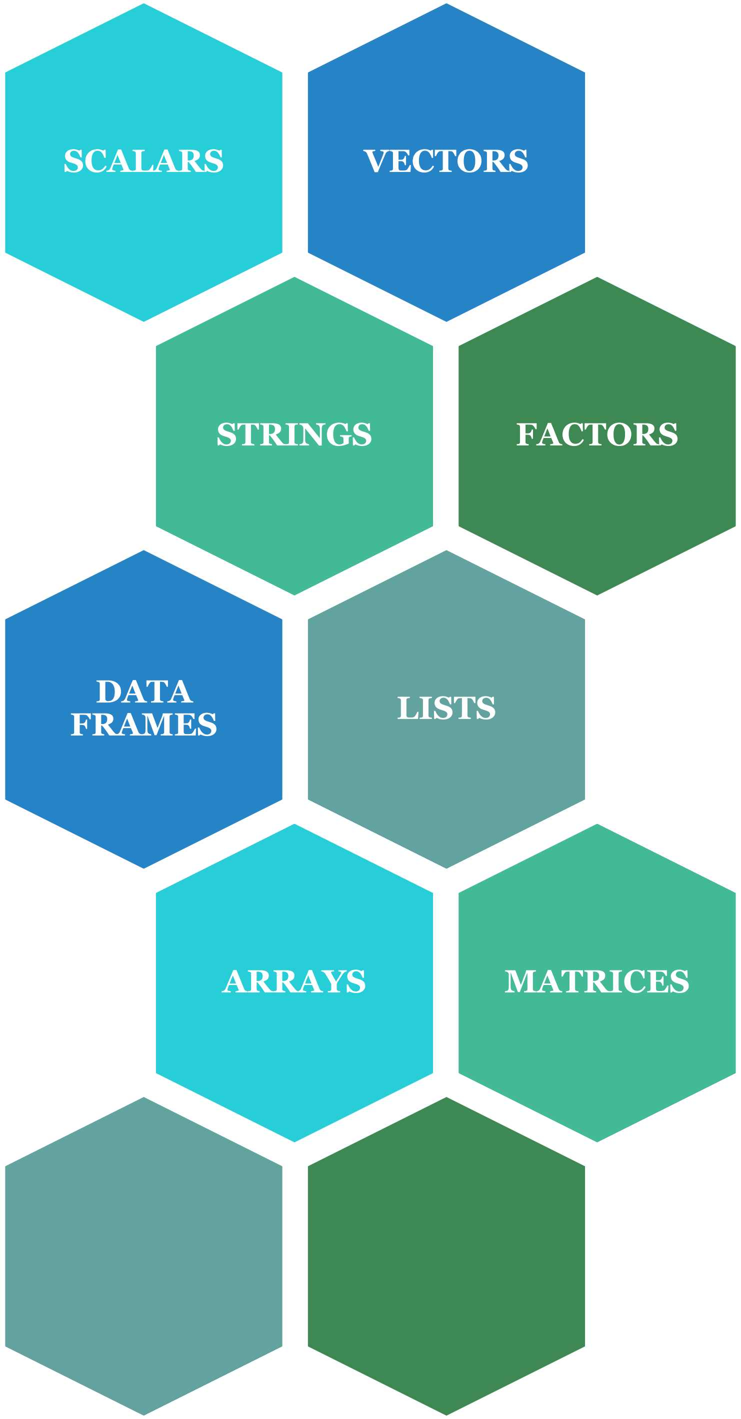
R Programming

OBJECT IN R

R has a wide variety of data types including



SCALARS

VECTORS

STRINGS

FACTORS

DATA FRAMES

LISTS

ARRAYS

MATRICES

Declaring an Object in R

Results of calculations can be stored in objects using the assignment operators:

- An arrow (<-) formed by a smaller than character and a hyphen without a space!
- The equal character (=)

Try this

```
A=1
B<-2
A
B
```

DECLARING AN OBJECT IN R

Object name: (RULES)

There are some restrictions when giving an object a name:

- Object names cannot contain `strange' symbols like !, +, -,#
- A dot (.) and an underscore (_) are allowed, also a name starting with a dot
- Object names can contain a number but cannot start with a number
- R is case sensitive, X and x are two different objects, as well as temp and temP

Variable Name	Validity	Reason
var_name2.	valid	Has letters, numbers, dot and underscore
var_name%	Invalid	Has the character '%'. Only dot(.) and underscore allowed.
2var_name	invalid	Starts with a number
.var_name, var.name	valid	Can start with a dot(.) but the dot(.)should not be followed by a number.
.2var_name	invalid	The starting dot is followed by a number making it invalid.
_var_name	invalid	Starts with _ which is not valid

DATA TYPES – SCALARS (NUMBER) DATA

- In computer programming, scalar refers to an atomic quantity that can hold only one value at a time. Scalars are the most basic data types of Number, Character & Logical Values.

- The plain values that are whole numbers are integer values.

- The values that contain decimals are called numeric.

Try this

```
#Number        #Logical value
               x=1                                   m & n       # AND
x <- 1         y=2.5                                 m | n      # OR
y <- 2.5       m <- x > y    # Is x larger than y?   !m         # Negation
class(x)       n <- x < y   # Is x smaller than y?
class(y)       m
class(x+y)     n
               class(m)
               class(n)
```

DATA TYPES – STRINGS

- A string is specified by using quotes
- Both single and double quotes will work
- The name of the **type** given to strings is *character*

Eg. a <- “hello”

Try this

```
a <- "1"; b <- "2.5"
a
b
a+b #a+b=3.5?
str(a)
str(b)

c=as.numeric(a)
d=as.numeric(b)
c+d #c+d=3.5?

str(c)
str(d)
```

DATA TYPES – VECTORS

- A collection of values that all have the same data type

- The elements of a vector are all numbers, giving a numeric vector, or all character values, giving a character vector

The simplest way to create a sample is to use the **c()** command

```
data1=c(3,5,7,5,3,2,6,8,5,6,9)
#Similar way we can enter text items as
day1=c('Mon','Tue','Wed','Thu')
```

Try this

#Vectors:

```
a <- c(1,2,5.3,6,-2,4) # numeric vector

b <- c("one","two","three") # character vector

c <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE) #logical vector

a[c(2,4)] # 2nd and 4th elements of vector
```

DATA TYPES – FACTORS

- **Factors** is a collection of values that all come from a fixed set of possible values.
- Factors can be used to represent a **categorical** variable in a data set.
- A list of levels, either numeric or string. No Char or Numeric operations can be carried out if the variable is in factor. Only grouping can be done.
- Examples: Gender (M/F)

Try this

```
#Factors - The factor stores the nominal values as a vector of integers

# variable gender with 20 "male" entries and
# 30 "female" entries
gender <- c(rep("male",20), rep("female", 30))
gender <- factor(gender)
# stores gender as 20 1s and 30 2s and associates
# 1=female, 2=male internally (alphabetically)
# R now treats gender as a nominal variable
summary(gender)
```

DATA TYPES – ARRAYS

Arrays are the R data objects which can store data in more than two dimensions.

array(data, dim=(rows, columns, no of matrices))

Try this

```
# Create two vectors of different lengths.
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)

# Take these vectors as input to the array.
result <- array(c(vector1,vector2),dim = c(3,3,2))
print(result)
```

```
# Create two vectors of different lengths.
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)
column.names <- c("COL1","COL2","COL3")
row.names <- c("ROW1","ROW2","ROW3")
matrix.names <- c("Matrix1","Matrix2")

# Take these vectors as input to the array.
result <- array(c(vector1,vector2),dim =
c(3,3,2),dimnames = list(row.names,column.names,
  matrix.names))
print(result)
```

DATA TYPES – ARRAYS

Try this

```
# Print the third row of the second matrix of the array.
print(result[3,,2])

# Print the element in the 1st row and 3rd column of the 1st matrix.
print(result[1,3,1])

# Print the 2nd Matrix.
print(result[,,2])
```

DATA TYPES – MATRICES

- A two-dimensional collection of values that all have the same type
- The values are arranged in rows and columns, There is also an **array** data structure that extends this idea to more than two dimensions

mymatrix <- **matrix(***vector*, **nrow=***r*, **ncol=***c*, **byrow=***FALSE*, **dimnames=list(***char_vector_rownames*, *char_vector_colnames***))**

byrow=TRUE indicates that the matrix should be filled by rows

DATA TYPES – MATRICES

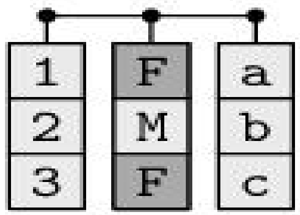
Try this

```
#Matrices

# generates 5 x 4 numeric matrix
y<-matrix(1:20, nrow=5,ncol=4)

# another example
cells <- c(1,26,24,68)
rnames <- c("R1", "R2")
cnames <- c("C1", "C2")
mymatrix <- matrix(cells, nrow=2, ncol=2, byrow=TRUE, dimnames=list(rnames, cnames))

#Identify rows, columns or elements using subscripts.
y[,4] # 4th column of matrix
y[3,] # 3rd row of matrix
y[2:4,1:3] # rows 2,3,4 of columns 1,2,3
```

DATA TYPES – DATA FRAMES

- A collection of vectors that all have the same length
- This is like a matrix, except that each column can contain a different data type.
- A data frame can be used to represent an entire data set.

data.frame(data)

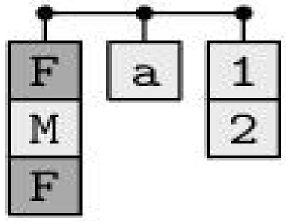


1	F	a
2	M	b
3	F	c

```
#Data Frames

d <- c(1,2,3,4)
e <- c("red", "white", "red", NA)
f <- c(TRUE,TRUE,TRUE,FALSE)
d
e
f
mydata <- data.frame(d,e,f)
mydata
names(mydata) <- c("ID","Color","Passed") # variable names
mydata

#Identify the elements of a data frame
mydata[2:3] # columns 3,4,5 of data frame
mydata[c("ID","Color")] # columns ID and Age from data frame
mydata$Color # variable x1 in the data frame
newdata <- mydata[1:2,2:3]
newdata
```

DATA TYPES – LISTS

- A collection of data structures
- The **components** of a list can be simply vectors-similar to a data frame, but with each column allowed to have a different length
- Lists can be used to store any combination of data values together



DATA TYPES – LISTS

Try this

```
#List

# example of a list with 4 components -
# a string, a numeric vector, a matrix, and a scaler
w <- list(name="Fred", mynumbers=a, mymatrix=y, age=5.3)
v <- list(name="Fred", mynumbers=a, mymatrix=y, test=5.3, age1=2.3)

# example of a list containing two lists
Final_list <- c(w,v)
Final_list
```

Identify elements of a list using the [[]] convention.

```
Final_list[[2]] # 2nd component of the list
Final_list[["mynumbers"]] # component named mynumbers in list
```

TASK!

1. Create a character Vector months with all month into it?
2. Create a Numeric Vector with rep function 12 1's and 2 2's?
3. Display the 3rd and 6th columns in mtcars dataset?
4. In mtcars, rename the variables mpg - "Miles_per_gallon", disp - "Displacement", hp - "Horse_Power"?
5. display 15th - 20th rows in mtcars dataset?

optional:
1. List the objects created in R?
2. Remove all objects created in R? Hint: rm(list = ls())

DATA TYPE CONVERSION

	to one long vector	To matrix	To data frame

from vector	c(x,y)	cbind(x,y)
		rbind(x,y)
			data.frame(x,y)

from matrix	as.vector(mymatrix)		as.data.frame(mymatrix)
from data frame		as.matrix(myframe)	

is.numeric(), is.character(), is.vector(), is.matrix(), is.data.frame()
as.numeric(), as.character(), as.vector(), as.matrix(), as.data.frame()

VIEWING NAMED OBJECTS

The ***ls()*** command lists all the named items that are available,
We can use the ***objects()*** command as well

Viewing only matching names : If we want to limit the display to object with certain names; this especially helpful if we have lot of data already in R

>ls(pattern=“a”) ,Here the pattern looks for the everything containing “a”

Try this

```
ls()
objects()
ls(pattern="a")
```

WORKING WITH HISTORY COMMANDS

We can Save the lot of time for coding by Using R built in history

We can access previous command history by using the up and down arrows in the keyboard

We can view the current list of history items by using the command ***history(),*** more specifically we can say like ***history(max.show=25)*** will displays the last 25 lines of the current history

OPERATORS IN R

Arithmetic Operators

Operator	Description	Operator
+	addition	3+5
-	subtraction	3-5
*	multiplication	3*5
/	division	5/3
^ or **	exponentiation	5^3 or 5**3
x %% y	modulus (x mod y)	5 %% 3
x %/% y	integer division	5 %/% 3

OPERATORS IN R

Relational Operators

Operator	Description	Examples

<	less than	v <- c(2,5.5,6,9)
		t <- c(8,2.5,14,9)
		print(v < t)

<=	less than or equal to	v <- c(2,5.5,6,9)
		t <- c(8,2.5,14,9)
		print(v<=t)

>	greater than	v <- c(2,5.5,6,9)
		t <- c(8,2.5,14,9)
		print(v>t)

>=	greater than or equal	v <- c(2,5.5,6,9)
		t <- c(8,2.5,14,9)
		print(v>=t)

==	exactly equal to	v <- c(2,5.5,6,9)
		t <- c(8,2.5,14,9)
		print(v == t)

!=	not equal to	v <- c(2,5.5,6,9)
		t <- c(8,2.5,14,9)
		print(v!=t)

OPERATORS IN R

Logical Operators

Operator	Description
!	Logical NOT
&	Element-wise logical AND
&&	Logical AND
|	Element-wise logical OR
| |	Logical OR

```
x <- c(TRUE,FALSE,0,6)
y <- c(FALSE,TRUE,FALSE,TRUE)
!x
[1] FALSE  TRUE  TRUE FALSE

x&y
[1] FALSE FALSE FALSE  TRUE

x&&y
[1] FALSE

x|y
[1]  TRUE  TRUE FALSE  TRUE

x||y
[1] TRUE
```

OPERATORS IN R

R Assignment Operators

Operator	Description
<-, <<-, =	Leftwards assignment
->, ->>	Rightwards assignment

```
x <- 5
x
[1] 5

x = 9
x
[1] 9

10 -> x
x
[1] 10
```

GETTING AND SETTING THE WORKING DIRECTORY

You can check which directory the R workspace is pointing to using the **getwd()** function. You can also set a new working directory using **setwd()**function.

Try this

```
# Get and print current working directory.
print(getwd())

# Set current working directory.
setwd(" C:\\Users\\jrv\\Documents\\test ")

# Get and print current working directory.
print(getwd())
```

IMPORTING DATA INTO R – READ.TABLE

```
Input_table=read.table(file.choose(),sep=";",header=TRUE, stringsAsFactors=FALSE)
```

Arguments	Description
file	The name of the file which the data are to be read from
header	a logical value indicating whether the file contains the names of the variables as its first line.
sep	the field separator character. Values on each line of the file are separated by this character.
stringsAsFactors	logical: should character vectors be converted to factors?
nrows	integer: the maximum number of rows to read in. Negative and other invalid values are ignored.
as.is	the default behavior of read.table is to convert character variables (which are not converted to logical, numeric or complex) to factors.

IMPORTING DATA INTO R – READ.CSV

```
csv_file=read.csv(file.choose(), header=TRUE, stringsAsFactors=FALSE)
```

Arguments	Description
file	The name of the file which the data are to be read from
header	a logical value indicating whether the file contains the names of the variables as its first line.
sep	the field separator character. Values on each line of the file are separated by this character.
stringsAsFactors	logical: should character vectors be converted to factors?
nrows	integer: the maximum number of rows to read in. Negative and other invalid values are ignored.
as.is	the default behavior of read.table is to convert character variables (which are not converted to logical, numeric or complex) to factors.

IMPORTING DATA INTO R – READ.XLSX

```
install.packages("xlsx")

library("xlsx")

data <- read.xlsx(file.choose(), sheetIndex = 1)
print(data)
```

IMPORTING DATA INTO R – JSON

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

# Give the input file name to the function.
result <- fromJSON(file = file.choose())

# Convert JSON file to a data frame.
json_data_frame <- as.data.frame(result)

print(json_data_frame)
```

IMPORTING DATA INTO R – XML

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

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

IMPORTING DATA INTO R – SQL SERVER

```
####	Connection to SQL Server through ODBC
sink("Data_Quality_Log.txt", append=TRUE, split=TRUE)
####	PACKAGE INITIALIZATION
library(RODBC)
library(sqldf)
#PLEASE PROVIDE SERVER AND DATBASE NAME
dbhandle <- odbcDriverConnect('driver={SQL
Server};server=servername;database=dbname;trusted_connection=true')

#VIEWING TABLE OBJECTS IN DATABASE
res <- sqlQuery(dbhandle, 'select * from information_schema.tables')
head(res,2)

#DATA IMPORT - PLEASE ENTER TABLE NAME
input1=sqlQuery(dbhandle, paste("select top 100 * from table_name"))
input2=sqlQuery(dbhandle, paste("select top 100 * from table_name"))

odbcClose(dbhandle)
```

WRITING DATA

```
# Write CSV in R
write.csv(MyData, file =
"MyData.csv",row.names=FALSE)
```

```
# Write CSV in R
write.csv(MyData, file =
"MyData.csv",row.names=FALSE, na="")
```

```
# Write CSV in R
write.table(MyData, file = "MyData.csv",row.names=FALSE, na="",col.names=FALSE,
sep=",")
```

```
write.xlsx(x, file, sheetName="Sheet1", col.names=TRUE, row.names=TRUE, append=FALSE)
write.xlsx2(x, file, sheetName="Sheet1", col.names=TRUE, row.names=TRUE, append=FALSE)
```

```
library(xlsx)
write.xlsx(USArrests, file="myworkbook.xlsx", sheetName="USA Arrests")
```

Thank You!