
>
> "Hello World"
[1] "Hello World"
> 'Hello World'
[1] "Hello World"
> 12
[1] 12
> 12i + 2
[1] 2+12i
> 12L
[1] 12
> 12.222
[1] 12.222
>
>
```

```
>
>
> print("Hello WOrld")
[1] "Hello WOrld"
> print('Hello World')
[1] "Hello World"
> print(12)
[1] 12
>
>
```

Reading Data

Read a file as a vector with the scan function.

```
> numbers = scan()
1: 12
2: 233
3: 22222
4: 12.2
5: 22
6: 1234.22222
7:
Read 6 items
> numbers
[1]    12.000   233.000 22222.000    12.200    22.000  1234.222
>
```

By default, scans numeric elements: it fails if the input contains characters.

```
> class(numbers)
[1] "numeric"
>
>
```

Note: class returns the type of the variable

What happens when you provide characters in the *scan()* function?

```
> strings = scan()
1: programming
```

```
> strings = scan()
1: programming
1:
Error in scan() : scan() expected 'a real', got 'programming'
>
>
```

What should be done to read characters using scan() function?

If non-numeric, you need to specify the type of data contained in the file:

```
> strings = scan(what="character")
1: hello
2: World
3: programming
4: R
5:
Read 4 items
> strings
[1] "hello"       "World"       "programming" "R"
>
>
```

What is returned on execution of *class(strings)* in the above example?

```
>
> class(strings)
[1] "character"
>
```

Reading a File

Read a file as a vector with the scan function

```
~ [14:03:37]
0:~$cat file.txt
1
2
3
4
5
```

Read in file

```
> scan(file="file.txt")
Read 5 items
[1] 1 2 3 4 5
>
>
```

Read the file and save in a variable.

```
> k <- scan(file="file.txt")
Read 5 items
> k
[1] 1 2 3 4 5
>
>
```

What is the class of *k*?

```
> scan(file="file.txt")
Read 5 items
[1] 1 2 3 4 5
> k <- scan(file="file.txt")
Read 5 items
> class(k)
[1] "numeric"
>
```

If the file is not in the current directory, you can provide a full or relative path. For example, if located in the home directory, read it as:

```
> k = scan(file="/home/macgok/file.txt")
Read 5 items
> k
[1] 1 2 3 4 5
>
>
```

Write a program that reads a file containing strings and displays it on the console.

```
~ [09:56:52]
0:~$cat file1.txt
hello
World
```

```
> items = scan(file="file1.txt", what="character")
Read 2 items
> items
[1] "hello" "World"
>
```

Writing to a file

Write the content of a vector in a file using the write function.

```
>
> mygenes <- c("SMAD4", "DKK1", "ASXL3", "ERG", "CKLF", "TIAM1", "VHL", "BTD", "EMP1", "MALL", "PAX3")
> write(x=mygenes, file="gene_list.txt")
>
> q()
Save workspace image? [y/n/c]: n

~ [10:02:15]
0:~$cat gene_list.txt
SMAD4
DKK1
ASXL3
ERG
CKLF
TIAM1
VHL
BTD
EMP1
MALL
PAX3

~ [10:02:27]
0:~$
```

CSV

R can read and write into various file formats like csv, excel, xml etc.

Read in a file into a data frame with the read.table function:

```
~/R_Samples [14:46:54]
0:~$cat file.csv
column1, column2, column3
1,2.22,abcd
2,3.33,fghi
```

```
> table1 = read.table(file="file.csv", sep=",")
> table1
       V1       V2       V3
1 column1  column2  column3
2       1     2.22     abcd
3       2     3.33     fghi
>
>
```

Excel

Microsoft Excel is the most widely used spreadsheet program which stores data in the .xls or .xlsx format. R can read directly from these files using some excel specific packages. Few such packages are - XLConnect, xlsx, gdata etc. We will be using the xlsx package. R can also write into excel files using this package.

Install xlsx Package

You can use the following command in the R console to install the "xlsx" package. It may ask to install some additional packages on which this package is dependent. Follow the same command with the required package name to install the additional packages.

```
install.packages("xlsx")
```

Use the following command to verify and load the "xlsx" package.

```
# Verify the package is installed.
any(grepl("xlsx",installed.packages()))

# Load the library into R workspace.
library("xlsx")
```

When the script is run we get the following output.

```
> any(grepl("xlsx", installed.packages()))
[1] TRUE
>
>
> library("xlsx")
>
```

Open Microsoft excel. Copy and paste the following data in the worksheet named as sheet1.

	A	B
1	**name**	**city**
2	Rick	Seattle
3	Dan	Tampa
4	Michelle	Chicago
5	Ryan	Seattle
6	Gary	Houston
7	Nina	Boston
8	Simon	Mumbai
9	Guru	Dallas

Save the Excel file as "cities.xlsx". You should save it in the current working directory of the R workspace.

```
~ [13:41:07]
0:~$ls cities.xlsx
cities.xlsx

~ [13:41:59]
0:~$pwd
/home/macgok
```

```
> getwd()
[1] "/home/macgok"
>
```

The cities.xlsx is read by using the *read.xlsx()* function as shown below. The result is stored as a data frame in the R environment.

```
data <- read.xlsx("cities.xlsx", sheetIndex = 1)
```

Binary Files

A binary file is a file that contains information stored only in the form of bits and bytes.(0's and 1's). They are not human readable as the bytes in it translate to characters and symbols which contain many other non-printable characters.

The binary file has to be read by specific programs to be usable. For example, the binary file of a Microsoft Word program can be read to a human readable form only by the Word program. Which indicates that, besides the human readable text, there is a lot more information like formatting of characters and page numbers etc., which are also stored along with alphanumeric

characters. And finally a binary file is a continuous sequence of bytes. The line break we see in a text file is a character joining the first line to the next.

Sometimes, the data generated by other programs are required to be processed by R as a binary file. Also R is required to create binary files which can be shared with other programs.

We consider the R inbuilt data "mtcars". First we create a csv file from it and convert it to a binary file and store it as an OS file. Next we read this binary file created into R.

We read the data frame "mtcars" as a csv file and then write it as a binary file to the OS.

```r
# Read the "mtcars" data frame as a csv file and store only the columns "cyl", "am" and "gear"
write.table(mtcars, file = "mtcars.csv",row.names = FALSE, na = "",
   col.names = TRUE, sep = ",")

# Store 5 records from the csv file as a new data frame.
new.mtcars <- read.table("mtcars.csv",sep = ",",header = TRUE,nrows = 5)

# Create a connection object to write the binary file using mode "wb".
write.filename = file("binmtcars.dat", "wb")

# Write the column names of the data frame to the connection object.
writeBin(colnames(new.mtcars), write.filename)

# Write the records in each of the column to the file.
writeBin(c(new.mtcars$cyl,new.mtcars$am,new.mtcars$gear), write.filename)

# Close the file for writing so that it can be read by other program.
close(write.filename)
```

The binary file created above stores all the data as continuous bytes. So we will read it by choosing appropriate values of column names as well as the column values.

```r
# Create a connection object to read the file in binary mode using "rb".
read.filename <- file("binmtcars.dat", "rb")

# First read the column names. n = 3 as we have 3 columns.
column.names <- readBin(read.filename, character(),  n = 3)

# Next read the column values. n = 18 as we have 3 column names and 15 values.
read.filename <- file("binmtcars.dat", "rb")
bindata <- readBin(read.filename, integer(),  n = 18)

# Print the data.
print(bindata)

# Read the values from 4th byte to 8th byte which represents "cyl".
cyldata = bindata[4:8]
print(cyldata)

# Read the values form 9th byte to 13th byte which represents "am".
amdata = bindata[9:13]
print(amdata)

# Read the values form 9th byte to 13th byte which represents "gear".
geardata = bindata[14:18]
print(geardata)

# Combine all the read values to a dat frame.
finaldata = cbind(cyldata, amdata, geardata)
colnames(finaldata) = column.names
print(finaldata)
```

XML Files

XML is a file format which shares both the file format and the data on the World Wide Web, intranets, and elsewhere using standard ASCII text. It stands for Extensible Markup Language (XML). Similar to HTML it contains markup tags. But unlike HTML where the markup tag describes structure of the page, in xml the markup tags describe the meaning of the data contained in the file.

You can read a xml file in R using the "XML" package. This package can be installed using the following command.

install.packages("XML")

In Linux, run the following commands to install the XML package.

sudo apt update
sudo apt-get install -y r-cran-xml

Create a XMl file by copying the below data into a text editor like notepad. Save the file with a .xml extension and choosing the file type as all files(*.*).

```
<RECORDS>
   <EMPLOYEE>
      <ID>1</ID>
      <NAME>Rick</NAME>
      <SALARY>623.3</SALARY>
      <STARTDATE>1/1/2012</STARTDATE>
      <DEPT>IT</DEPT>
   </EMPLOYEE>

   <EMPLOYEE>
      <ID>2</ID>
      <NAME>Dan</NAME>
      <SALARY>515.2</SALARY>
      <STARTDATE>9/23/2013</STARTDATE>
      <DEPT>Operations</DEPT>
   </EMPLOYEE>

   <EMPLOYEE>
      <ID>3</ID>
      <NAME>Michelle</NAME>
      <SALARY>611</SALARY>
      <STARTDATE>11/15/2014</STARTDATE>
      <DEPT>IT</DEPT>
   </EMPLOYEE>

   <EMPLOYEE>
      <ID>4</ID>
      <NAME>Ryan</NAME>
      <SALARY>729</SALARY>
      <STARTDATE>5/11/2014</STARTDATE>
      <DEPT>HR</DEPT>
   </EMPLOYEE>
</RECORDS>
```

The xml file is read by R using the function xmlParse(). It is stored as a list in R.

```
# Load the package required to read XML files.
library("XML")

# Also load the other required package.
library("methods")

# Give the input file name to the function.
result <- xmlParse(file = "input.xml")

# Print the result.
print(result)
```

Get Number of Nodes Present in XML File

```
# Load the packages required to read XML files.
library("XML")
library("methods")

# Give the input file name to the function.
result <- xmlParse(file = "input.xml")

# Exract the root node form the xml file.
rootnode <- xmlRoot(result)

# Find number of nodes in the root.
rootsize <- xmlSize(rootnode)

# Print the result.
print(rootsize)
```

Let's look at the first record of the parsed file. It will give us an idea of the various elements present in the top level node.

```r
# Load the packages required to read XML files.
library("XML")
library("methods")

# Give the input file name to the function.
result <- xmlParse(file = "input.xml")

# Exract the root node form the xml file.
rootnode <- xmlRoot(result)

# Print the result.
print(rootnode[1])
```

Get Different Elements of a Node

```r
# Load the packages required to read XML files.
library("XML")
library("methods")

# Give the input file name to the function.
result <- xmlParse(file = "input.xml")

# Exract the root node form the xml file.
rootnode <- xmlRoot(result)

# Get the first element of the first node.
print(rootnode[[1]][[1]])

# Get the fifth element of the first node.
print(rootnode[[1]][[5]])

# Get the second element of the third node.
print(rootnode[[3]][[2]])
```

To handle the data effectively in large files we read the data in the xml file as a data frame. Then process the data frame for data analysis.

```r
# Load the packages required to read XML files.
library("XML")
library("methods")

# Convert the input xml file to a data frame.
xmldataframe <- xmlToDataFrame("input.xml")
print(xmldataframe)
```