Importing and Exporting of Data in R:

1. writeLines() and readLines()

The writeLines() and readLines() functions in R are used for reading and writing data as lines of text. Here's the syntax and examples for each:

1. writeLines():
   - Syntax: writeLines(text, con)
     - `text`: A character vector or a single character string specifying the lines to be written.
     - `con`: The connection object or the file path where the text should be written.

   Example:
```
lines <- c("Line 1", "Line 2", "Line 3")
writeLines(lines, "output.txt")
```

   In this example, the character vector `lines` is written to a file named "output.txt" as three separate lines.

2. readLines():
   - Syntax: readLines(con, n)
     - con: The connection object or the file path from where the lines should be read.
     - n: The maximum number of lines to be read. If not specified, all lines are read.

   Example:

```
lines <- readLines("input.txt")
```

   In this example, the content of the file "input.txt" is read and stored in the character vector `lines`. Each line in the file is represented as an element in the vector.

It's important to note that the `writeLines()` function overwrites the contents of the file if it already exists. If you want to append lines to

an existing file, you can use the `append = TRUE` argument in the `writeLines()` function. For example:
writeLines("New line", "output.txt", append = TRUE)
This would append the string "New line" as a new line in the existing file "output.txt".

2. read.table() and write.table()

These functions in R are used for reading and writing tabular data in various formats. Here's the syntax and examples for each:

1. read.table():
   - Syntax: read.table(file, header = FALSE, sep = " ", quote = "\"", dec = ".", fill = TRUE, ...)`
     - `file`: The file name or connection to be read.
     - `header`: A logical value indicating whether the file has a header row. Default is `FALSE`.
     - `sep`: The separator used in the file to separate columns. Default is empty string `""`.
     - `quote`: The quote character used in the file to enclose character fields. Default is `"""`
     - `dec`: The character used in the file to represent decimal points. Default is `"."`
     - `fill`: A logical value indicating whether to fill in missing columns with `NA`. Default is `TRUE`.

   Example:
   data <- read.table("data.txt", header = TRUE, sep = "\t")
   In this example, the tab-separated file "data.txt" is read into a data frame called `data`. The file has a header row, and the columns are separated by tabs (`\t`).

```
df <- read.table(file='C:\\Users\\bob\\Desktop\\data.txt', header=TRUE)
```

```
#view data frame
print(df)

 var1 var2 var3
1   1   7   3
2   2   3   7
3   3   3   8
4   4   4   3
5   5   5   2
```

```
6  6  7  7
7  9  9  4
```

2. write.table():
- Syntax: write.table(x, file = "", sep = " ", quote = TRUE, dec = ".", row.names = TRUE, col.names = TRUE, ...)
    - `x`: The data object (e.g., data frame or matrix) to be written.
    - `file`: The file name or connection to write the data to. If empty, the output is printed to the console.
    - `sep`: The separator used to separate columns in the output file. Default is `" "`.
    - `quote`: A logical value indicating whether character fields should be enclosed in quotes. Default is `TRUE`.
    - `dec`: The character used to represent decimal points in numeric fields. Default is `"."`.
    - `row.names`: A logical value indicating whether to write row names. Default is `TRUE`.
    - `col.names`: A logical value indicating whether to write column names. Default is `TRUE`.

Example:
write.table(data, "output.txt", sep = ",", quote = FALSE)

```
sample_data <- data.frame( name= c("Geeks1", "Geeks2", "Geeks3",
                                   "Geeks4", "Geeks5", "Geeks6"),
                           value= c( 11, 15, 10, 23, 32, 53 ) )

# write dataframe into a space separated text file
write.table( sample_data, file='sample.txt')
```

```
# create sample dataframe
sample_data <- data.frame( name= c("Geeks1", "Geeks2", "Geeks3",
                                   "Geeks4", "Geeks5", "Geeks6"),
                           value= c( 11, 15, 10, 23, 32, 53 ) )

# write dataframe into a space separated text file
write.table( sample_data, file='sample.txt', sep=",")
```

In this example, the data frame `data` is written to a comma-separated file named "output.txt". The columns are separated by commas, and character fields are not enclosed in quotes.

3. read.csv() and write.csv() functions in R:

1. Reading a CSV file using `read.csv()`:
# Reading a CSV file into a data frame
Syntax:data <- read.csv("filename.csv")
In this example, we read a CSV file named `filename.csv` into a data frame called `data`. The file should be located in the current working directory or specified with the full file path.

2. Writing a data frame to a CSV file using `write.csv()`:
# Writing a data frame to a CSV file
Syntax: write.csv(data, "output.csv", row.names = FALSE)
In this example, we write the data frame called `data` to a CSV file named `output.csv`. The `row.names` argument is set to `FALSE` to exclude row names in the output file.

4.`dump()` and `source()`

In R, these functions are used for saving and loading R objects (such as functions, variables, and data) respectively. Here's a detailed explanation of each function:

`dump()` function:

The `dump()` function is used to save R objects to a file in binary format. It takes one or more R objects and writes them to a file, which can be later loaded using the `source()` function.

Syntax:
dump(list = ls(), file = "filename.RData")

Explanation:
- `list`: Specifies the objects to be saved. By default, it takes all the objects in the current environment using the `ls()` function.

- `file`: Specifies the filename or path where the objects will be saved. By convention, the filename should end with ".RData".

Example:

```
# Save the mtcars dataset and the object "my_variable" to a file
dump(list = c("mtcars", "my_variable"), file = "my_data.RData")
```

In this example, we save the `mtcars` dataset and an object called `my_variable` to a file named "my_data.RData".

`source()` function:

The `source()` function is used to load R code from a file and execute it. It reads the contents of the file and evaluates the R expressions within it, making the objects and functions defined in the file available in the current environment.

Syntax:

```
source("filename.R")
```

Explanation:

- `filename`: Specifies the name or path of the file to be sourced. It should be an R script file (ending with ".R") containing R code.

Example:

```
# Load and execute the R code from the file "my_script.R"
source("my_script.R")
```

In this example, we load and execute the R code from the file "my_script.R", which may contain functions, variable assignments, or any other valid R code.

5. `dput()` and `dget()`

In R, these functions are used to serialize R objects to a textual representation and deserialize them back into R objects, respectively. Here's a detailed explanation of each function:

1. `dput()` function:

The `dput()` function is used to serialize R objects into a textual representation. It generates an ASCII representation of an R object that can be easily read and reconstructed by R using the `dget()` function.

Syntax:
dput(object, file = "")

Explanation:
- `object`: Specifies the R object to be serialized.
- `file`: Optional parameter that specifies the file to which the serialized object will be written. If not specified, the output is printed to the console.

Example:
```
# Serialize the mtcars dataset and print the output
dput(mtcars)
```

In this example, the `dput()` function is used to serialize the `mtcars` dataset and the textual representation of the object is printed to the console.

2. `dget()` function:
The `dget()` function is used to deserialize R objects from their textual representation created by `dput()`. It reads the serialized object from a file or directly from a character vector and reconstructs it as an R object.

Syntax:
dget(file = "")

Explanation:
- `file`: Specifies the file or character vector containing the serialized object to be deserialized.

Example:
```
# Deserialize the object from the file "serialized_data.txt"
```

deserialized_data <- dget("serialized_data.txt")

In this example, the `dget()` function is used to deserialize the R object stored in the file "serialized_data.txt" and assign it to the variable `deserialized_data`.

6. `save()` and `load()` :
In R, these functions are used to save R objects to disk and load them back into an R session, respectively. Here's a detailed explanation of each function:

1. `save()` function:
The `save()` function is used to save one or more R objects to a file in binary format. The saved objects can be loaded back into R using the `load()` function.

Syntax:
save(..., file = "filename")

Explanation:
- `...`: Specifies one or more R objects to be saved. Objects can be specified by their names or using the `list` function.
- `file`: Specifies the name of the file where the objects will be saved. The file name should include the extension ".RData".

Example:
# Save the mtcars and iris datasets to a file named "data.RData"
save(mtcars, iris, file = "data.RData")

In this example, the `save()` function is used to save the `mtcars` and `iris` datasets to a file named "data.RData".

2. `load()` function:
The `load()` function is used to load previously saved R objects back into an R session. It reads the objects from the specified file and makes them available in the current R environment.

Syntax:
load("filename")

Explanation:
- `filename`: Specifies the name of the file from which the objects will be loaded. The file should be in the ".RData" format.

Example:
# Load the previously saved objects from the file "data.RData"
load("data.RData")

In this example, the `load()` function is used to load the objects saved in the "data.RData" file.

7. `serialize()` and `unserialize()`
In R, these functions are used to convert R objects into a binary representation and restore them back into their original form. These functions are useful for saving objects in a serialized format, such as when working with databases or transferring data between different programming languages. Here's an explanation of each function:

1. `serialize()` function:
The `serialize()` function converts an R object into a binary representation that can be saved or transmitted.

Syntax:
serialize(object, connection)

Explanation:
- `object`: Specifies the R object to be serialized.
- `connection`: Specifies the connection where the serialized object will be written. It can be a file connection or any other suitable connection.

Example:
# Serialize a numeric vector and write it to a file
data <- c(1, 2, 3, 4, 5)

```
file_conn <- file("serialized_data.bin", "wb")
serialize(data, file_conn)
close(file_conn)
```

In this example, the `serialize()` function is used to convert the `data` vector into a serialized binary representation, and then it is written to a file named "serialized_data.bin" using a binary file connection.

2. `unserialize()` function:
The `unserialize()` function reads a serialized object from a connection and restores it back into its original form as an R object.

Syntax:

```
unserialize(connection)
```

Explanation:
- `connection`: Specifies the connection from which the serialized object will be read.

Example:

```
# Read the serialized object from the file and unserialize it
file_conn <- file("serialized_data.bin", "rb")
data <- unserialize(file_conn)
close(file_conn)

# Check the restored object
print(data)
```

In this example, the `unserialize()` function is used to read the serialized object from the "serialized_data.bin" file using a binary file connection, and then it is restored back into its original form and assigned to the `data` variable.

8. `scan()`

In R, the function is used to read data values from a file or the console interactively. It is a versatile function that allows you to read data of different types, such as numeric values, character strings, logical values, and more. Here's an explanation of the `scan()` function in R:

Syntax:
scan(file = "", what = "", nmax = -1, ...)

Parameters:
- `file`: Specifies the file name or connection from which the data will be read. If omitted or an empty string, the data is read from the console.
- `what`: Specifies the type of data values to be read. It can be a character string specifying the data type (e.g., "numeric", "character", "logical"), or a list of such strings if the data contains multiple types. If omitted, it will try to automatically determine the data type.
- `nmax`: Specifies the maximum number of data values to be read. By default, it reads all the values in the file or until the end of the input stream.
- `...`: Additional arguments to be passed to the `read.table()` function, which is internally called by `scan()`.

Example 1: Reading Numeric Values from a File

```
# Create a text file with numeric values: data.txt
# 1
# 2
# 3
# 4
# 5

# Read the numeric values from the file
data <- scan("data.txt", what = numeric( ))

# Print the values
print(data)
```

In this example, the `scan()` function is used to read numeric values from the file "data.txt" and store them in the `data` vector. The `what = numeric()` parameter specifies that the values should be interpreted as numeric.

Example 2: Reading Character Strings from the Console

```
# Read three character strings from the console
strings <- scan(what = character(), nmax = 3)

# Print the strings
print(strings)
```

In this example, the `scan()` function is used to read three character strings from the console interactively. The `what = character()` parameter specifies that the values should be interpreted as character strings, and `nmax = 3` specifies that it should read only three values.

10. `read.delim()` and `write.delim()`

The functions in R are used to read and write tabular data in delimited text format, where the delimiter is typically a tab character ("\t"). Here's an explanation of the syntax and examples for both functions:

Syntax read.delim(file, header = TRUE, sep = "\t", ...)

Parameters:
- `file`: Specifies the file name or connection from which the data will be read.
- `header`: Specifies whether the file has a header row. By default, it is set to `TRUE` assuming the first row contains column names.
- `sep`: Specifies the delimiter used to separate columns. The default is a tab character ("\t").
- `...`: Additional arguments to be passed to the `read.table()` function, which is internally called by `read.delim()`.

Example: Reading a Tab-Delimited File

```
# Read tabular data from a delimited text file using read.delim()
data <- read.delim("data.txt")

# View the data frame
print(data)
```

In this example, the `read.delim()` function is used to read data from the file "data.txt", where the columns are separated by tabs. The resulting data is stored in the `data` object.

Syntax

```
write.delim(x, file, sep = "\t", ...)
```

Parameters:
- `x`: Specifies the data object to be written to the file.
- `file`: Specifies the file name or connection to which the data will be written.
- `sep`: Specifies the delimiter to be used to separate columns. The default is a tab character ("\t").
- `...`: Additional arguments to be passed to the `write.table()` function, which is internally called by `write.delim()`.

Example: Writing Data to a Tab-Delimited File

```
# Create a data frame
data <- data.frame(
  Name = c("John", "Jane", "Alice"),
  Age = c(25, 30, 35),
  Country = c("USA", "Canada", "UK")
)

# Write the data frame to a delimited text file using write.delim()
write.delim(data, "output.txt")
```