Introduction to Scientific Computing for Biologists

ISCB20.09 - Introduction to R

Md. Jubayer Hossain
https://jhossain.me/
jubayer@hdrobd.org

Founder
Health Data Research Organization
Lead Organizer
Scientific Computing for Biologists

01 February 2021

Section-1: Introduction to R

What is R

- R is a dialect of S(R is an implementation of the S programming language).
- R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, and is developed by the R Development Core Team.
- R is a programming language and environment commonly used in statistical computing, data analytics and scientific research.
- R is a programming language and free software environment for statistical computing and graphics supported by the R Foundation for Statistical Computing.
- The R language is widely used among statisticians and data miners for developing statistical software and data analysis.

Why R?

- R is open source and free!
  - R is free to download as it is licensed under the terms of the GNU General Public License.
  - You can look at the source to see what's happening under the hood.
  - There's more, most R packages are available under the same license so you can use them, even in commercial applications without having to call your lawyer.
- R is popular – and increasing in popularity.
- R runs on all platforms.(Windows, Linux and Mac)
- R is being used by the biggest tech giants(google, facebook, microsoft, twitter)

Applications of R

- Data Science
- Data Analysis
- Genomic Data Science
- Biological Data Analysis
- Mutational Signature Analysis
- Genomic Analysis
- Statistical Computing
- Machine Learning

R Packages for Data Analysis/Data Science

- `dplyr`
  - dplyr is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges
  - Documentation- https://dplyr.tidyverse.org/
- `ggplot2`
  - ggplot2 is a data visualization package for the statistical programming language R.
  - Created by Hadley Wickham in 2005, ggplot2 is an implementation of Leland Wilkinson's Grammar of Graphics—a general scheme for data visualization which breaks up graphs into semantic components such as scales and layers.
  - https://ggplot2.tidyverse.org/

R Packages for Data Analysis/Data Science(Cont. . .)

- Plotly
  - Plotly's R graphing library makes interactive, publication-quality graphs.
  - https://plotly.com/r/
- Leaflet
  - Leaflet is one of the most popular open-source JavaScript libraries for interactive maps.
  - It's used by websites ranging from The New York Times and The Washington Post to GitHub and Flickr, as well as GIS specialists like OpenStreetMap, Mapbox, and CartoDB.
  - https://rstudio.github.io/leaflet/
- tidyverse
  - The tidyverse is an opinionated collection of R packages designed for data science.
  - All packages share an underlying design philosophy, grammar, and data structures.
  - https://www.tidyverse.org/

R Packages for Bioinformatics/Genomic Data Science

- Bioconductor
  - Bioconductor is a free, open source and open development software project for the analysis and comprehension of genomic data generated by wet lab experiments in molecular biology.
  - Bioconductor is based primarily on the statistical R programming language, but does contain contributions in other programming languages.
  - https://www.bioconductor.org/
- seqinr
  - Exploratory data analysis and data visualization for biological sequence (DNA and protein) data.
  - https://cran.r-project.org/web/packages/seqinr/index.html
- MutatioanlPattern
  - Mutational processes leave characteristic footprints in genomic DNA.
  - https://bioconductor.org/packages/release/bioc/html/MutationalPatterns.html

Resources: Books

- R for Data Science by Roger D.Peng
- Introduction to Data Science by Rafael Irizarry
- Data Analysis for the Life Sciences by Rafael Irizarry
- Exploratory Data Analysis with R by Roger D.Peng

Resources: Blogs

- https://www.datamentor.io/r-programming/
- https://online.stat.psu.edu/stat484/
- https://online.stat.psu.edu/stat485/
- https://www.statmethods.net/index.html
- https://simplystatistics.org/
- https://www.tutorialspoint.com/r/index.htm
- https://www.rforbiologists.org/
- https://compgenomr.github.io/book/
- https://statsandr.com/
- https://rafalab.github.io/pages/harvardx.html
- https://bolt.mph.ufl.edu/software/r-phc-6055/

Alternatives to R Programming

- Python
  - Python is a very powerful high-level, object-oriented programming language with an easy-to-use and simple syntax.
  - Python is extremely popular among data scientists and researchers. Most of the packages in R have equivalent libraries in Python as well.
- SAS (Statistical Analysis System)
  - SAS is a powerful software that has been the first choice of private enterprise for their analytics needs for a long time.
- SPSS – Software Package for Statistical Analysis
  - SPSS is another popular statistical tool. It is used most commonly in the social sciences and is considered the easiest to learn among enterprise statistical tools.

Section-2: Getting Started

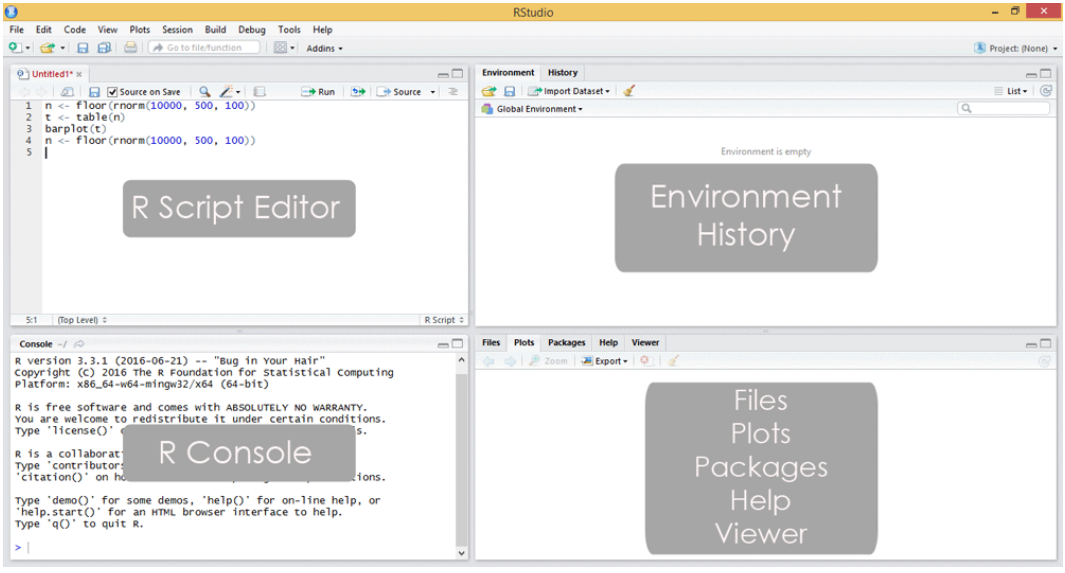
Installing R

- Installing R on Windows - https://cran.r-project.org/bin/windows/base/
- Installing R on Linux - https://cran.r-project.org/bin/linux/ubuntu/README.html
- Installing R on Mac - https://cran.r-project.org/bin/macosx/

Installing RStudio

- RStudio is an Integrated Development Environment(IDE) available for R that is built by RStudio.
- Download and Install
  https://rstudio.com/products/rstudio/download/#download

The RStudio Interface



Section-3: Variables and Reserved Keywords

Variables(Identifiers) in R

- Variables are used to **store data**, whose value can be changed according to our need.
- A variable is a name given to a memory location, which is used to store values in a computer program.
- Variables in R programming can be used to store numbers (real and complex), words, matrices, and even tables.
- R is a dynamically programmed language which means that unlike other programming languages, we do not have to declare the data type of a variable before we can use it in our program.
- Unique name given to variable (function and objects as well) is **identifier**.

Rules for writing Identifiers in R

- Identifiers can be a combination of letters, digits, period (.) and underscore (_).
- It must start with a letter or a period. If it starts with a period, it cannot be followed by a digit.
- It should not start with a number (e.g: 2x)
- It should not start with a dot followed by a number (e.g: .2x)
- It should not start with an underscore (e.g: _x)
- Reserved words in R cannot be used as identifiers(e.g: TRUE, FALSE)

Basically, there are 5 naming conventions

- alllowercase: e.g. `myname`
- period.separated: e.g. `new.name`
- underscore_separated: e.g. `my_name`
- lowerCamelCase: e.g. `myName`
- UpperCamelCase: e.g. `MyName`

Creating Variables

Using equal(=) operator

```
x = 10
```

Using leftward(<-) operator

```
y <- 15
```

Reserved Keaywords in R

- Don't use any reserved keyword as variable name. List all of reserved words in R by using (?Reserved).

```
?Reserved
```

Entering Input

At the R prompt/console we type expressions.

```
num <- 10
```

The <- symbol is the **assignment** operator.
The grammar of the language determines whether an expression is complete or not.

Evaluation

When a complete expression is entered at the R console, it is evaluated and the result of evaluated expression is returned. The result may be auto-printed.

```
x <- 10
x
```

```
[1] 10
```

```
x <- 10
print(x)
```

```
[1] 10
```

```
x <- 10
cat(x)
```

```
10
```

Comments in R

The # character indicates a comment.

```
x <- 10 # This is a comment
```

Anything to the right of the # (inlcuding the # itself) is ignored.
This is only comment character in R.
R does not support multi-line comments or comment block.

Section-4: Data Types in R

Data Types in R

R has five basic data types

- character(e.g: ‘abul’, “abul”)
- numeric(e.g: 2, 3)
- integer(e.g: 5L)
- complex(e.g: 5i)
- logical(True/False)

Numbers

- Numbers in R generally treated as numeric objects(i.e. double precision real numbers)
- If you explicitly want an integer, you need to specify the `L` suffix.
- There is also a special number `Inf` which is represents infinity; e.g. `1 / 0`
- `Inf` can be used in ordinary calculations; e.g. `1/Inf` is `0`
- The value `NaN` represents an undefined value("not a number"); e.g. `0/0`
- `NaN` can also be thought of as a missing value.

The numeric constants are

- integer(`L`)
- double
- complex(`i`)

Creating Numeric Objects

```
# Create numeric object
n <- 5
```

```
# Check type of object
typeof(n)
```

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

```
# Create an integer type object
i = 5L
```

```
# Check type of object
typeof(i)
```

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

Creating Numeric Objects(Cont. . .)

```
# Create a double type object
d = 22
```

```
# Check type of object
typeof(d)
```

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

```
# Create a complex type object
c = 4i
```

```
# Check type of object
typeof(c)
```

```
[1] "complex"
```

Decimal vs Double vs Float

The Decimal, Double, and Float variable types are different in the way that they store the values. Precision is the main difference where float is a single precision (32 bit) floating point data type, double is a double precision (64 bit) floating point data type and decimal is a 128-bit floating point data type.

- Float - 32 bit (7 digits)
- Double - 64 bit (15-16 digits)
- Decimal - 128 bit (28-29 significant digits)

Characters

Character constants can be represented using either single quotes (') or double quotes ("") as delimiters.

```
# Create a character type object
char = "Hello"
```

```
# Check type of object
typeof(char)
```

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

Atributes

R objects can have attributes

- names, dimnames
- dimensions(e.g. matrices, arrays)
- class
- length
- other user-defined attributes/metadata

Attributes of an object can be accessed using the `attributes()` function.

Built-in Constants in R

```
pi # value of pi
```

```
[1] 3.141593
```

```
LETTERS
```

```
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R"
[20] "T" "U" "V" "W" "X" "Y" "Z"
```

```
month.name
```

```
 [1] "January"   "February"  "March"     "April"     "May"       "June"
 [7] "July"      "August"    "September" "October"   "November"  "December"
```

Section-5: Operators

Operators in R

- R has many operators to carry out different mathematical and logical operations.
- Operators in R can mainly be classified into the following categories.
  1. Arithmetic Operators
  2. Relational Operators
  3. Logical Operators
  4. Assignment Operators

Arithmetic Operators

```
x <- 10
y <- 2
```

```
# Addition
x+y
```

```
[1] 12
```

```
# Subtraction
2-5
```

```
[1] -3
```

Arithmetic Operators(Cont. . .)

```
# Multiplication
2 * 5
```

```
[1] 10
```

```
# Division
2 / 5
```

```
[1] 0.4
```

```
# Exponent
2 ^ 5
```

```
[1] 32
```

Arithmetic Operators(Cont. . .)

```
# Modulus(Remainder from division)
2 %% 5
```

```
[1] 2
```

```
# Integer Division
2 %/% 5
```

```
[1] 0
```

Logical Operators

```
# Logical NOT(!)
! TRUE
```

```
[1] FALSE
```

```
! FALSE
```

```
[1] TRUE
```

Logical Operators(Cont. . .)

```
# Logical AND(&&)
TRUE && TRUE
```

```
[1] TRUE
```

```
TRUE && FALSE
```

```
[1] FALSE
```

```
FALSE && FALSE
```

```
[1] FALSE
```

Logical Operators(Cont. . .)

```
# Logical OR(|)
TRUE | TRUE
```

```
[1] TRUE
```

```
TRUE | FALSE
```

```
[1] TRUE
```

```
FALSE | FALSE
```

```
[1] FALSE
```

Relational Operators

```
x <- 10
y <- 5
# Equality
x == y
```

```
[1] FALSE
```

```
# Inequality
x != y
```

```
[1] TRUE
```

```
# Less Than
x < y
```

```
[1] FALSE
```

Relational Operators(Cont. . .)

```
# Greater Than
x > y
```

```
[1] TRUE
```

```
# Less or Equal
x <= y
```

```
[1] FALSE
```

```
# Greater or Equal
x >= y
```

```
[1] TRUE
```

Section-6: Data Structures in R

What is Data Structures?

In computer science, a data structure is a data organization, management, and storage format that enables efficient access and modification. More precisely, a data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data.
(https://en.wikipedia.org/wiki/Data_structure)

Vector

- Vector is a basic data structure in R.
- It contains element of the same type.
- The data types can be logical, integer, double, character, and complex.
- A vector's type can be checked with the `typeof()` function.

Creating Vectors

- Vectors are generally created using the c() function.
- To create Vectors of consecutive number, the : operator very helpful.
- More complex sequence can be created using seq() function.

```
# Create a vector using c() function
v1 <- c(1, 2, 3, 4, 5, 6,7)
# print v1
v1
```

```
[1] 1 2 3 4 5 6 7
```

```
# Create a vector using : operator
v2 <- 1:10
# print v2
v2
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

Creating Vectors(Cont. . .)

```r
# Create a vector using seq(start, stop, step) function
v3 <- seq(1, 20, 2)
# print v3
v3
```

```
 [1]  1  3  5  7  9 11 13 15 17 19
```

```r
# Create a vector using seq(start, stop, by = step) function
v4 <- seq(1, 10, by = .5)
# print v3
v4
```

```
 [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5
[16]  8.5  9.0  9.5 10.0
```

Creating Vectors(Cont. . .)

```r
# Numeric Vector
v <- c(0.5, 0.6)
# Logical Vector
v <- c(TRUE, FALSE)
# Logical Vector
v <- c(T, F)
# Character Vector
v <- c("a", "b")
# Integer Vector
v <- 1:10
# Complex Vector
v <- c(1+0i, 2+0i)
```

Mixing Objects

```r
# Character
x <- c(1.7, "a")
# Numeric
y <- c(TRUE, 2)
# Character
z <- c("a", TRUE)
```

Matrix

- Matrix is a two dimensional data structure in R programming.
- Matrix is similar to **vector** but additionally contains the dimension attributes.
- All attributes of an object can be checked by `attributes()` function.
- Dimention can be checked by directly with the `dim()` function.
- We can check if a variable is a matrix or not with the `class()` function.

Creatinig Matrix

- Matrix can be created using the `matrix()` function.
- Dimension of the matrix can be defined by passing appropriate value for arguments `nrow` and `ncol`.

```
# Create a matrix using matrix function
mat <- matrix(1:9, nrow = 3, ncol = 3)
# print matrix
mat
```

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

Creatinig Matrix(Cont. . .)

```r
# Create a matrix using matrix function: only one dimension
mat <- matrix(1:9, nrow = 3)
# print matrix
mat
```

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

Creatinig Matrix(Cont. . .)

```r
# Create a matrix using matrix function: filling by row-wise
mat <- matrix(1:9, nrow = 3, byrow = TRUE)
# print matrix
mat
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

Creatinig Matrix(Cont. . .)

```r
# Create a matrix using matrix function: dimension names
mat <- matrix(1:9, nrow = 3, dimnames = list(c("X", "Y", "Z"),
                                             c("A","B","C")))
# print matrix
mat
```

```
  A B C
X 1 4 7
Y 2 5 8
Z 3 6 9
```

Colnames, Rownames and Dimension of Matrix

```r
# Create a matrix using matrix function
mat <- matrix(1:9, nrow = 3, dimnames = list(c("X", "Y", "Z"),
                                             c("A","B","C")))
# Column Names
colnames(mat)
```

```
[1] "A" "B" "C"
```

```r
# Row Names
rownames(mat)
```

```
[1] "X" "Y" "Z"
```

```r
# Dimension
dim(mat)
```

```
[1] 3 3
```

List

- List is a data structure having components of mixed data types.
- A `vector` having all elements of the same type is called atomic vector but a vector having elements of different type is called list.
- We can check if it's a list with `typeof()` function and find its length using `length()` function.

Creating List

List can be created using the `list()` function.

```
# Create a list
L = list(1, "a", TRUE, 1+3i)
# Print list
L
```

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

[[2]]
[1] "a"

[[3]]
[1] TRUE

[[4]]
```

Factors

- Factor is a data structure use for fields that takes only predefined, finite number of values(categorical values)
- Factors are used to represent categorical data and can be ordered and unordered.

Creating Factors

```r
# Create a factor using factor() function
f <- factor(c("yes", "no", "yes", "no"))
# Print factor
f
```

```
[1] yes no  yes no
Levels: no yes
```

```r
# Check levels
levels(f)
```

```
[1] "no"  "yes"
```

Data Frame

- Data frame is a two dimensional data structure in R.
- It is a special case of a `list` which has each component of equal length.
- Each component from the column and contents of the component from rows.

Create Data Frame

```r
# Create a Data Frame
df <- data.frame("Age" = c(21, 22, 14, 15, 16, 23),
      "Name" = c("Jim","Tim", "Babul", "Bithi", "Abul", "Akhi"),
      "Married" = factor(c("yes", "no", "yes", "no", "yes", "yes")))
# Print Data Frame
df
```

```
  Age  Name Married
1  21   Jim     yes
2  22   Tim      no
3  14 Babul     yes
4  15 Bithi      no
5  16  Abul     yes
6  23  Akhi     yes
```

Exploring Data Frame

```r
# Dimension
dim(df)
```

```
[1] 6 3
```

```r
# Data Structures
str(df)
```

```
'data.frame':   6 obs. of  3 variables:
 $ Age    : num  21 22 14 15 16 23
 $ Name   : chr  "Jim" "Tim" "Babul" "Bithi" ...
 $ Married: Factor w/ 2 levels "no","yes": 2 1 2 1 2 2
```

Exploring Data Frame(Cont..)

```
# Summary
summary(df)
```

```
      Age            Name              Married
 Min.   :14.00   Length:6           no :2
 1st Qu.:15.25   Class :character   yes:4
 Median :18.50   Mode  :character
 Mean   :18.50
 3rd Qu.:21.75
 Max.   :23.00
```

```
# Colnames
names(df)
```

```
[1] "Age"     "Name"    "Married"
```

Exploring Data Frame(Cont..)

```
# Accessing Columns
df$Name
```

```
[1] "Jim"   "Tim"   "Babul" "Bithi" "Abul"  "Akhi"
```

Section-7: Subsetting R Objects

Subsetting a Vector

```r
# Create a vector
x <- c(1, 11, 111)

# Extract first element
x[1]
```

```
[1] 1
```

```r
# Extract second element
x[2]
```

```
[1] 11
```

Subsetting a Vector(Cont. . .)

```r
# Create a vector
x <- c("a", "b", "c", "d", "e")

# Extract data using :
x[1:3]
```

```
[1] "a" "b" "c"
```

```r
# Extract data using c() function
x[c(1, 3, 4)]
```

```
[1] "a" "c" "d"
```

Subsetting a Vector(Cont. . .)

```r
# Create a vector
x <- c(1, 20, 21, 11, 23, 40, 42)

# Extract data using relational operator
x[x > 32]
```

```
[1] 40 42
```

Subsetting List

```
L = list(x = 1:5, y=0.5)

# Extract first element
L[[1]]
```

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

```
# Extract first element
L[[2]]
```

```
[1] 0.5
```

```
# Access elements
L$x
```

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

Subsetting Matrix

```
m <- matrix(1:6, 2, 3)
m
```

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

```
# Extract first row
m[1, ]
```

```
[1] 1 3 5
```

```
# Extract first element from first row
m[1, 1]
```

```
[1] 1
```

Subsetting Matrix(Cont. . .)

```
m <- matrix(1:6, 2, 3)
m
```

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

```
# Extract second column
m[, 2]
```

```
[1] 3 4
```

Section-8: Control Structures

if statement

```
if(condition) {
  # do something
}
```

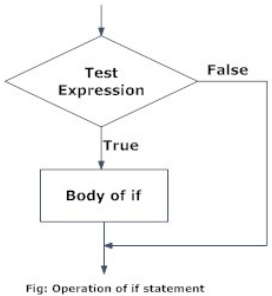


Figure 2: https://www.datamentor.io/r-programming/if-else-statement/

Write a program to check a number is a positive

```
num = 10
if (num > 0) {
    print("positive number")
}
```

```
[1] "positive number"
```

`if..else` statement

```
if(condition) {
  # do something
} else {
  # do something
}
```

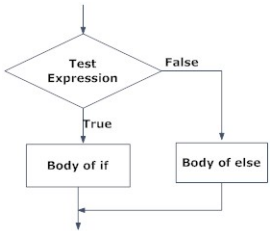


Figure 3: https://www.datamentor.io/r-programming/if-else-statement/

Write a program to check a number is a positive or negative

```
num = 10
if (num > 0) {
    print("positive number")
} else {
    print("Negative number")
}
```

```
[1] "positive number"
```

if..else if...else statement

```
if(condition-1) {
  # do something
} else if(condition-2){
  # do something
} else if(condition-3) {
  # do something
} else {
  # do something
}
```

Write a program to check your bmi and health condition

```
bmi = 18.5
if(bmi < 18.5) {
  print("underweight")
} else if(bmi >= 18.5 && bmi < 25){
  print("normal")
} else if(bmi <= 25 && bmi < 30 ){
  print("overweight")
} else {
  print("obese")
}
```

```
[1] "normal"
```

`ifelse` function

- Vectors form the basic building block of R programming.
- Most of the functions in R take vector as input and output resultant vector.
- This vectorization of code, will be much faster than applying the same function to each element of the vector indivodually.
- Similar concept, there is a vector equivalent form of the `if...else` statement in R, the `ifelse()` function.

```
ifelse(condition, x, y)
```

Check even and odd with ifelse function

```
a = 10
ifelse(a %% 2 == 0, "even", "odd")
```

```
[1] "even"
```

Check even or odd on a sequence of object

```r
b = c(22, 12, 23, 24, 21, 28)
ifelse(b %% 2 == 0, "even", "odd")
```

```
[1] "even" "even" "odd"  "even" "odd"  "even"
```

for loop

```
for (val in sequence){
  # do something
}
```

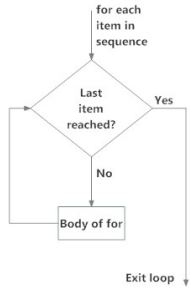


Fig: operation of for loop

Figure 4: https://www.datamentor.io/r-programming/if-else-statement/

Print first 10 (1 to 10) natural numbers

```
for (i in 1:10) {
    print(i)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

Print first 10 (1 to 10) natural numbers in reverse order

```
for (i in 10:1) {
    print(i)
}
```

```
[1] 10
[1] 9
[1] 8
[1] 7
[1] 6
[1] 5
[1] 4
[1] 3
[1] 2
[1] 1
```

Iteration of a sequence

```
L = c(2, 1, 4, 5, 6, 7)
for (val in L) {
    print(val)
}
```

```
[1] 2
[1] 1
[1] 4
[1] 5
[1] 6
[1] 7
```

Iteration of a sequence with condition

```
L = c(2, 1, 4, 5, 6, 7)
for (val in L) {
    if(val %% 2 == 0) {
        print(val)
    }
}
```

```
[1] 2
[1] 4
[1] 6
```

`while` loop

```
while(condition){
  # do something
}
```

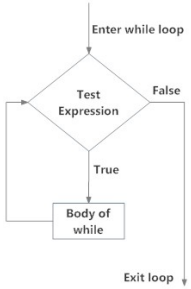


Fig: operation of while loop

Figure 5: https://www.datamentor.io/r-programming/if-else-statement/

Print first 10 natural numbers

```
i = 1
while (i <= 10) {
    print(i)
    i = i + 1
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

Print first 10 natural numbers in reverse order

```
i = 10
while(i > 0) {
    print(i)
    i = i -1
}
```

```
[1] 10
[1] 9
[1] 8
[1] 7
[1] 6
[1] 5
[1] 4
[1] 3
[1] 2
[1] 1
```

`break` statement

```
if(condition){
  break
}
```

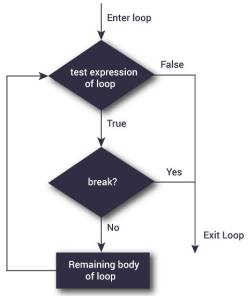


Figure 6: https://www.datamentor.io/r-programming/if-else-statement/

Example of break statement using for loop

```r
x <- 1:10
for (i in x) {
    if(i == 5) {
        break
    }
    print(i)
}
```

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

Example of break statement using while loop

```
i = 0
while (i <= 10) {
    i = i + 1
    if(i == 5) {
        break
    }
    print(i)
}
```

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

next statement

```
if(condition){
  next
}
```

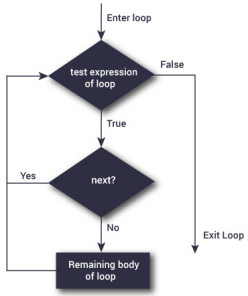


Figure 7: https://www.datamentor.io/r-programming/if-else-statement/

Example of next statement using for loop

```
for (val in 1:10) {
    if (val == 5) {
        next
    }
    print(val)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

Example of next statement using while loop

```
i = 0
while(i <= 5) {
    i = i + 1
    if(i == 2) {
        next
    }
    print(i)
}
```

```
[1] 1
[1] 3
[1] 4
[1] 5
[1] 6
```

`repeat` loop

```
repeat {
  # do something
}
```

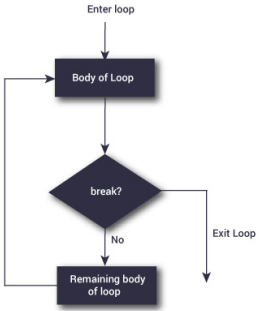


Figure 8: https://www.datamentor.io/r-programming/if-else-statement/

Example of repeat loop

```
x <- 1
repeat {
print(x)
x = x+1
if (x == 6){
break
}
}
```

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

Section-9: Functions

Functions in R

- A function is a set of statements organized together to perform a specific task.
- R has a large number of in-built functions and the user can create their own functions.
- In R, a function is an object so the R interpreter is able to pass control to the function, along with arguments that may be necessary for the function to accomplish the actions.
- The function in turn performs its task and returns control to the interpreter as well as any result which may be stored in other objects.

Two Types of Function in R

- Built-in: R has many in-built functions which can be directly called in the program without defining them first.
- User Defined: We can also create and use our own functions referred as user defined functions.
- Simple examples of in-built functions are `seq()`, `mean()`, `max()`, `sum()` and paste(...) etc. They are directly called by user written programs.

Creating Functions in R

```r
f <- function() {
  # empty function
}
# Function have their own class

class(f)
```

```
[1] "function"
```

```r
# Execute / Call this function
f()
```

```
NULL
```

Create a Hello World Function

```r
# Create a function with no arguments
say_hello <- function() {
    cat("Hello World\n")
}
# Call
say_hello()
```

```
Hello World
```

Create a function for printitng Hello World 3 times

```r
say_hello <- function(num) {
    for(i in seq_len(num)) {
        cat("Hello World!\n")
    }
}

# Call function with arguments
say_hello(3)
```

```
Hello World!
Hello World!
Hello World!
```

References

- https://www.datamentor.io/r-programming/
- https://online.stat.psu.edu/stat484/
- https://online.stat.psu.edu/stat485/
- https://www.statmethods.net/index.html
- https://simplystatistics.org/
- https://www.tutorialspoint.com/r/index.htm
- https://www.rforbiologists.org/
- https://compgenomr.github.io/book/
- https://statsandr.com/
- https://rafalab.github.io/pages/harvardx.html
- https://bolt.mph.ufl.edu/software/r-phc-6055/