KHWAJA YUNUS ALI UNIVERSITY
QUEST FOR KNOWLEDGE

Assignment: 01

Topic: Introduction to R Programming Language

Course Title: Data Mining

Course Code: CSE 4107

Submit To:

A. Hasib Uddin

Lecturer, Dept. of Computer Science and Engineering

Khwaja Yunus Ali University

Submitted By:

Name: Hasnain Kabir

ID: 0621930105101004

Semester: Spring 2023

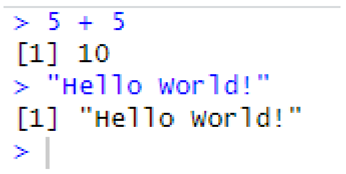
Batch: 9th

Submission Date: 23rd January, 2023

Assignment No: 1

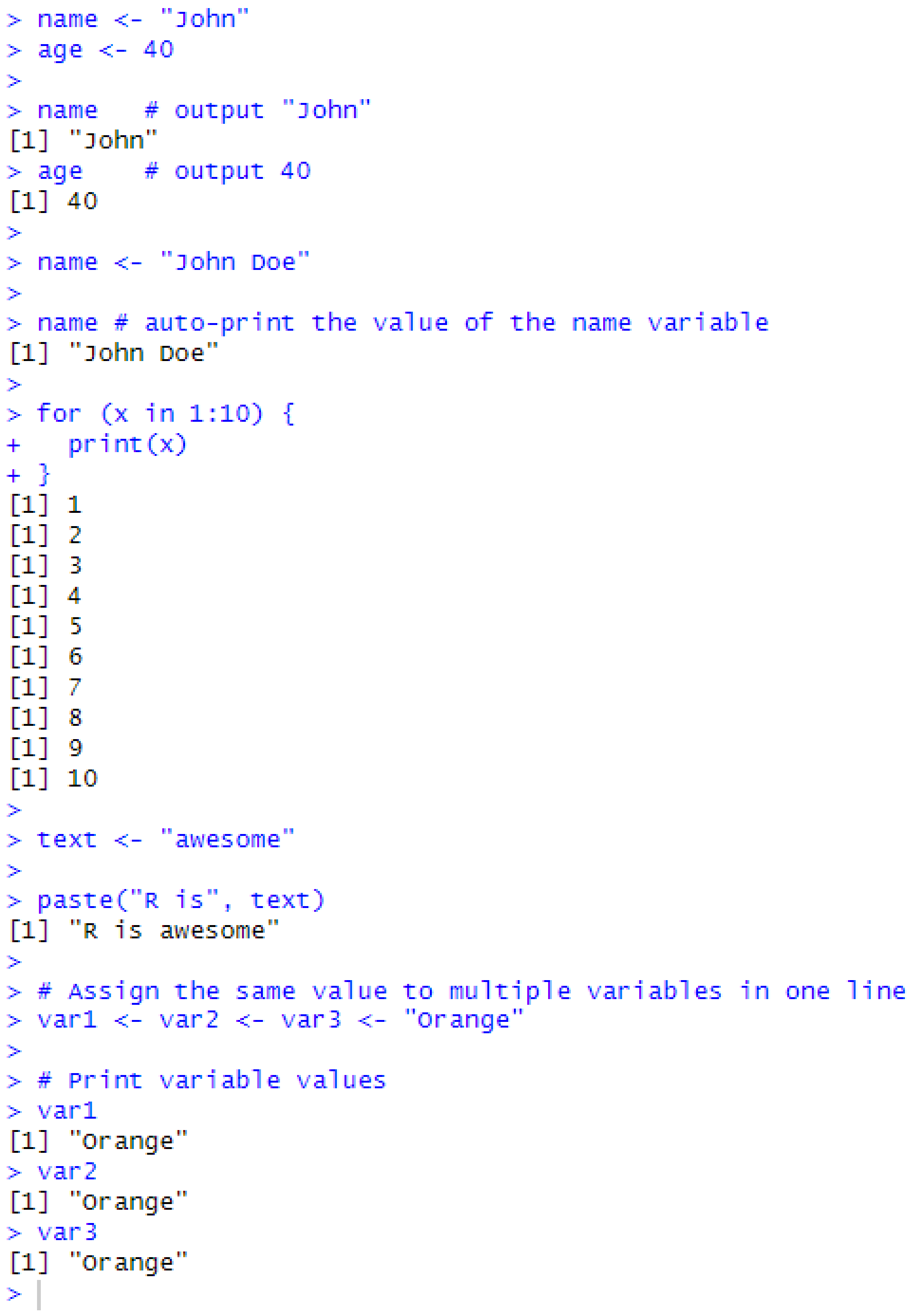
Assignment Topic: Introduction to R Programming Language.

1. **R Syntax**



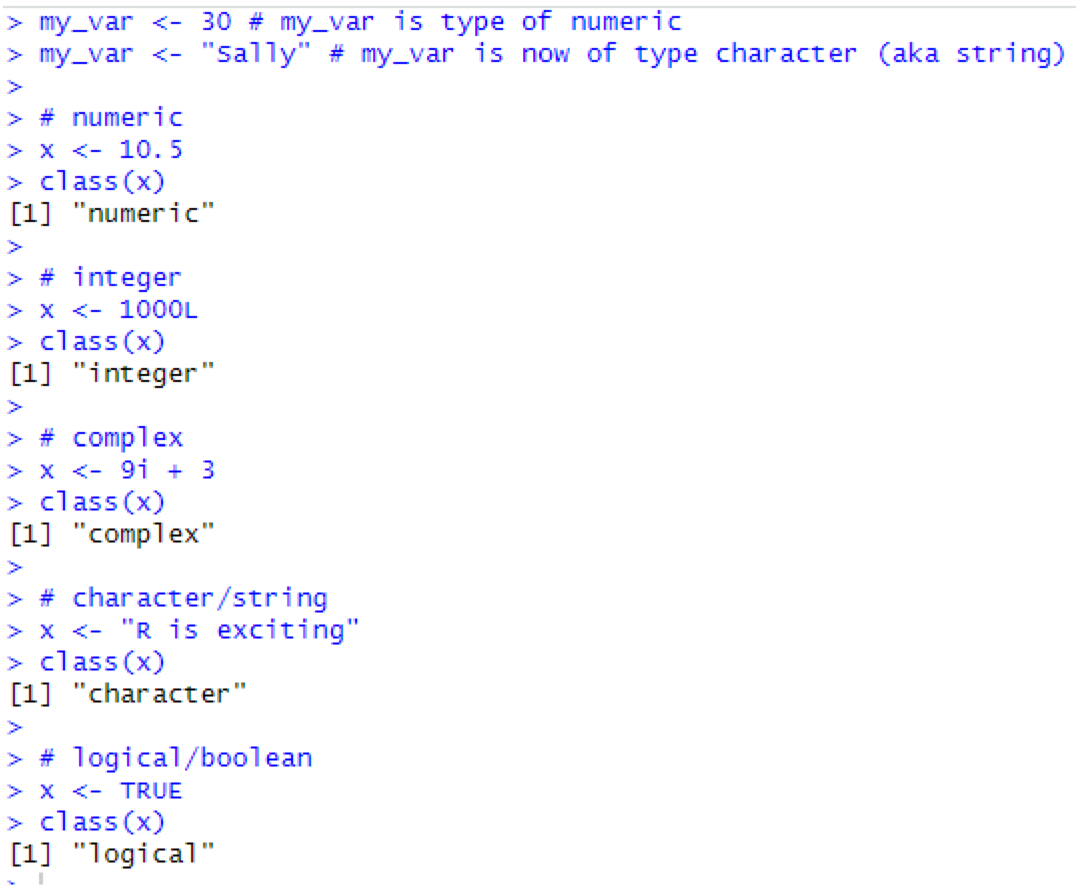
```
> 5 + 5
[1] 10
> "Hello World!"
[1] "Hello World!"
> 
```

2. **R Variables**



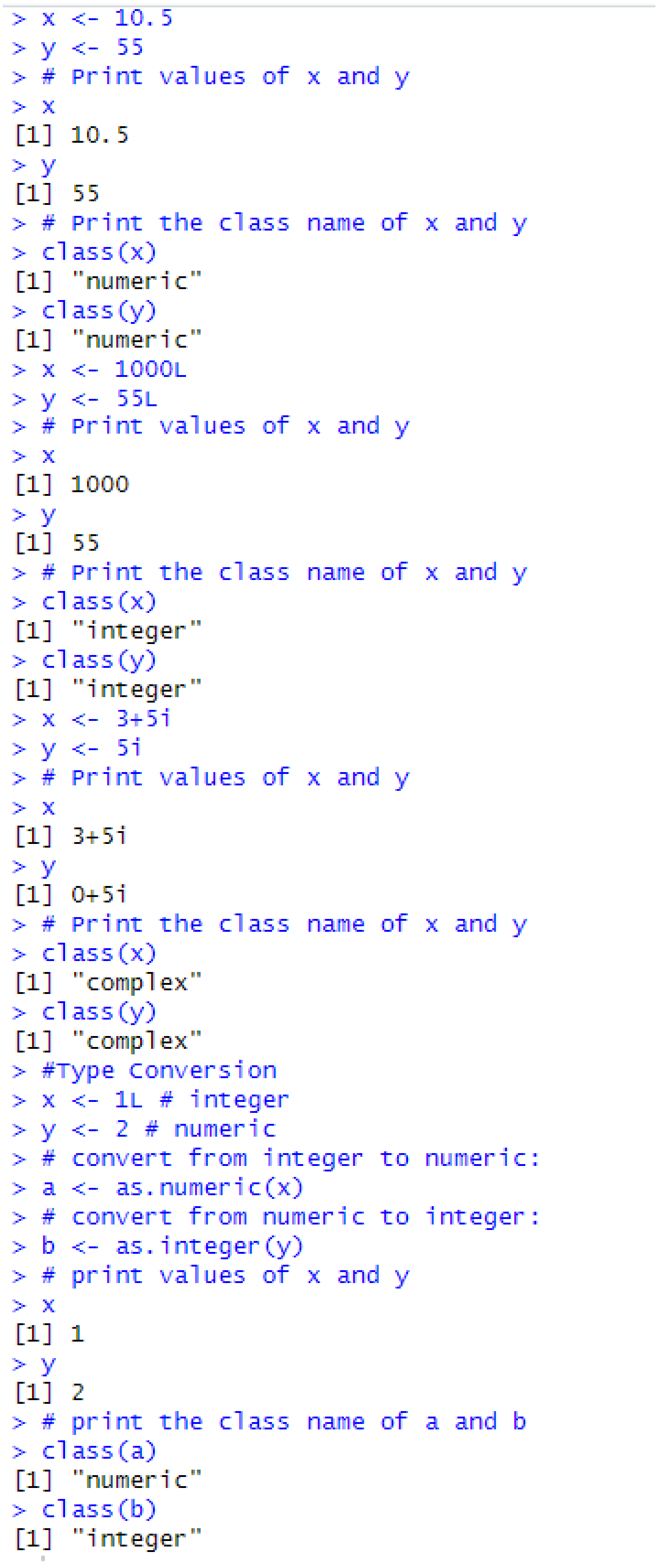
```
> name <- "John"
> age <- 40
>
> name    # output "John"
[1] "John"
> age     # output 40
[1] 40
>
> name <- "John Doe"
>
> name # auto-print the value of the name variable
[1] "John Doe"
>
> for (x in 1:10) {
+   print(x)
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
>
> text <- "awesome"
>
> paste("R is", text)
[1] "R is awesome"
>
> # Assign the same value to multiple variables in one line
> var1 <- var2 <- var3 <- "Orange"
>
> # Print variable values
> var1
[1] "Orange"
> var2
[1] "Orange"
> var3
[1] "Orange"
> 
```

3. R Data Types



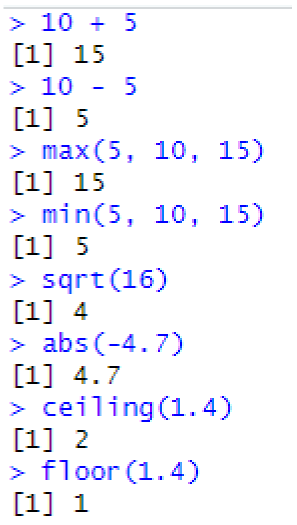
```
> my_var <- 30 # my_var is type of numeric
> my_var <- "Sally" # my_var is now of type character (aka string)
>
> # numeric
> x <- 10.5
> class(x)
[1] "numeric"
>
> # integer
> x <- 1000L
> class(x)
[1] "integer"
>
> # complex
> x <- 9i + 3
> class(x)
[1] "complex"
>
> # character/string
> x <- "R is exciting"
> class(x)
[1] "character"
>
> # logical/boolean
> x <- TRUE
> class(x)
[1] "logical"
```

4. R Numbers



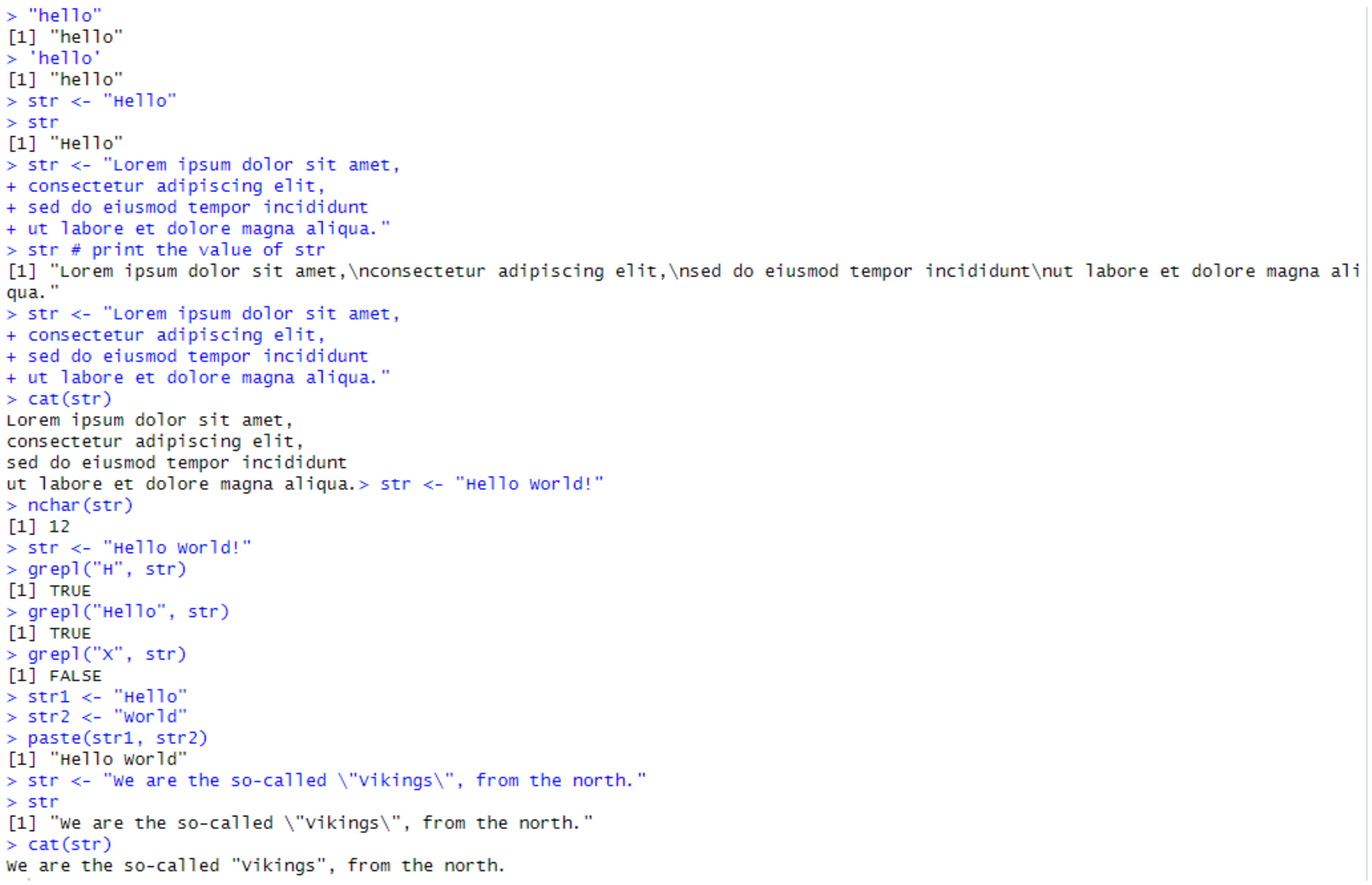
```
> x <- 10.5
> y <- 55
> # Print values of x and y
> x
[1] 10.5
> y
[1] 55
> # Print the class name of x and y
> class(x)
[1] "numeric"
> class(y)
[1] "numeric"
> x <- 1000L
> y <- 55L
> # Print values of x and y
> x
[1] 1000
> y
[1] 55
> # Print the class name of x and y
> class(x)
[1] "integer"
> class(y)
[1] "integer"
> x <- 3+5i
> y <- 5i
> # Print values of x and y
> x
[1] 3+5i
> y
[1] 0+5i
> # Print the class name of x and y
> class(x)
[1] "complex"
> class(y)
[1] "complex"
> #Type Conversion
> x <- 1L # integer
> y <- 2 # numeric
> # convert from integer to numeric:
> a <- as.numeric(x)
> # convert from numeric to integer:
> b <- as.integer(y)
> # print values of x and y
> x
[1] 1
> y
[1] 2
> # print the class name of a and b
> class(a)
[1] "numeric"
> class(b)
[1] "integer"
```

5. R Math



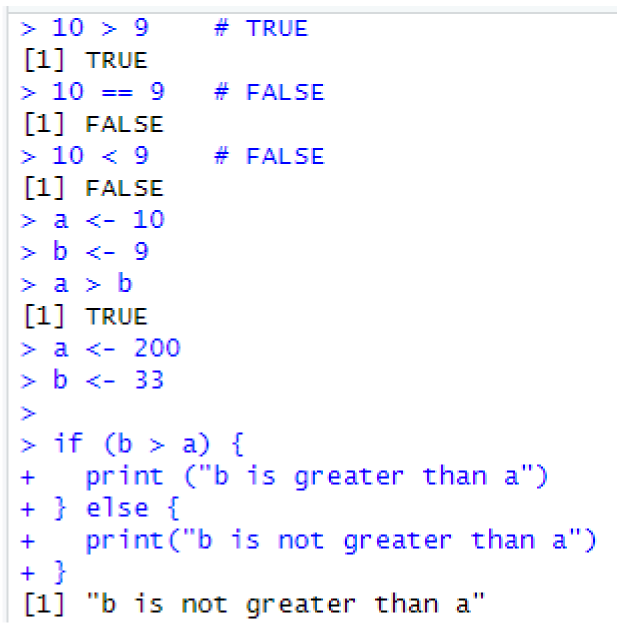
```
> 10 + 5
[1] 15
> 10 - 5
[1] 5
> max(5, 10, 15)
[1] 15
> min(5, 10, 15)
[1] 5
> sqrt(16)
[1] 4
> abs(-4.7)
[1] 4.7
> ceiling(1.4)
[1] 2
> floor(1.4)
[1] 1
```

6. R Strings



```
> "hello"
[1] "hello"
> 'hello'
[1] "hello"
> str <- "Hello"
> str
[1] "Hello"
> str <- "Lorem ipsum dolor sit amet,
+ consectetur adipiscing elit,
+ sed do eiusmod tempor incididunt
+ ut labore et dolore magna aliqua."
> str # print the value of str
[1] "Lorem ipsum dolor sit amet,\nconsectetur adipiscing elit,\nsed do eiusmod tempor incididunt\nut labore et dolore magna ali
qua."
> str <- "Lorem ipsum dolor sit amet,
+ consectetur adipiscing elit,
+ sed do eiusmod tempor incididunt
+ ut labore et dolore magna aliqua."
> cat(str)
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.> str <- "Hello World!"
> nchar(str)
[1] 12
> str <- "Hello World!"
> grepl("H", str)
[1] TRUE
> grepl("Hello", str)
[1] TRUE
> grepl("X", str)
[1] FALSE
> str1 <- "Hello"
> str2 <- "World"
> paste(str1, str2)
[1] "Hello World"
> str <- "We are the so-called \"Vikings\", from the north."
> str
[1] "We are the so-called \"Vikings\", from the north."
> cat(str)
We are the so-called "Vikings", from the north.
```

7. R Booleans / Logical Values

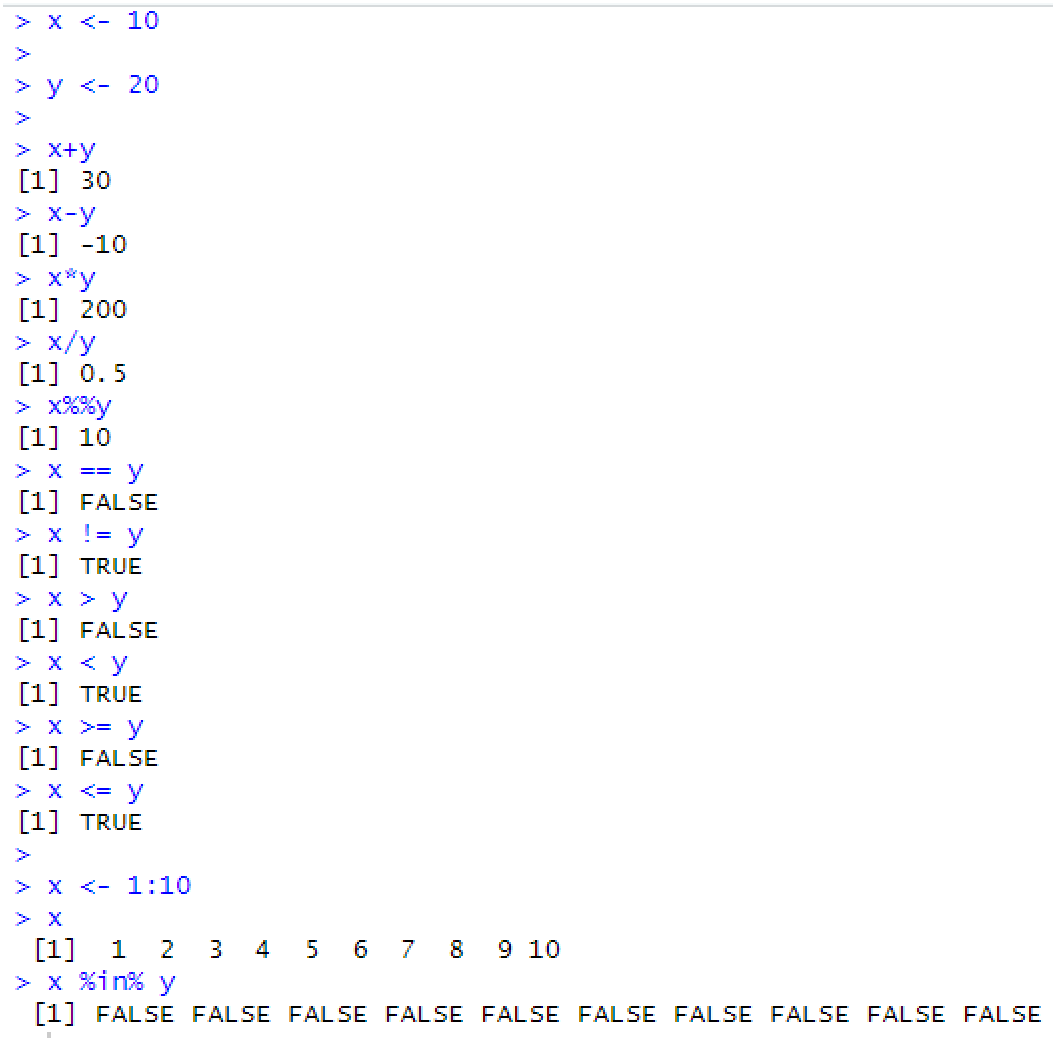


```
> 10 > 9     # TRUE
[1] TRUE
> 10 == 9    # FALSE
[1] FALSE
> 10 < 9     # FALSE
[1] FALSE
> a <- 10
> b <- 9
> a > b
[1] TRUE
> a <- 200
> b <- 33
>
> if (b > a) {
+   print ("b is greater than a")
+ } else {
+   print("b is not greater than a")
+ }
[1] "b is not greater than a"
```

8. R Operators

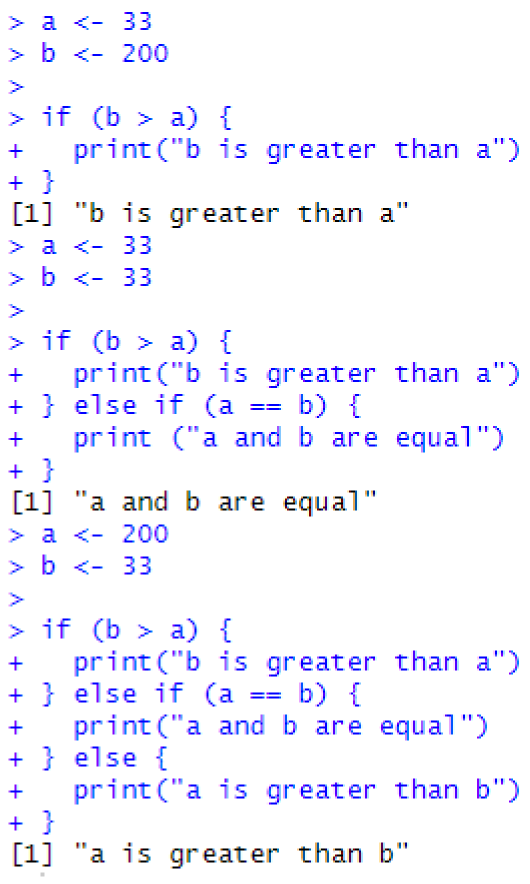
R divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Miscellaneous operators



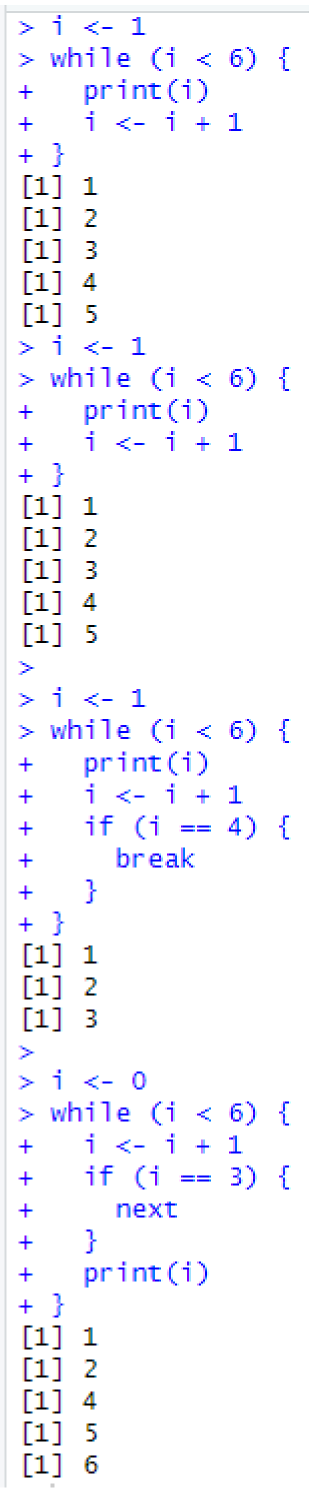
```
> x <- 10
>
> y <- 20
>
> x+y
[1] 30
> x-y
[1] -10
> x*y
[1] 200
> x/y
[1] 0.5
> x%%y
[1] 10
> x == y
[1] FALSE
> x != y
[1] TRUE
> x > y
[1] FALSE
> x < y
[1] TRUE
> x >= y
[1] FALSE
> x <= y
[1] TRUE
>
> x <- 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> x %in% y
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

9. R if…………….else



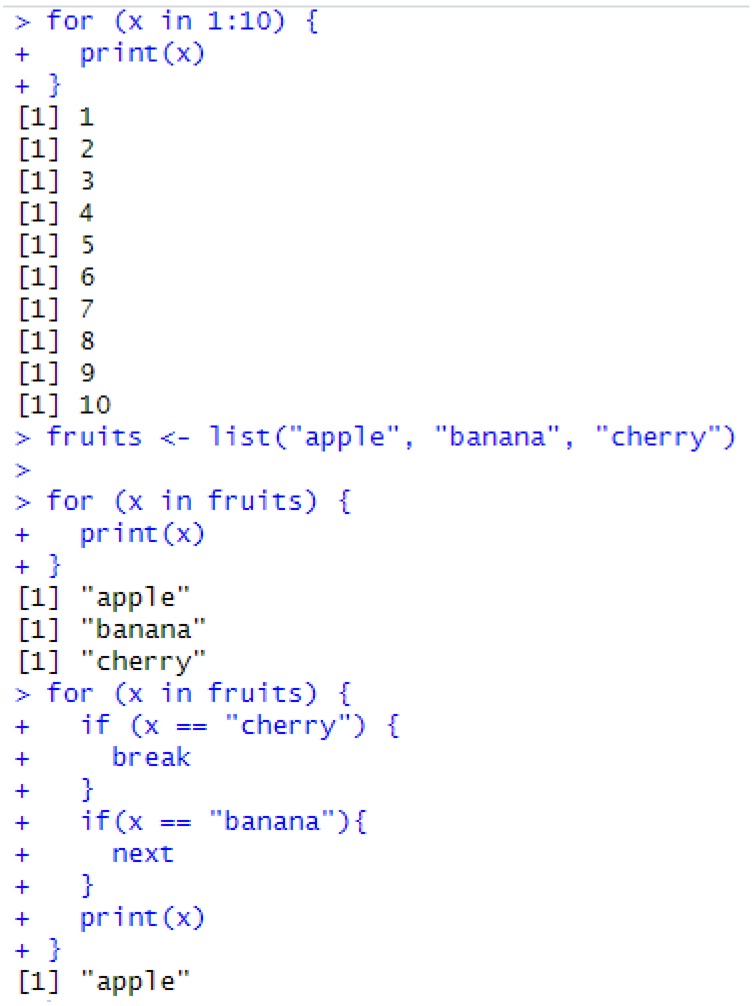
```
> a <- 33
> b <- 200
>
> if (b > a) {
+   print("b is greater than a")
+ }
[1] "b is greater than a"
> a <- 33
> b <- 33
>
> if (b > a) {
+   print("b is greater than a")
+ } else if (a == b) {
+   print ("a and b are equal")
+ }
[1] "a and b are equal"
> a <- 200
> b <- 33
>
> if (b > a) {
+   print("b is greater than a")
+ } else if (a == b) {
+   print("a and b are equal")
+ } else {
+   print("a is greater than b")
+ }
[1] "a is greater than b"
```

10. R while loop



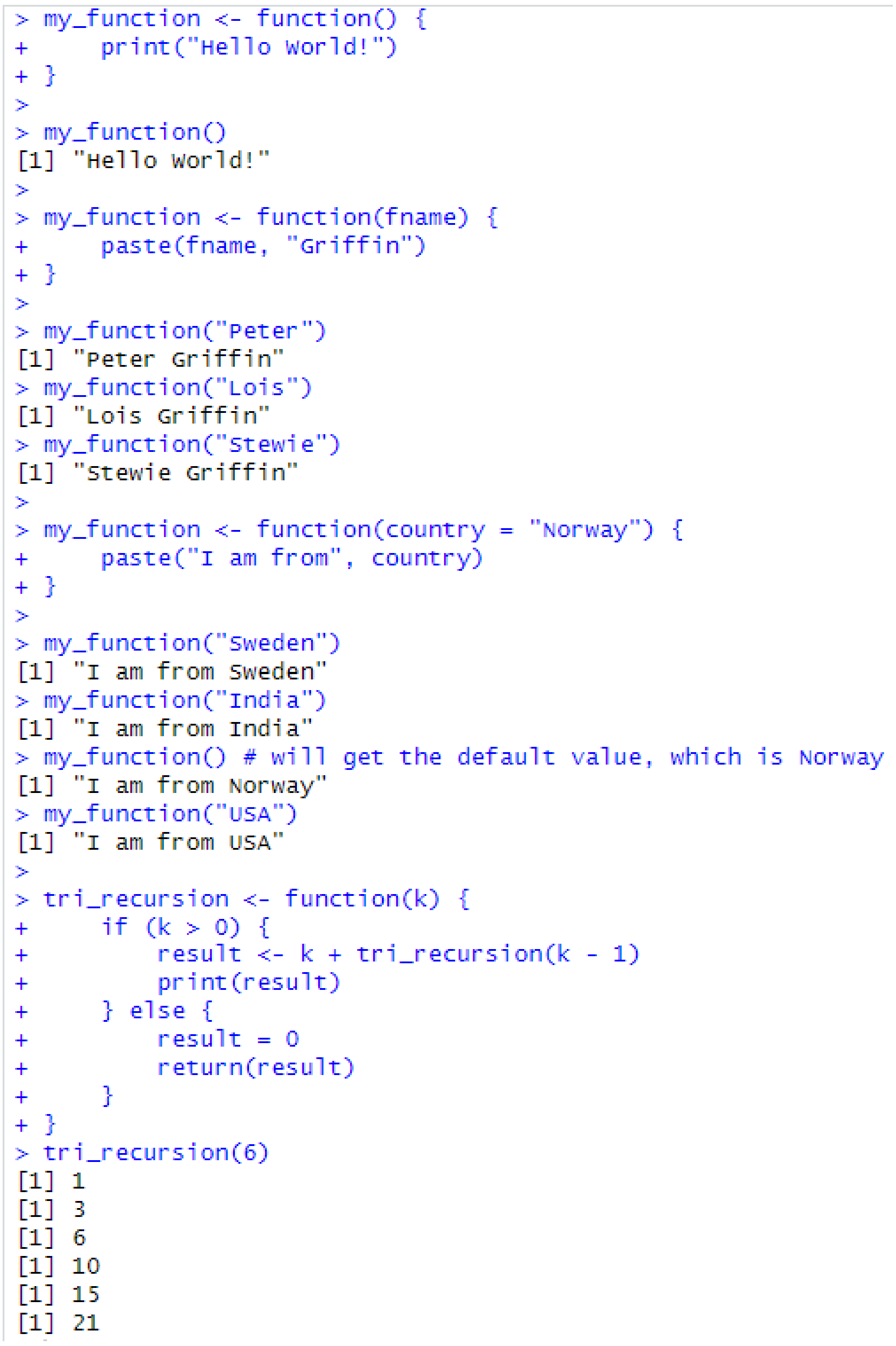
```
> i <- 1
> while (i < 6) {
+   print(i)
+   i <- i + 1
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
> i <- 1
> while (i < 6) {
+   print(i)
+   i <- i + 1
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
>
> i <- 1
> while (i < 6) {
+   print(i)
+   i <- i + 1
+   if (i == 4) {
+     break
+   }
+ }
[1] 1
[1] 2
[1] 3
>
> i <- 0
> while (i < 6) {
+   i <- i + 1
+   if (i == 3) {
+     next
+   }
+   print(i)
+ }
[1] 1
[1] 2
[1] 4
[1] 5
[1] 6
```

11. R for loops



```
> for (x in 1:10) {
+   print(x)
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
> fruits <- list("apple", "banana", "cherry")
>
> for (x in fruits) {
+   print(x)
+ }
[1] "apple"
[1] "banana"
[1] "cherry"
> for (x in fruits) {
+   if (x == "cherry") {
+     break
+   }
+   if(x == "banana"){
+     next
+   }
+   print(x)
+ }
[1] "apple"
```

12. R Functions



```
> my_function <- function() {
+     print("Hello World!")
+ }
>
> my_function()
[1] "Hello World!"
>
> my_function <- function(fname) {
+     paste(fname, "Griffin")
+ }
>
> my_function("Peter")
[1] "Peter Griffin"
> my_function("Lois")
[1] "Lois Griffin"
> my_function("Stewie")
[1] "Stewie Griffin"
>
> my_function <- function(country = "Norway") {
+     paste("I am from", country)
+ }
>
> my_function("Sweden")
[1] "I am from Sweden"
> my_function("India")
[1] "I am from India"
> my_function() # will get the default value, which is Norway
[1] "I am from Norway"
> my_function("USA")
[1] "I am from USA"
>
> tri_recursion <- function(k) {
+     if (k > 0) {
+         result <- k + tri_recursion(k - 1)
+         print(result)
+     } else {
+         result = 0
+         return(result)
+     }
+ }
> tri_recursion(6)
[1] 1
[1] 3
[1] 6
[1] 10
[1] 15
[1] 21
```

R Data Structures.

1. **R Vectors**

Source Code and Output:

```
> # Vector of strings
> fruits <- c("banana", "apple", "orange")
>
> # Print fruits
> fruits
[1] "banana" "apple"  "orange"
> # Vector of numerical values
> numbers <- c(1, 2, 3)
>
> # Print numbers
> numbers
[1] 1 2 3
> # Vector with numerical decimals in a sequence
> numbers1 <- 1.5:6.5
> numbers1
[1] 1.5 2.5 3.5 4.5 5.5 6.5
>
> # Vector with numerical decimals in a sequence where the last element is not used
> numbers2 <- 1.5:6.3
> numbers2
[1] 1.5 2.5 3.5 4.5 5.5
> fruits <- c("banana", "apple", "orange", "mango", "lemon")
> numbers <- c(13, 3, 5, 7, 20, 2)
>
> sort(fruits)  # Sort a string
[1] "apple"  "banana" "lemon"  "mango"  "orange"
> sort(numbers) # Sort numbers
[1]  2  3  5  7 13 20
> fruits <- c("banana", "apple", "orange")
>
> # Access the first item (banana)
> fruits[1]
[1] "banana"
> fruits <- c("banana", "apple", "orange", "mango", "lemon")
>
> # Change "banana" to "pear"
> fruits[1] <- "pear"
>
> # Print fruits
> fruits
[1] "pear"   "apple"  "orange" "mango"  "lemon"
> numbers <- seq(from = 0, to = 100, by = 20)
>
> numbers
[1]   0  20  40  60  80 100
```

2. **R Lists.**

Source Code and Output:

```
> # List of strings
> thislist <- list("apple", "banana", "cherry")
>
> # Print the list
> thislist
[[1]]
[1] "apple"

[[2]]
[1] "banana"

[[3]]
[1] "cherry"

> thislist <- list("apple", "banana", "cherry")
>
> thislist[1]
[[1]]
[1] "apple"

> thislist <- list("apple", "banana", "cherry")
> thislist[1] <- "blackcurrant"
>
> # Print the updated list
> thislist
[[1]]
[1] "blackcurrant"

[[2]]
[1] "banana"

[[3]]
[1] "cherry"

> thislist <- list("apple", "banana", "cherry")
>
> length(thislist)
[1] 3
> thislist <- list("apple", "banana", "cherry")
>
> "apple" %in% thislist
[1] TRUE
> thislist <- list("apple", "banana", "cherry")
>
> append(thislist, "orange")
[[1]]
[1] "apple"

[[2]]
[1] "banana"

[[3]]
[1] "cherry"

[[4]]
[1] "orange"

> thislist <- list("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
>
> (thislist)[2:5]
[[1]]
[1] "banana"

[[2]]
[1] "cherry"

[[3]]
[1] "orange"

[[4]]
```

```
[1] "kiwi"

> thislist <- list("apple", "banana", "cherry")
>
> for (x in thislist) {
+     print(x)
+ }
[1] "apple"
[1] "banana"
[1] "cherry"
```

3. R Matrices

Source Code and Output:

```
> # Create a matrix
> thismatrix <- matrix(c(1,2,3,4,5,6), nrow = 3, ncol = 2)
>
> # Print the matrix
> thismatrix
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> thismatrix <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2, ncol = 2)
>
> thismatrix[1, 2]
[1] "cherry"
> thismatrix <- matrix(c("apple", "banana", "cherry", "orange","grape", "pineapple", "
pear", "melon", "fig"), nrow = 3, ncol = 3)
>
> thismatrix[c(1,2),]
     [,1]     [,2]     [,3]
[1,] "apple"  "orange" "pear"
[2,] "banana" "grape"  "melon"
> thismatrix <- matrix(c("apple", "banana", "cherry", "orange","grape", "pineapple", "
pear", "melon", "fig"), nrow = 3, ncol = 3)
>
> newmatrix <- cbind(thismatrix, c("strawberry", "blueberry", "raspberry"))
>
> # Print the new matrix
> newmatrix
     [,1]     [,2]        [,3]    [,4]
[1,] "apple"  "orange"    "pear"  "strawberry"
[2,] "banana" "grape"     "melon" "blueberry"
[3,] "cherry" "pineapple" "fig"   "raspberry"
> thismatrix <- matrix(c("apple", "banana", "cherry", "orange", "mango", "pineapple"),
nrow = 3, ncol =2)
>
> #Remove the first row and the first column
> thismatrix <- thismatrix[-c(1), -c(1)]
>
> thismatrix
[1] "mango"     "pineapple"
> thismatrix <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2, ncol = 2)
>
> for (rows in 1:nrow(thismatrix)) {
+     for (columns in 1:ncol(thismatrix)) {
+         print(thismatrix[rows, columns])
+     }
+ }
[1] "apple"
[1] "cherry"
[1] "banana"
[1] "orange"
```

4. R Arrays

Source Code and Output:

```
> # An array with one dimension with values ranging from 1 to 24
> thisarray <- c(1:24)
> thisarray
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
>
> # An array with more than one dimension
> multiarray <- array(thisarray, dim = c(4, 3, 2))
> multiarray
, , 1

     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12

, , 2

     [,1] [,2] [,3]
[1,]   13   17   21
[2,]   14   18   22
[3,]   15   19   23
[4,]   16   20   24

> thisarray <- c(1:24)
> multiarray <- array(thisarray, dim = c(4, 3, 2))
>
> multiarray[2, 3, 2]
[1] 22
> thisarray <- c(1:24)
>
> # Access all the items from the first row from matrix one
> multiarray <- array(thisarray, dim = c(4, 3, 2))
> multiarray[c(1),,1]
[1] 1 5 9
>
> # Access all the items from the first column from matrix one
> multiarray <- array(thisarray, dim = c(4, 3, 2))
> multiarray[,c(1),1]
[1] 1 2 3 4
> for(x in multiarray){
+     print(x)
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
[1] 11
[1] 12
[1] 13
[1] 14
[1] 15
[1] 16
[1] 17
[1] 18
[1] 19
[1] 20
[1] 21
[1] 22
[1] 23
[1] 24
```

5. R Data Frames

Source Code and Output:

```
> # Create a data frame
> Data_Frame <- data.frame (
+     Training = c("Strength", "Stamina", "Other"),
+     Pulse = c(100, 150, 120),
+     Duration = c(60, 30, 45)
+ )
>
> # Print the data frame
> Data_Frame
  Training Pulse Duration
1 Strength   100       60
2  Stamina   150       30
3    Other   120       45
> Data_Frame <- data.frame (
+     Training = c("Strength", "Stamina", "Other"),
+     Pulse = c(100, 150, 120),
+     Duration = c(60, 30, 45)
+ )
>
> Data_Frame
  Training Pulse Duration
1 Strength   100       60
2  Stamina   150       30
3    Other   120       45
>
> summary(Data_Frame)
   Training             Pulse          Duration
 Length:3           Min.   :100.0   Min.   :30.0
 Class :character   1st Qu.:110.0   1st Qu.:37.5
 Mode  :character   Median :120.0   Median :45.0
                    Mean   :123.3   Mean   :45.0
                    3rd Qu.:135.0   3rd Qu.:52.5
                    Max.   :150.0   Max.   :60.0
> #We can use single brackets [ ], double brackets [[ ]] or $ to access columns from a
data frame:
> Data_Frame <- data.frame (
+     Training = c("Strength", "Stamina", "Other"),
+     Pulse = c(100, 150, 120),
+     Duration = c(60, 30, 45)
+ )
>
> Data_Frame[1]
  Training
1 Strength
2  Stamina
3    Other
>
> Data_Frame[["Training"]]
[1] "Strength" "Stamina"  "Other"
>
> Data_Frame$Training
[1] "Strength" "Stamina"  "Other"
```

6. R Factors

Factors are used to categorize data. Examples of factors are:

- Demography: Male/Female
- Music: Rock, Pop, Classic, Jazz
- Training: Strength, Stamina

To create a factor, use the factor() function and add a vector as argument:

```
> # Create a factor
> music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock",
"Jazz"))
>
> # Print the factor
> music_genre
[1] Jazz    Rock    Classic Classic Pop     Jazz    Rock    Jazz
Levels: Classic Jazz Pop Rock
> music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock",
"Jazz"), levels = c("Classic", "Jazz", "Pop", "Rock", "Other"))
>
> levels(music_genre)
[1] "Classic" "Jazz"    "Pop"     "Rock"    "Other"
> music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock",
"Jazz"), levels = c("Classic", "Jazz", "Pop", "Rock", "Opera"))
>
> music_genre[3] <- "Opera"
>
> music_genre[3]
[1] Opera
Levels: Classic Jazz Pop Rock Opera
```