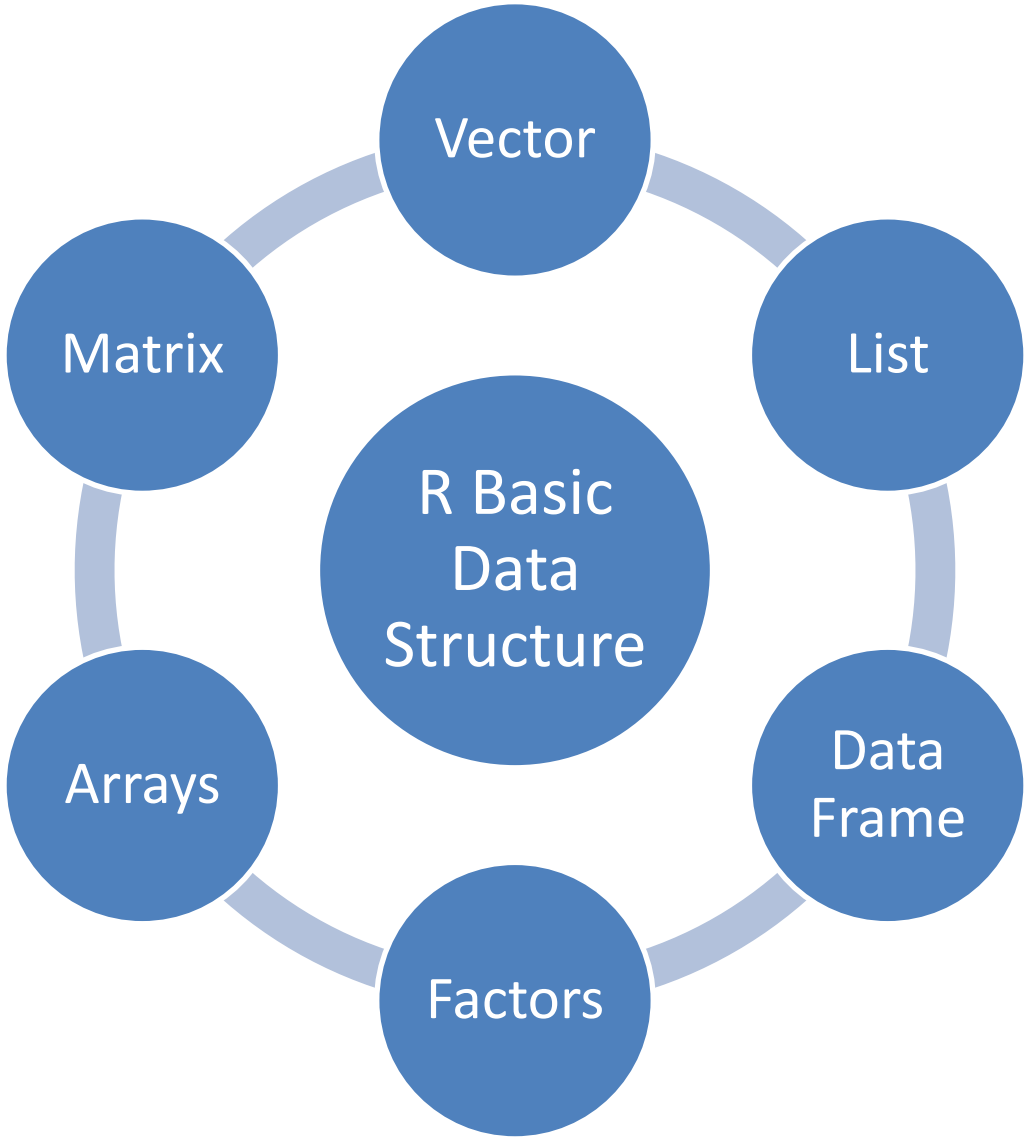
CCS2233 : STATISTICAL PROGRAMMING

Data Structure, Control Structure and Functions

Dr. Maz Jamilah Binti Masnan

Siti Nur Ezzati Binti Rusmi

Data Structure



Data Types

Numeric

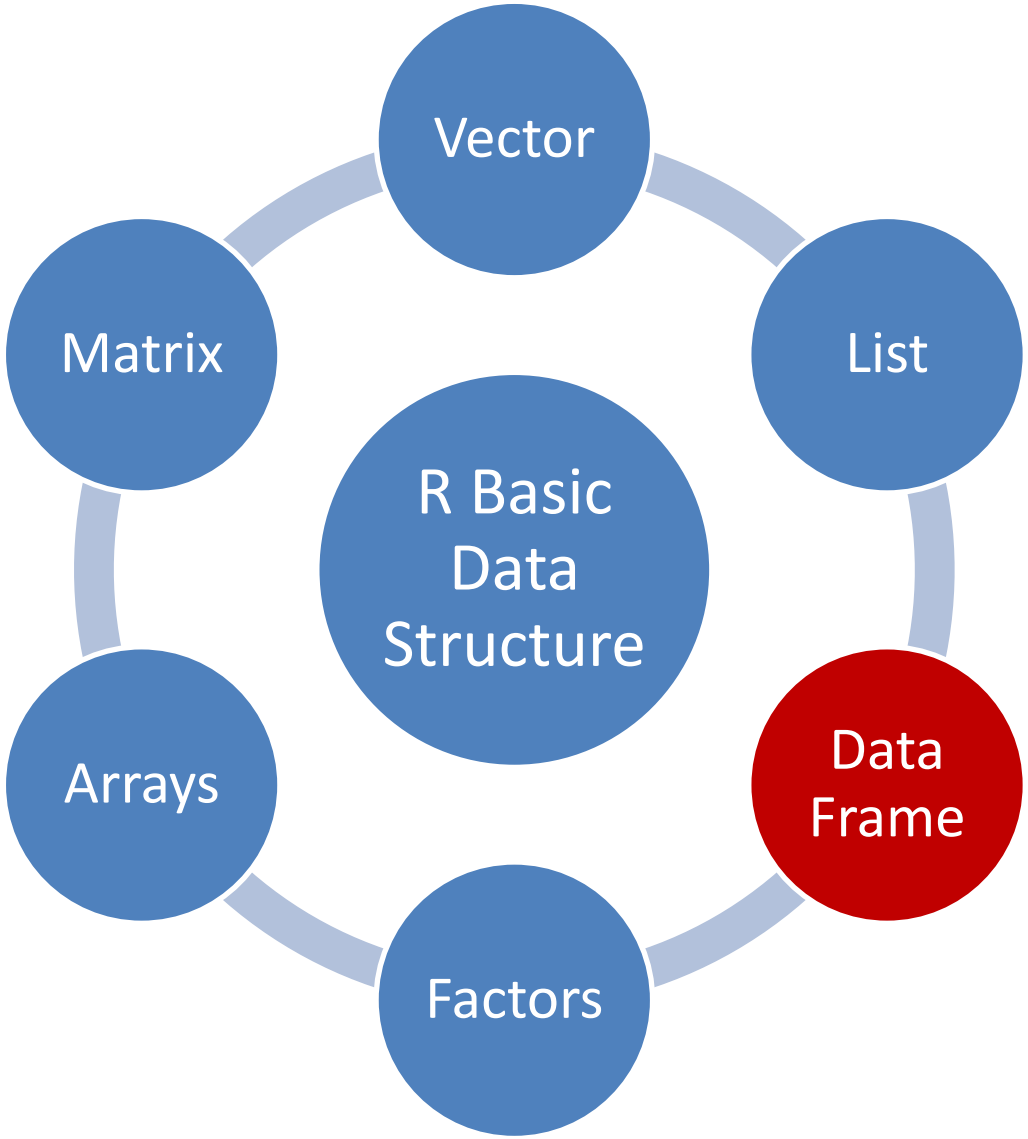
Integer

Logical

Character

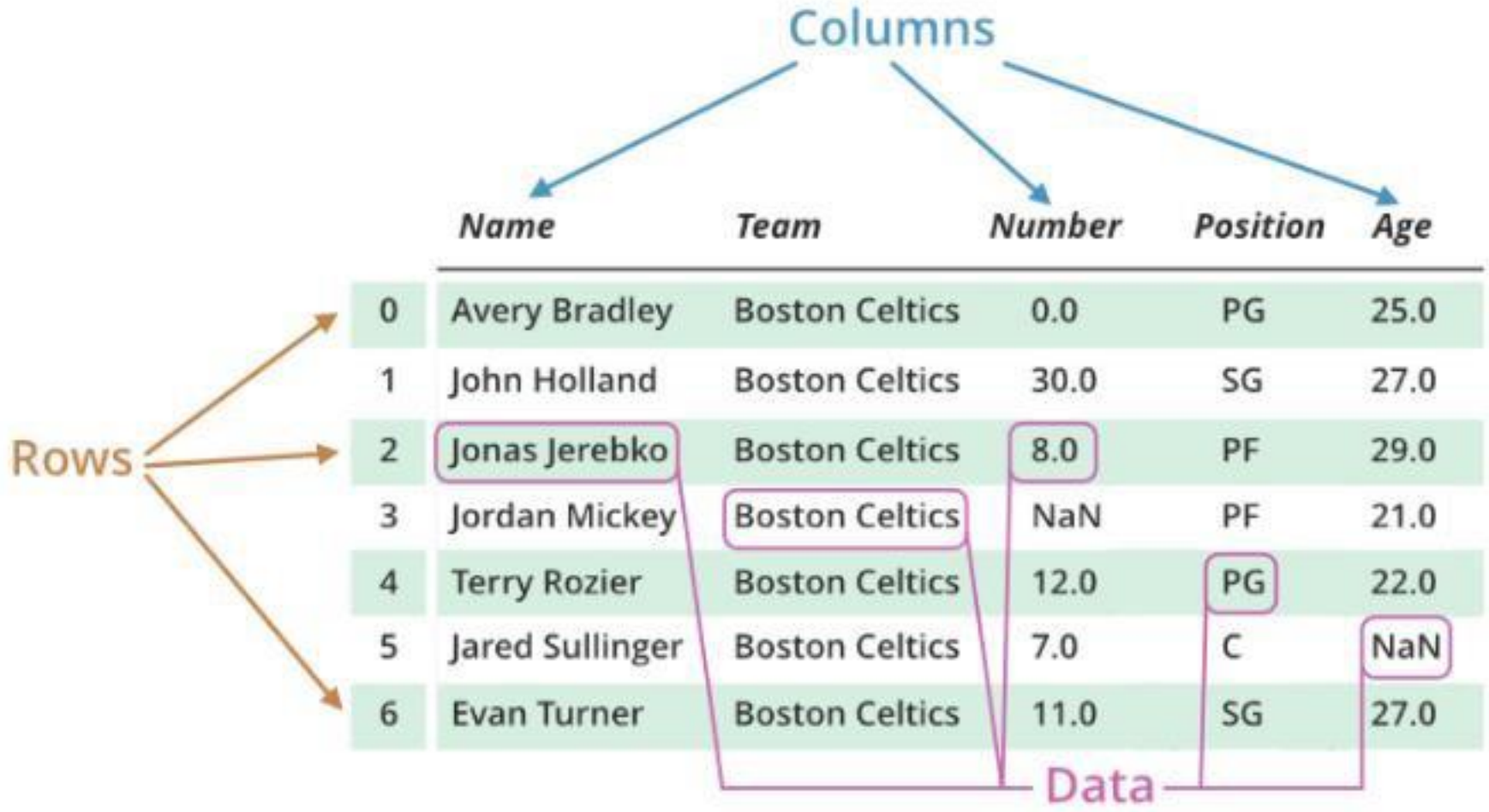
Complex

Data Structure – **Data Frame**



- **R Programming Language** is an open-source programming language that is widely used as a statistical software and data analysis tool.
- **Data Frames in R Language** are generic data objects of R which are used to store the tabular data.
- Data frames can also be interpreted as matrices where each column of a matrix can be of the different data types.
- DataFrame is made up of three principal components, the data, rows, and columns.

Data Frames (cont.)



	Name	Team	Number	Position	Age
0	Avery Bradley	Boston Celtics	0.0	PG	25.0
1	John Holland	Boston Celtics	30.0	SG	27.0
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN
6	Evan Turner	Boston Celtics	11.0	SG	27.0

Create Dataframe in R Programming Language

- To create a data frame in R use **data.frame()** command and then pass each of the vectors you have created as arguments to the function.

Input:

```r
# creating a data frame
friend.data <- data.frame(
  friend_id = c(1:5),
  friend_name = c("Ava", "Diana", "David", "Thomas", "Adam"),
  stringsAsFactors = FALSE
)

# print the data frame
print(friend.data)
```

Output:

```
  friend_id friend_name
1         1         Ava
2         2       Diana
3         3       David
4         4      Thomas
5         5        Adam
```

Get the Structure of the R – Data Frame

- One can get the structure of the data frame using **str()** function in R.
- It can display even the internal structure of large lists which are nested.
- It provides one-liner output for the basic R objects letting the user know about the object and its constituents.

Input:

```
# creating a data frame
friend.data <- data.frame(
  friend_id = c(1:5),
  friend_name = c("Ava", "Diana", "David", "Thomas", "Adam"),
  stringsAsFactors = FALSE
)

# using str()
print(str(friend.data))
```

Output:

```
'data.frame':   5 obs. of  2 variables:
 $ friend_id  : int  1 2 3 4 5
 $ friend_name: chr  "Ava" "Diana" "David" "Thomas" ...
NULL
```

Summary of data in the data frame

- In R data frame, the statistical summary and nature of the data can be obtained by applying **summary()** function.
- It is a generic function used to produce result summaries of the results of various model fitting functions.
- The function invokes particular methods which depend on the class of the first argument.

Input:

```
# creating a data frame
friend.data <- data.frame(
  friend_id = c(1:5),
  friend_name = c("Ava", "Diana", "David", "Thomas", "Adam"),
  stringsAsFactors = FALSE
)

# using summary()
print(summary(friend.data))
```

Output:

```
   friend_id friend_name
 Min.   :1   Length:5
 1st Qu.:2   Class :character
 Median :3   Mode  :character
 Mean   :3
 3rd Qu.:4
 Max.   :5
```

Extract Data from Data Frame in R Language

- Extract data from a data frame means that to access its rows or columns.
- One can extract a specific column from a data frame using its column name.

Input:

```r
# creating a data frame
friend.data <- data.frame(
  friend_id = c(1:5),
  friend_name = c("Ava", "Diana", "David", "Thomas", "Adam"),
  stringsAsFactors = FALSE
)

# Extracting friend_name column
result <- data.frame(friend.data$friend_name)
print(result)
```

Output:

```
  friend.data.friend_name
1                     Ava
2                   Diana
3                   David
4                  Thomas
5                    Adam
```

Expand Data Frame

- A data frame in R can be expanded by adding new columns and rows to the already existing data frame.
- In R, one can perform various types of operations on a data frame like **accessing rows and columns, selecting the subset of the data frame, editing data frames, delete rows and columns in a data frame**, etc.
- Please refer to DataFrame Operations in R to know about all types of operations that can be performed on a data frame.

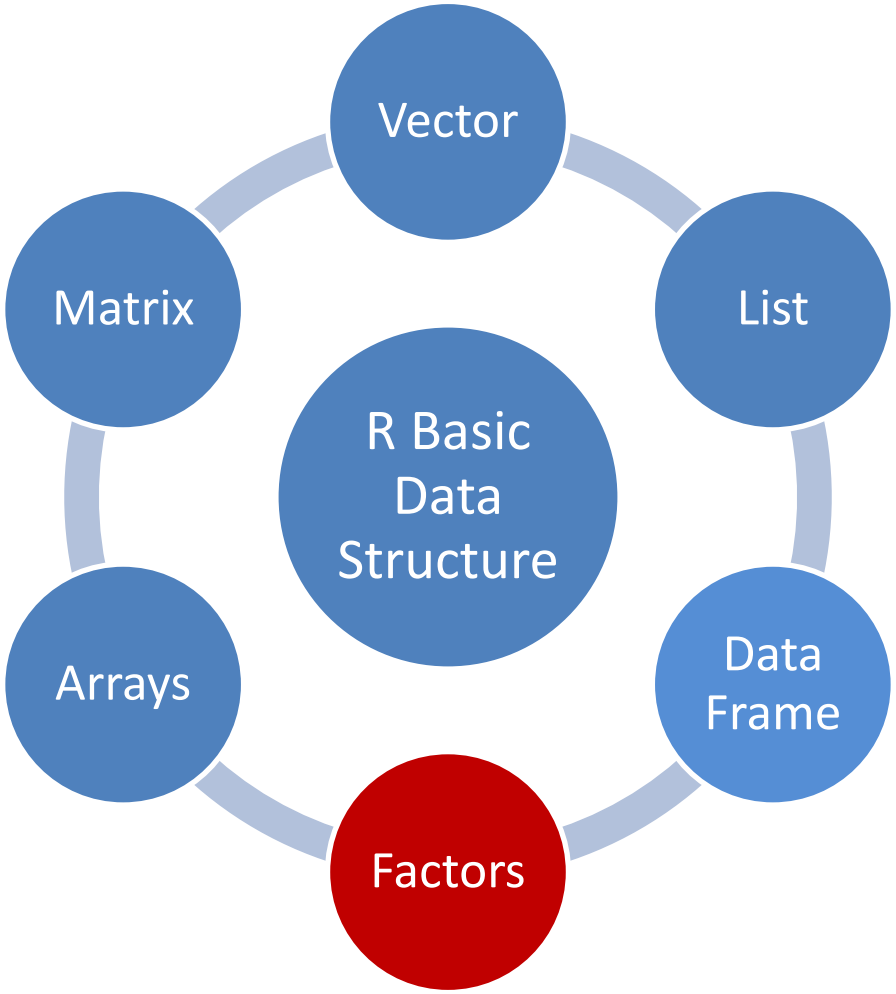
Input:

```
# creating a data frame
friend.data <- data.frame(
  friend_id = c(1:5),
  friend_name = c("Ava", "Diana", "David", "Thomas", "Adam"),
  stringsAsFactors = FALSE)

# Expanding data frame
friend.data$location <- c("Norwich", "London", "Birmingham", "Sheffield",
                          "Nottingham")
resultant <- friend.data

# print the modified data frame
print(resultant)
```

Output:

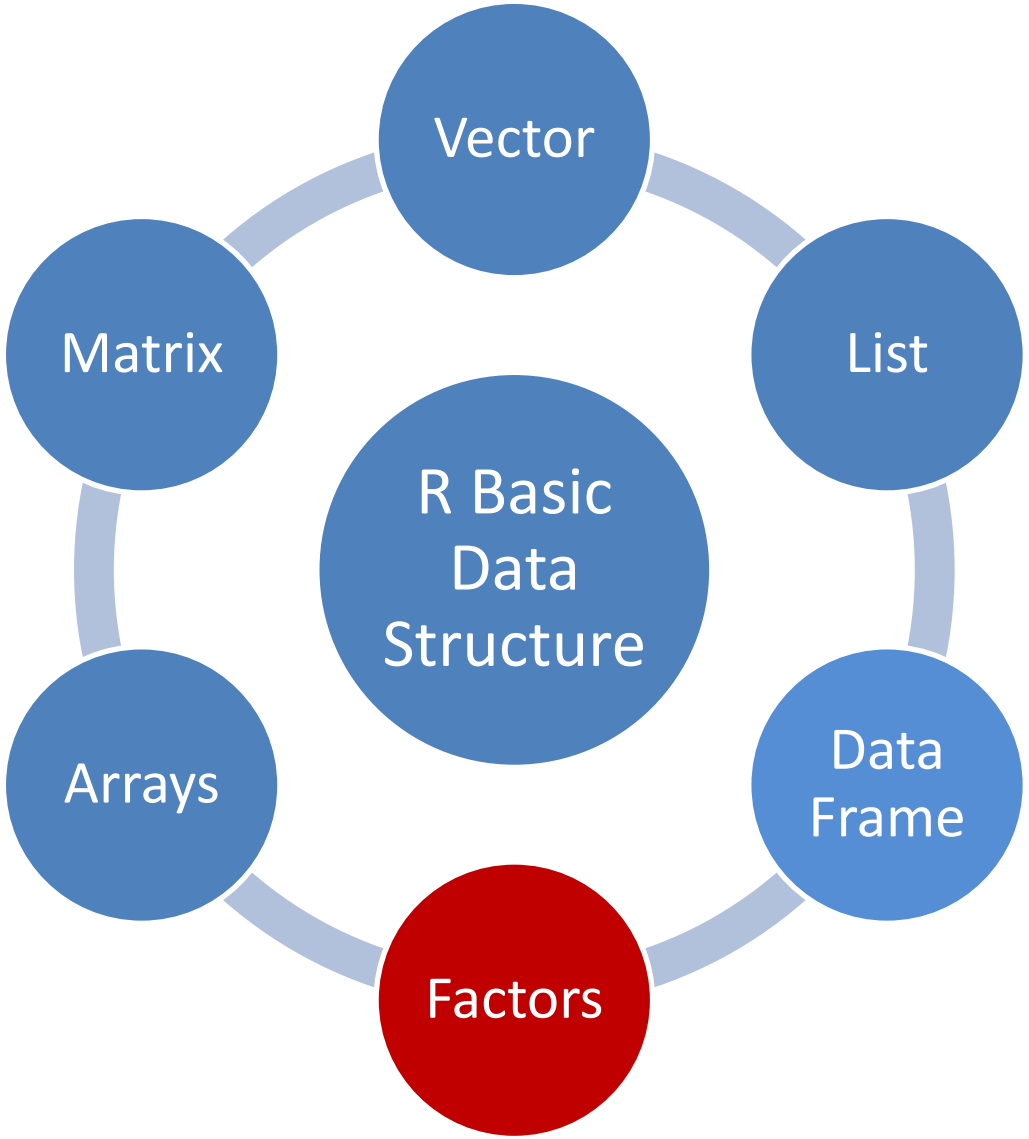
```
  friend_id friend_name   location
1         1         Ava    Norwich
2         2       Diana     London
3         3       David Birmingham
4         4      Thomas  Sheffield
5         5        Adam Nottingham
```

Data Structure - **Factors**



- Factors (are the variable) in R Programming Language are data structures that are implemented to categorize the data or represent categorical data (e.g. “male” “female”, “bad” “good” “better”) and store it on multiple levels.
- They can be stored as integers (e.g. 2,4,7) with a corresponding label to every unique integer.
- May look similar to character vectors, but they are integers, and care must be taken while using them as strings (eg. “ok1”)

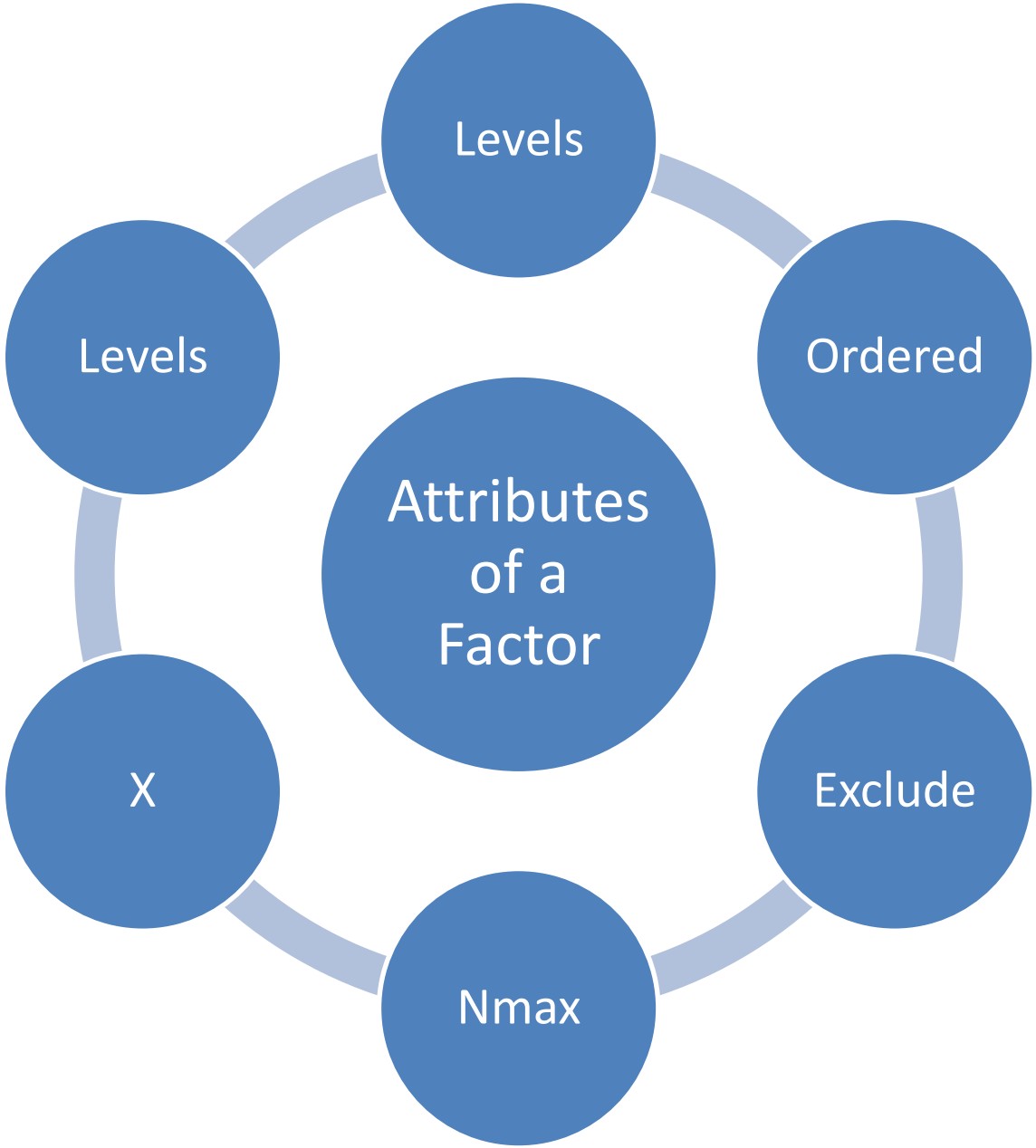
Data Structure - **Factors**



- The factor accepts only a restricted number of distinct values. For example, a data field such as gender may contain values only from female, male, or transgender.

- In the above example, all the possible cases are known beforehand and are predefined. These distinct values are known as levels. After a factor is created it only consists of levels that are by default sorted alphabetically.

Attributes of a Factor



Attributes of Factors in R Language

- **x:** It is the vector that needs to be converted into a factor.
- **Levels:** It is a set of distinct values which are given to the input vector x.
- **Labels:** It is a character vector corresponding to the number of labels.
- **Exclude:** This will mention all the values you want to exclude.
- **Ordered:** This logical attribute decides whether the levels are ordered.
- **nmax:** It will decide the upper limit for the maximum number of levels.

Creating a Factor in R Programming Language

- The command used to create or modify a factor in R language is –**factor()** with a vector as input.
- The two steps to creating a factor are:
  1. Creating a vector
  2. Converting the vector created into a factor using function factor()

Creating a Factor in R (cont.)

- **Example: Let us create a factor gender with levels female and male**

Input:

```
# Creating a vector
x <- c("female", "female", "male", "male")
print(x)

# Converting the vector x into a factor named gender
gender <- factor(x)
print(gender)
```

Output:

```
[1] "female" "female" "male"   "male"
[1] female female male   male
Levels: female male
```

Creating a Factor in R (cont.)

Levels can also be predefined by the programmer.

Input:

```
gender <- factor(c("female", "female", "male", "male"),
                 levels = c("female", "male"));
gender
```

Output:

```
[1] female female male   male
Levels: female male
```

Further one can check the levels of a factor by using function **levels()**.

Checking for a Factor in R

The function **is.factor()** is used to check whether the variable is a factor and returns “TRUE” if it is a factor.

Input:

```
gender <- factor(c("female", "female", "male", "male"));
print(is.factor(gender))
```

Output:

```
[1] TRUE
```

- Function **class()** is also used to check whether the variable is a factor and if true returns “factor”.

Input:

```
gender <- factor(c("female", "female", "male", "male"));
class(gender)
```

Output:

```
[1] "factor"
```

Accessing elements of a Factor in R

Like we access elements of a vector, the same way we access the elements of a factor. If gender is a factor then gender[i] would mean accessing i^{th} element in the factor.

Input:

```
gender <- factor(c("female", "female", "male", "male"));
gender[3]
```

Output:

```
[1] male
Levels: female male
```

Accessing elements of a Factor in R (cont)

- More than one element can be accessed at a time.

Input:

```
gender <- factor(c("female", "female", "male", "male"));
gender[c(2, 4)]
```

Output:

```
[1] female male
Levels: female male
```

Accessing elements of a Factor in R (cont)

Input:

```
gender <- factor(c("female", "female", "male", "male"));
gender[-3]
```

Output:

```
[1] female female male
Levels: female male
```

Modification of a Factor in R

- After a factor is formed, its components can be modified but the new values which need to be assigned must be at the predefined level.

Input:

```
gender <- factor(c("female", "female", "male", "male"));
gender[2]<-"male"
gender
```

Output:

```
[1] female male   male   male
Levels: female male
```

Factors in Data Frame

The Data frame is similar to a 2D array with the columns containing all the values of one variable and the rows having one set of values from every column. There are four things to remember about data frames:

- Column names are compulsory and cannot be empty.
- Unique names should be assigned to each row.
- The data frame's data can be only of three types- factor, numeric, and character type.
- The same number of data items must be present in each column.
- In R language when we create a data frame, its column is categorical data and hence a factor is automatically created on it.

We can create a data frame and check if its column is a factor.

Factors in Data Frame (cont.)

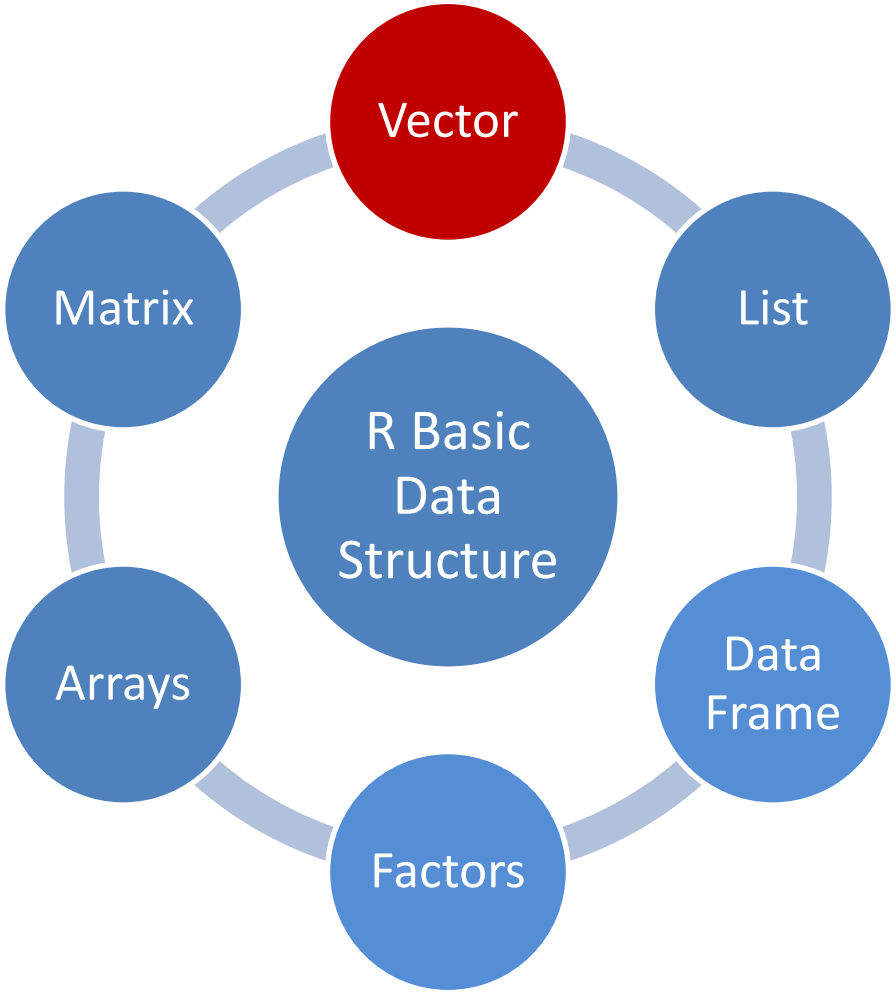
Input:

```
age <- c(42, 48, 49, 40, 52, 58, 53)
salary <- c(10320, 10620, 15020, 8606, 12390, 14070, 10220)
gender <- c("female", "male", "male", "female", "male", "female", "male")
employee<- data.frame(age, salary, gender)
print(employee)
print(is.factor(employee$gender))
```

Output:

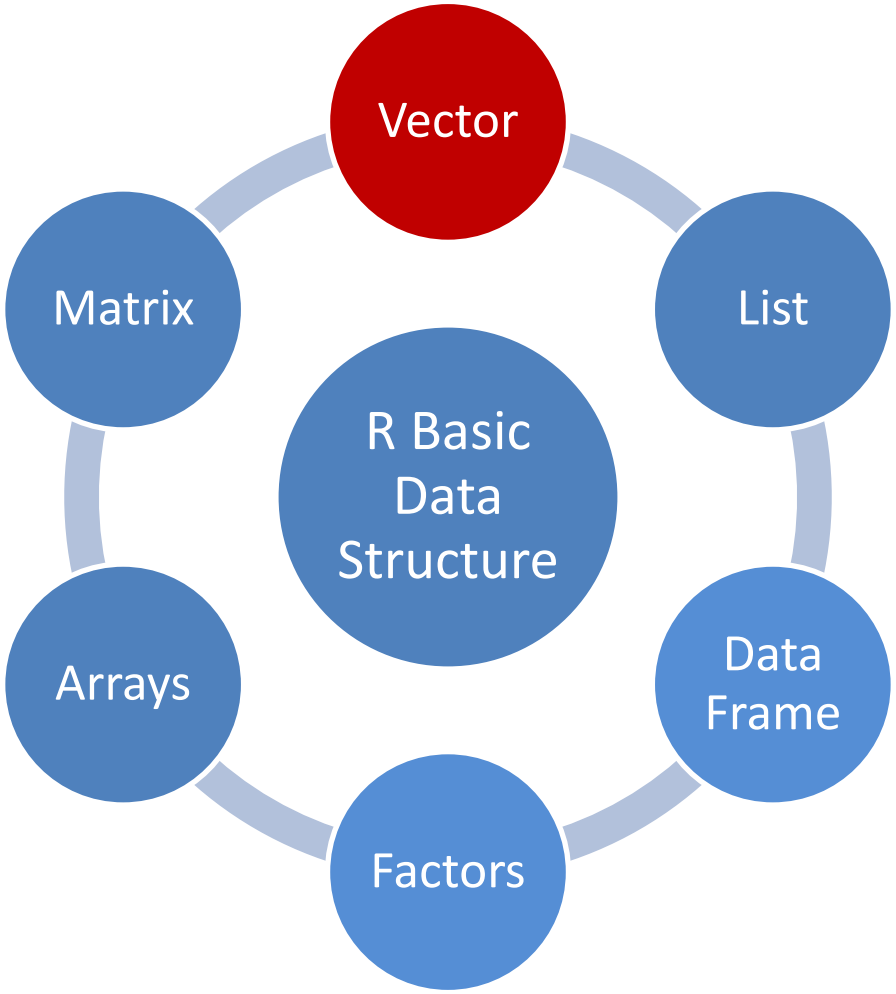
```
  age salary gender
1  42  10320 female
2  48  10620   male
3  49  15020   male
4  40   8606 female
5  52  12390   male
6  58  14070 female
7  53  10220   male
[1] FALSE
```

Data Structure - **Vector**



- R programming is one of the most popular languages when it comes to data science, statistical computations or scientific research.
- R programming is widely used in machine learning and it is very efficient and user-friendly.
- It provides flexibility in doing big statistical operations with a few lines of code.

Data Structure - **Vector**



- Vectors in R are the same as the arrays in C language which are used to hold multiple data values of the same type.
- One major key point is that in R the indexing of the vector will start from ‘1’ and not from ‘0’.
- We can create numeric vectors and character vectors as well.

What are Vectors (cont.)

Types of vectors

- Vectors are of different types which are used in R. There are many types of vectors. We will cover 3 type of vectors (Numeric, character and logical)

Numeric vectors

- Numeric vectors are those which contain numeric values such as integer, float, etc.

Numeric vectors

Input:

```
# creation of vectors using c() function
vector_1 <- c(4, 5, 6, 7)

# display type of vector
typeof(vector_1)

# use 'L' to specify that we want integer values.
vector_2 <- c(1L, 3L, 2L, 6L)

# display type of vector
typeof(vector_2)
```

Output:

```
[1] "double"
[1] "integer"
```

Character Vectors

- Character vectors contain alphanumeric values and special characters.

Input:

```
# by default numeric values are converted into characters
vector_1 <- c('world', '8', 'hello', 57)

# Displaying type of vector
typeof(vector_1)
```

Output:

```
[1] "character"
```

Logical vectors

- Logical vectors contain boolean values such as TRUE, FALSE and NA for Null values.

Input:

```
# Creating logical vector using c() function
vector_1 <- c(FALSE, FALSE, TRUE, NA)

# Displaying type of vector
typeof(vector_1)
```

Output:

```
[1] "logical"
```

Creating a vector

- There are different ways of creating vectors. Generally, we use ‘c’ to combine different elements together.

Input:

```
# use the c function combine the values as a vector.
# By default the type will be double
X <- c(61, 7, 21, 77, 89, 5)
cat('using c function', X, '\n')

# seq() function for creating creating a sequence of continuous values.
# length.out defines the length of vector.
Y <- seq(1, 10, length.out = 5)
cat('using seq() function', Y, '\n')

# use':' to create a vector of continuous values.
Z <- 2:7
cat('using colon', Z)
```

Output:

```
using c function 61 7 21 77 89 5
using seq() function 1 3.25 5.5 7.75 10
using colon 2 3 4 5 6 7
```

Accessing vector elements

- Accessing elements in a vector is the process of performing operation on an individual element of a vector. There are many ways through which we can access the elements of the vector. The most common is using the ‘[]’, symbol.

- ***Note:*** *Vectors in R are 1 based indexing unlike the normal C, python, etc format.*

Input:

```
# accessing elements with an index number.
X <- c(2, 3, 28, 19, 12)
cat('Using Subscript operator', X[2], '\n')

# by passing a range of values inside the vector index.
Y <- c(4, 13, 2, 6, 17)
cat('Using combine() function', Y[c(4, 1)], '\n')

# using logical expressions
Z <- c(5, 8, 1, 4, 4, 9)
cat('Using Logical indexing', Z[Z>4])
```

Output:

```
Using Subscript operator 3
Using combine() function 6 4
Using Logical indexing 5 8 9
```

Modifying a vector

- Modification of a Vector is the process of applying some operation on an individual element of a vector to change its value in the vector. There are different ways through which we can modify a vector:

Input:

```r
# Creating a vector
X <- c(2, 7, 6, 7, 5, 2)

# modify a specific element
X[3] <- 1
X[2] <- 3
cat('subscript operator', X, '\n')

# Modify using different logic.
X[X>5] <- 0
cat('Logical indexing', X, '\n')

# Modify by specifying the position or elements.
X <- X[c(3, 2, 1)]
cat('combine() function', X)
```

Output:

```
subscript operator 2 3 1 7 5 2
Logical indexing 2 3 1 0 5 2
combine() function 1 3 2
```

Deleting a vector

- Deletion of a Vector is the process of deleting all of the elements of the vector. This can be done by assigning it to a NULL value.

Input:

```
# Creating a Vector
V <- c(8, 11, 3, 5)

# set NULL to the vector
V <- NULL
cat('Output vector', V)
```

Output:

```
Output vector
```

Sorting elements of a Vector

- sort() function is used with the help of which we can sort the values in ascending or descending order.

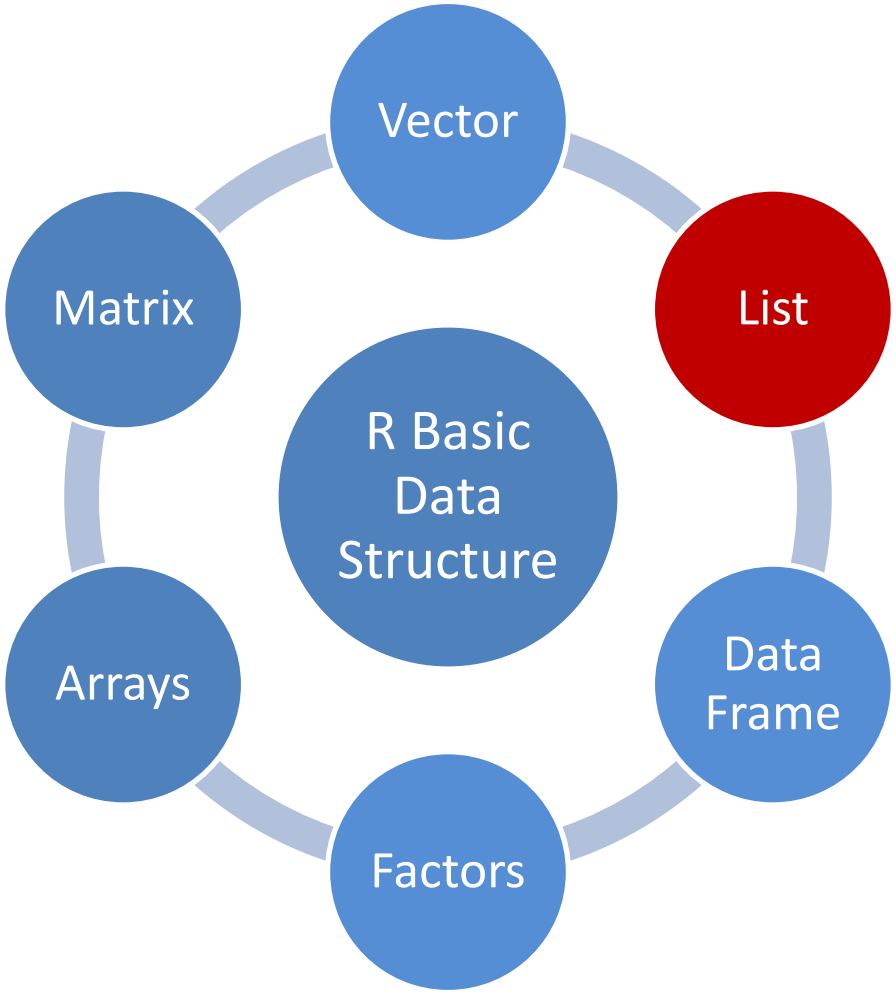
Input:

```
# Creation of Vector
X <- c(5, 5, 7, 1, 11, 2)

# Sort in ascending order
A <- sort(X)
cat('ascending order', A, '\n')

# sort in descending order by setting decreasing as TRUE
B <- sort(X, decreasing = TRUE)
cat('descending order', B)
```

Output:

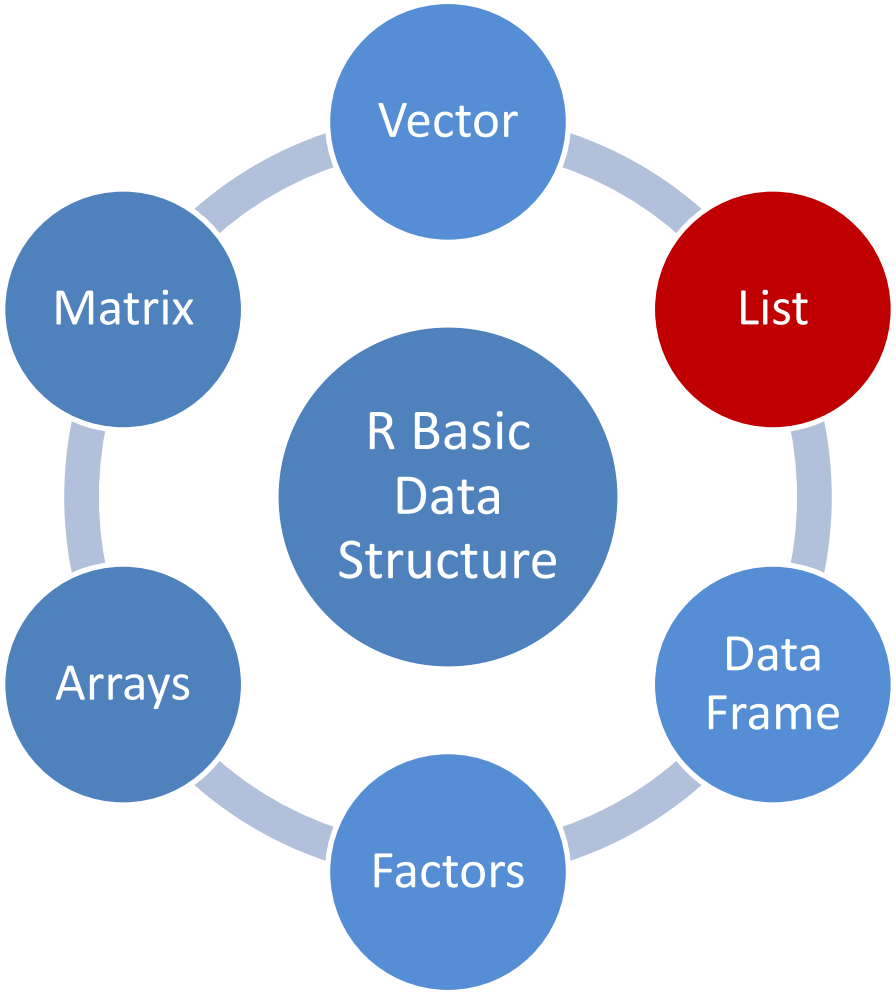
```
ascending order 1 2 5 5 7 11
descending order 11 7 5 5 2 1
```

Data Structure - **List**



- A list in R is a generic object consisting of an ordered collection of objects.
- Lists are one-dimensional, heterogeneous data structures.
- The list can be a list of vectors, a list of matrices, a list of characters and a list of functions, and so on.
- A list is a vector but with heterogeneous data elements.

Data Structure - **List**



- A list in R is created with the use of **list()** function.
- R allows accessing elements of a list with the use of the index value.
- In R, the indexing of a list starts with 1 instead of 0 like other programming languages.

Creating a List

- To create a List in R you need to use the function called “list()”. In other words, a list is a generic vector containing other objects. To illustrate how a list looks, we take an example here. We want to build a list of employees with the details. So for this, we want attributes such as ID, employee name, and the number of employees.

Input:

```r
# First attribute - a numeric vector containing the employee IDs
emp_id = c(1, 2, 3, 4)

# Second attribute - a character vector containing employees name
emp_name = c("Thomas", "Sandra", "Haris",  "Sofia")

# Third attribute - a single numeric variable containing the number of employees
number_of_emp = 4

# Combine all these data types into a list containing the details of employees
emp_list = list(emp_id, emp_name, number_of_emp)

print(emp_list)
```

Output:

```
[[1]]
[1] 1 2 3 4

[[2]]
[1] "Thomas" "Sandra" "Haris"  "Sofia"

[[3]]
[1] 4
```

Accessing components of a list

- We can access components of a list in two ways.
- **Access components by names:** All the components of a list can be named and we can use those names to access the components of the list using the dollar command.

Input:

```
# Creating a list by naming all its  components
emp_id = c(1, 2, 3, 4)
emp_name = c("Thomas", "Sandra", "Haris",  "Sofia")
number_of_emp = 4
emp_list = list("ID" = emp_id,
                "Name" = emp_name,
                "Total Staff" = number_of_emp)

print(emp_list)

# Accessing components by names
cat("Accessing name components using $ command\n")
print(emp_list$Name)
```

Output:

```
$ID
[1] 1 2 3 4

$Name
[1] "Thomas" "Sandra" "Haris"  "Sofia"

$`Total Staff`
[1] 4
Accessing name components using $ command
[1] "Thomas" "Sandra" "Haris"  "Sofia"
```

Access components by indices

- We can also access the components of the list using indices.

- To access the top-level components of a list we have to use a double slicing operator "**[[ ]]**" which is two square brackets and if we want to access the lower or inner level components of a list we have to use another square bracket "**[ ]**" along with the double slicing operator "**[[ ]]**".

Input:

```
# Creating a list by naming all its components
emp_id = c(1, 2, 3, 4)
emp_name = c("Thomas", "Sandra", "Haris",  "Sofia")
number_of_emp = 4
emp_list = list("ID" = emp_id,
                "Name" = emp_name,
                "Total Staff" = number_of_emp)

print(emp_list)

# Accessing a top level components by indices
cat("Accessing name components using indices\n")
print(emp_list[[2]])

# Accessing a inner level components by indices
cat("Accessing Sandra from name using indices\n")
print(emp_list[[2]][2])

# Accessing another inner level components by  indices
cat("Accessing 4 from ID using indices\n")
print(emp_list[[1]][4])
```

Output:

```
$ID
[1] 1 2 3 4

$Name
[1] "Thomas" "Sandra" "Haris"  "Sofia"

$`Total Staff`
[1] 4

>
> # Accessing a top level components by indices
> cat("Accessing name components using indices\n")
Accessing name components using indices
> print(emp_list[[2]])
[1] "Thomas" "Sandra" "Haris"  "Sofia"
>
> # Accessing a inner level components by indices
> cat("Accessing Sandra from name using indices\n")
Accessing Sandra from name using indices
> print(emp_list[[2]][2])
[1] "Sandra"
>
> # Accessing another inner level components by  indices
> cat("Accessing 4 from ID using indices\n")
Accessing 4 from ID using indices
> print(emp_list[[1]][4])
[1] 4
```

Modifying components of a list

- A list can also be modified by accessing the components and replacing them with the ones which you want.

Input:

```r
# Creating a list by naming all its  components
emp_id = c(1, 2, 3, 4)
emp_name = c("Thomas", "Sandra", "Haris",  "Sofia")
number_of_emp = 4
emp_list = list("ID" = emp_id,
                "Name" = emp_name,
                "Total Staff" = number_of_emp)

cat("Before modifying the list\n")
print(emp_list)

# Modifying the top-level component
emp_list$`Total Staff` = 5

# Modifying inner level component
emp_list[[1]][5] = 5
emp_list[[2]][5] = "Kamala"

cat("After modified the list\n")
print(emp_list)
```

Output:

```
Before modifying the list
$ID
[1] 1 2 3 4

$Name
[1] "Thomas" "Sandra" "Haris"  "Sofia"

$`Total Staff`
[1] 4

After modified the list
$ID
[1] 1 2 3 4 5

$Name
[1] "Thomas" "Sandra" "Haris"  "Sofia"  "Kamala"

$`Total Staff`
[1] 5
```

Concatenation of lists

Two lists can be concatenated using the concatenation function. So, when we want to concatenate two lists we have to use the concatenation operator.

Syntax:

list = c(list, list1)

list = the original list

list1 = the new list

Input:

```
# Creating a list by naming all its components
emp_id = c(1, 2, 3, 4)
emp_name = c("Thomas", "Sandra", "Haris",  "Sofia")
number_of_emp = 4
emp_list = list("ID" = emp_id,
                "Name" = emp_name,
                "Total Staff" = number_of_emp)

cat("Before concatenation of the new  list\n")
print(emp_list)

# Creating another list
emp_age = c(34, 27, 20, 45)
emp_age_list = list("Age" = emp_age)

# Concatenation of list using  concatenation operator
emp_list = c(emp_list, emp_age_list)

cat("After concatenation of the new  list\n")
print(emp_list)
```

Output:

```
Before concatenation of the new  list
$ID
[1] 1 2 3 4

$Name
[1] "Thomas" "Sandra" "Haris"  "Sofia"

$`Total Staff`
[1] 4

After concatenation of the new  list
$ID
[1] 1 2 3 4

$Name
[1] "Thomas" "Sandra" "Haris"  "Sofia"

$`Total Staff`
[1] 4

$Age
[1] 34 27 20 45
```

Deleting components of a list

- To delete components of a list, first of all, we need to access those components and then insert a negative sign before those components. It indicates that we had to delete that component.

Input:

```
# Creating a list by naming all its  components
emp_id = c(1, 2, 3, 4)
emp_name = c("Thomas", "Sandra", "Haris",  "Sofia")
number_of_emp = 4
emp_list = list("ID" = emp_id,
                "Name" = emp_name,
                "Total Staff" = number_of_emp)

cat("Before deletion the list is\n")
print(emp_list)

# Deleting a top level components
cat("After Deleting Total staff  components\n")
print(emp_list[-3])

# Deleting a inner level components
cat("After Deleting sandeep from  name\n")
print(emp_list[[2]][-2])
```

Output:

```
Before deletion the list is
$ID
[1] 1 2 3 4

$Name
[1] "Thomas" "Sandra" "Haris"  "Sofia"

$`Total Staff`
[1] 4

After Deleting Total staff  components
$ID
[1] 1 2 3 4

$Name
[1] "Thomas" "Sandra" "Haris"  "Sofia"

After Deleting sandeep from  name
[1] "Thomas" "Haris"  "Sofia"
```

Merging list

- We can merge the list by placing all the lists into a single list.

Input:

```
#Create two lists
list_1 <- list(1,2,3)
list_2 <- list("Fri","Sat","Sun")

# Merge the two lists
new_list <- c(list_1,list_2)

# Print the merged list
print(new_list)
```

Output:

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

[[2]]
[1] 2

[[3]]
[1] 3

[[4]]
[1] "Fri"

[[5]]
[1] "Sat"

[[6]]
[1] "Sun"
```

Converting List to Vector

- Here we are going to convert the list to vector, for this we will create a list first and then unlist the list into the vector.

Input:

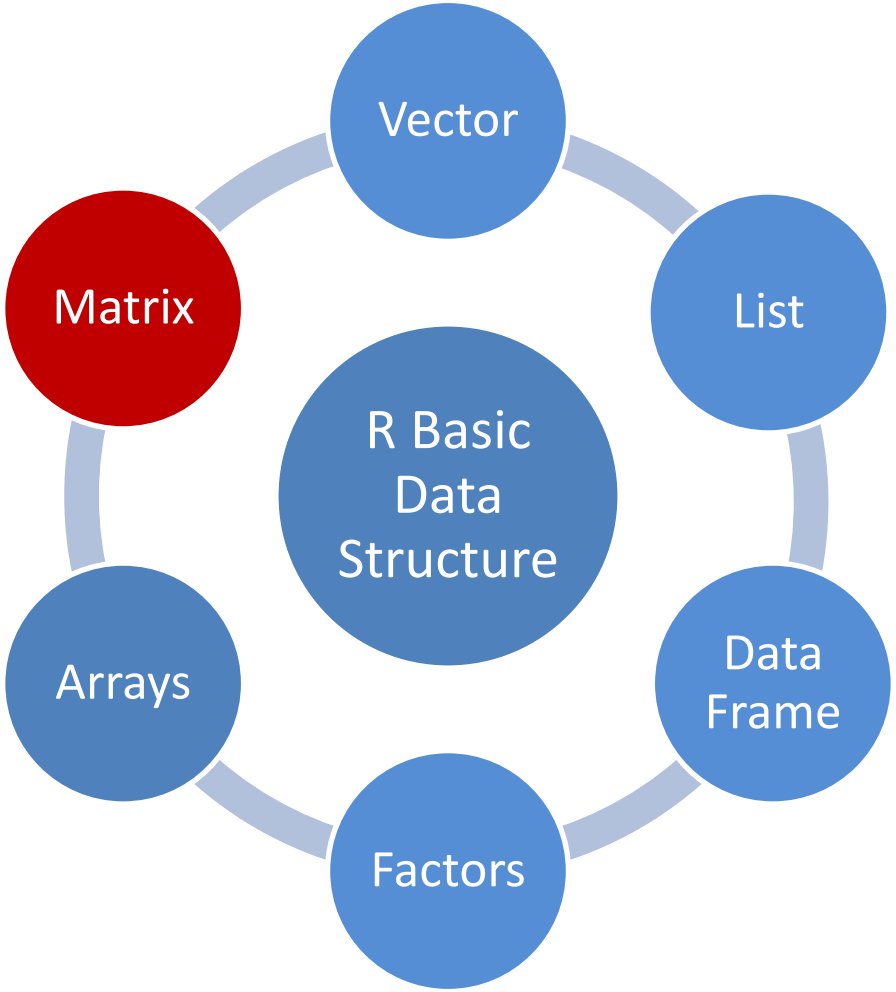
```
lst <- list(1:5)
print(lst)

# Convert the lists to vectors
vect <- unlist(lst)

print(vect)
```

Output:

```
[[1]]
[1] 1 2 3 4 5

[1] 1 2 3 4 5
```

Data Structure - R List to **Matrix**

- We will create matrices using matrix() function in R programming.
- Another function that will be used is unlist() function to convert the lists into a vector.
- The unlist() method in R will simplify it to produce a vector that contains all the atomic components which occur in list data



Syntax

- matrix<(unlist(z), ncol =, nrow =)

- Example:

Input:

```
vect_1 <- 1:5
vect_2 <- 6:10
vect_3 <- 11:15
vect_4 <- 16:20

# convert vector to list and pass all the vectors in the argument
data <- list(vect_1, vect_2, vect_3, vect_4)
data
```

Output:

```
[[1]]
[1] 1 2 3 4 5

[[2]]
[1]  6  7  8  9 10

[[3]]
[1] 11 12 13 14 15

[[4]]
[1] 16 17 18 19 20
```

- To convert the list to matrix, use the matrix() function. The matrix() function takes three parameters.
- data: unlist(data)
- ncol
- nrow
- We will create a 5×4 matrix. See the following code.

Input:

```
vect_1 <- 1:5
vect_2 <- 6:10
vect_3 <- 11:15
vect_4 <- 16:20

data <- list(vect_1, vect_2, vect_3, vect_4)
mtrx <- matrix(unlist(data), ncol = 4, nrow = 5)
mtrx
```

Output:

```
     [,1] [,2] [,3] [,4]
[1,]    1    6   11   16
[2,]    2    7   12   17
[3,]    3    8   13   18
[4,]    4    9   14   19
[5,]    5   10   15   20
```

You have now converted a list into the matrix.

Input:

```r
# Defining list
list_1 <- list(list(2, 3, 4), list(6, 7, 8))

# Print list
cat("The list is:\n")
print(list_1)
cat("Class:", class(list_1), "\n")

# Convert list to matrix
mat <- matrix(unlist(list_1), nrow = 2, byrow = TRUE)

# Print matrix
cat("\nAfter conversion to matrix:\n")
print(mat)
cat("Class:", class(mat), "\n")
```

Output:

```
The list is:
[[1]]
[[1]][[1]]
[1] 2

[[1]][[2]]
[1] 3

[[1]][[3]]
[1] 4


[[2]]
[[2]][[1]]
[1] 6

[[2]][[2]]
[1] 7

[[2]][[3]]
[1] 8


Class: list

After conversion to matrix:
     [,1] [,2] [,3]
[1,]    2    3    4
[2,]    6    7    8
Class: matrix array
```