Introduction

- R (the language) was created in the early 1990s, by Ross Ihaka and Robert Gentleman.
- It is based upon the S language that was developed at Bell Laboratories in the 1970s.
- It is a high-level language like C#, Java, etc..,
- R is an interpreted language (sometimes called a scripting language), which means that your code doesn't need to be compiled before you run it.
- R supports a mixture of programming paradigms (At its core, it is an imperative language, but it also supports OOP, and functional programming).

Getting started

- Where to get R?

  The newest version of R and its documentation can be downloaded from http://www.r-project.org.
- Download, Packages: Select CRAN
- Set your Mirror: India (Indian Institute of Technology Madras) Select http://ftp.iitm.ac.in/cran/
- Select Download R for Windows
- Select base.
- Select Download R 3.4.2 for Windows
- Execute the R-3.4.2-win.exe with administrator privileges. Once the program is installed, run the R program by clicking on its icon

A Scientific Calculator

- R is at heart a supercharged scientific calculator, so typing commands directly into the R Console.

```
> 5+5
[1] 10

> 4-7
[1] -3

> 7*3
[1] 21

> 16/31
[1] 0.516129

> log2(32)
[1] 5
```

Variable Assignment

- We assign values to variables with the assignment operator "=".
- Just typing the variable by itself at the prompt will print out the value.
- We should note that another form of assignment operator "<-" is also in use.

```
> X = 2
[1] 2

> X <- 5
[1] 5

> X * X
[1] 25
```

Comments

- All text after the pound sign "#" within the same line is considered a comment.

```
> X = 2          # this is a comment
[1] 2


# 5 is assign to variable X
> X <- 5
[1] 5
```

Basic Data Types

- There are several basic R data types that are of frequent occurrence in routine R calculations.
  - ✓ Numeric
  - ✓ Integer
  - ✓ Complex
  - ✓ Logical
  - ✓ Character
  - ✓ Factor

Numeric

- Decimal values are called numerics in R. It is the default computational data type.
- If we assign a decimal value to a variable x as follows, x will be of numeric type.

```
> x = 10.5          # assign a decimal value
> x                 # print the value of x
[1] 10.5

> class(x)          # print the class name of x
[1] "numeric"
```

Numeric

- Furthermore, even if we assign an integer to a variable k, it is still being saved as a numeric value.

```
> k = 1
> k                 # print the value of k
[1] 1

> class(k)          # print the class name of k
[1] "numeric"
```

- The fact that k is not an integer can be confirmed with the is.integer function.

```
> is.integer(k)     # is k an integer?
 [1] FALSE
```

Integer

- In order to create an integer variable in R, we invoke the as.integer function.
- For example,

```
> y = as.integer(3)
> y                 # print the value of y
[1] 3

> class(y)          # print the class name of y
[1] "integer"

> is.integer(y)     # is y an integer?
[1] TRUE
```

Complex

- A complex value in R is defined via the pure imaginary value i.
- For example,

```
> z = 1 + 2i        # create a complex number
> z                 # print the value of z
[1] 1+2i

> class(z)          # print the class name of z
[1] "complex"
```

- The following gives an error as −1 is not a complex value.

```
> sqrt(−1)      # square root of −1
[1] NaN
```

- Warning message: In sqrt(−1) : NaNs produced

Complex

- Instead, we have to use the complex value −1 + 0i.
- For example,

```
> sqrt(−1+0i)    # square root of −1+0i
[1] 0+1i
```

- An alternative is to coerce −1 into a complex value.

```
> sqrt(as.complex(−1))
[1] 0+1i
```

Logical

- A logical value is often created via comparison between variables.
- For example,

```
> x = 1; y = 2    # sample values
> z = x > y       # is x larger than y?
> z               # print the logical value
[1] FALSE

> class(z)        # print the class name of z
[1] "logical"
```

Logical

- A Standard logical operations are "&", "|" , "!" .
- For example,

```
> u = TRUE; v = FALSE
> u & v          # u AND v
[1] FALSE

> u | v          # u OR v
[1] TRUE

> !u             # negation of u
[1] FALSE
```

Character

- A character object is used to represent string values in R. We convert objects into character values with the as.character(). For example,

```
> x = as.character(3.14)
> x              # print the character string
[1] "3.14"

> class(x)       # print the class name of x
[1] "character"

> x = as.character( “hai”)
> x              # print the character string
[1]  “hai”

> class(x)       # print the class name of x
[1]  "character"
```

Factor

- The factor data type is used to represent categorical data. (i.e. data of which the value range is a collection of codes).
- For example, to create a vector of length five of type factor do the following:

```
>sex <- c("male","male","female","male","female")
```

- The object sex is a character object. You need to transform it to factor.

```
>sex <- factor(sex)
>sex
[1] male male female male female
Levels: female male
```

- Use the function levels to see the different levels a factor variable has.

Data structures

- Before you can perform statistical analysis in R, your data has to be structured in some coherent way. To store your data R has the following structures:
  - ✓ Vector
  - ✓ Matrix
  - ✓ Array
  - ✓ Data frame
  - ✓ Time-series
  - ✓ List

Vectors

- A vector is a sequence of data elements of the same basic type.
- Members in a vector are officially called components.
- For example, Here is a vector containing three numeric values 2, 3, 5.

```
> c(2, 3, 5)
[1] 2 3 5
```

- Here is a vector of logical values.

```
> c(TRUE, FALSE, TRUE, FALSE, FALSE)
[1]  TRUE FALSE  TRUE FALSE FALSE
```

Combining Vectors

- Vectors can be combined via the function c.
- For example, Here is a vector containing three numeric values 2, 3, 5.

```
> n = c(2, 3, 5)
> s = c("aa", "bb", "cc", "dd", "ee")
> c(n, s)
[1]  "2"  "3"  "5"  "aa" "bb" "cc" "dd" "ee"
```

Vector Arithmetics

- Arithmetic operations of vectors are performed member-by-member.
- For example, Here is a vector containing three numeric values 2, 3, 5.

```
> a = c(1, 3, 5, 7)
> b = c(1, 2, 4, 8)
```

- We add a and b together, the sum would be a vector whose members are the sum of the corresponding members from a and b.

```
> a + b
[1]  2  5  9 15
```

- Similarly for subtraction, multiplication and division, we get new vectors via member wise operations.

Vector Recycling Rule

- If two vectors are of unequal length, the shorter one will be recycled in order to match the longer vector.
- For example, sum is computed by recycling values of the shorter vector.

```
> u = c(10, 20, 30)
> v = c(1, 2, 3, 4, 5, 6, 7, 8, 9)
> u + v
[1] 11 22 33 14 25 36 17 28 39
```