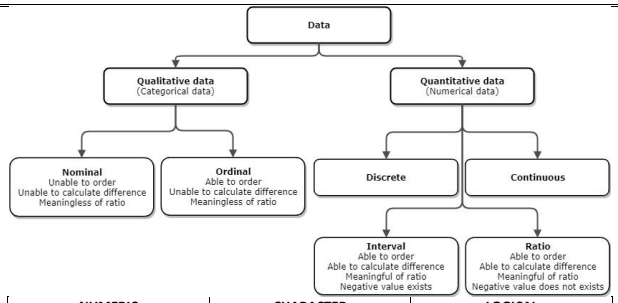


NUMERIC	CHARACTER	LOGICAL
# assignment of numeric	# assignment of character	# assignment of logical
> x = 1 > y <- 2	> x = "a" > "b" -> y	> x = TRUE > y = 1 > 2
> x > y	> x > y	> x > y
[1] 1 [1] 2	[1] "a" [1] "b"	[1] TRUE [1] FALSE
# classification of data	# classification of data	# classification of data
> class(x) > class(1L)	> class(x)	> class(x)
[1] "numeric" [1] "integer"	[1] "character"	[1] "logical"
# mode of data	# mode of data	# mode of data
> mode(x) > mode(1L)	> mode(x)	> mode(x)
[1] "numeric" [1] "numeric"	[1] "character"	[1] "logical"
# numeric vector	# character vector	# logical vector
> c(1, 2, 3)	> c("a", "b", "c")	> c(TRUE, FALSE, TRUE)
[1] 1 2 3	[1] "a" "b" "c"	[1] TRUE FALSE TRUE

General rules in R programming
- Identifiers of variables accepts only alphabets 'A' – 'Z', 'a' – 'z', digits '0' – '9', underscore '_'
  - The first character can only be alphabet
  - Cannot use R reserved words # Inf stands for infinity; NaN stands for not a number
  - Case sensitive

Variable assignment
- Global assignment is permanently assign a value to a variable, denoted by <- or ->
- Local assignment also assign value to a variable but it is temporary while within a function, denoted by =

Variable management
```
> ls()                # display a list of declared variables
> rm(list = ls())     # discard all the declared variables
```

Vector
```
                      # vector is a column vector by default
> vector[index]       # extracts a value by its index; index can be a vector or a logical expression
> length(vector)      # results number of value in the vector
> c(vector1, vector2, ...)   # combines values of vectors and convert into same data type
```


Auto-generate a sequence of a vector
```
> seq(from, to, by =, length =)   # 3rd argument is increment; 4th argument is max length
```

Auto-generate a repeat of a vector
```
> rep(data, times, each =)   # 2nd argument can be a vector which corresponds by location
                             # 3rd argument is number of repeats of each value
```

Matrix
```
> matrix(vector, nrow =, ncol =, byrow =)   # 2nd and 3rd argument is restriction; 4th argument is FALSE by default
                                            # if data does not fulfil the dimension, blank space fill by the same data
> dim(matrix)                               # returns the rows and columns which are the size of matrix
> nrow(matrix)                              # returns number of rows
> ncol(matrix)                              # returns number of columns
> matrix[indexRow, indexCol]                # index either a value or a vector; negative value excludes that index
> rbind(martix1, martix2, ...)              # merge matrix2 under matrix1; requires same number of columns
> cbind(martix1, martix2, ...)              # merge matrix2 next to matrix1; requires same number of rows
```

Array
```
> array(data = vector, dim = c(nrow, ncol, ndim))   # 2nd argument determines the size and number of dimensions
> array[,,dim][indexRow, indexCol]                  # extracts value(s) by referencing its dimension
```

List
```
> list(var1, var2, ...)      # var can be a value, a vector, or a matrix; number of arguments is number of dimensions
> list[[dim]][index]         # extracts data by referencing its dimension
> names(list) = c("name1", "name2", ...)   # renaming the reference of dimensions
> list$name[index]           # extracts data by referencing its dimension
> unlist(list)               # outputs a character vector combined with its arguments
```

Data frame
```
> data.frame(vector1, vector2, ...)   # groups vectors to be a data frame; requires same length of each vector
> summary(dataFrame)                  # outputs minimum, 1st quarter, median, 3rd quarter, maximum
> dataFram$vectorN[index]             # extracts values by its column of vectorN
> with(dataFrame, vectorN)            # gets direct access to vector without a sign of $
```

Factor
```
                                      # effectively storing character vector with duplicated values
> factor(vector)                      # stores a character vector with levels
> levels(factor)                      # return a non-duplicated values
> levels(factor) = c("name1", "namm2", ...)   # relabel the name of each value
```

Converting data type
```
> as.numeric(data)      # convert data into "numeric" data; character will convert into NA; logical will be 1 or 0
> as.integer(data)      # convert data into "integer" data
> as.character(data)    # convert data into "character" data; all data will convert in terms of " "
```

Testing the type of data
```
> is.numeric(data)      # test if data is numeric
> is.integer(data)      # test if data is integer
> is.character(data)    # test if data is character
> is.matrix(data)       # test if data is matrix
> is.na(data)           # test if data is NA
```


Operator precedence

$
[ ] [[ ]]
^
– (unary)
:
%% %/% %*%
* /
+ –
<> <= >= == !=
!
& | && ||
<- -> =

Built-in functions
```
> options(digits =)     # customise number of digits to display
> round(data, digits =) # rounding
> ceiling(data)         # always round up
> floor(data)           # always round down
> trunc(data)           # ignore decimal places
> sort(data, descending =)   # sort data in ascending order; 2nd argument is FALSE by default
> sum(data)             # sum of data
> cumsum(data)          # returns a vector of cumulative summation
> prod(data)            # product of data
> cumprod(data)         # returns a vector of cumulative production
> min(data)             # returns min.
> max(data)             # returns max.
> range(data)           # returns min. and max.
> mean(data)            # returns mean
> var(data)             # returns sample variance
> sd(data)              # returns sample standard deviation
```


Pseudorandom numbers
```
> set.seed(seed)                  # set the sequence of random numbers
> sample(data,size,replace=,prob=)   # generate size of random number form data
> rRName(n,arguments)             # generate n random number from the probability distribution of RName
```

Probability under probability distributions
```
> dRName(x,arguments)             # probability of x under the probability distribution of RName
```
- $P\{X = x\} = P(x)$ # probability mass function (p.m.f.)
- $P\{a < X < b\} = \int_a^b f(x)\,dx$ # probability density function (p.d.f.)

Cumulative probability under probability distributions
```
> pRName(q,arguments)             # cumulative probability of q under the probability distribution of RName
```
- $F(x) = \sum_{t=-\infty}^{x} p(t)$ # cumulative probability distribution function (c.d.f.) for discrete random variable
- $F(x) = \int_{-\infty}^{x} f(t)\,dt$ # cumulative probability distribution function (c.d.f.) for continuous random variable

Quantiles under probability distributions
```
> qRName(p,arguments)             # result x value for probability p under the probability distribution of RName
```

Distribution	R name	Additional arguments
Uniform	unif	min = 0, max = 1
Binomial	binom	size, prob
Exponential	exp	rate = 1
Poisson	pois	lambda
Normal	norm	mean = 0, sd = 1

Apply
```
> apply(data,dimension,function)   # apply function to dimension (1 for row; 2 for column) of data
```

Working directory
```
> getwd()                          # return current file path
> setwd("pathName")                # set the file path
> edit(file = "fileName.r")        # open on current path and launch a window's editor to edit fileName.r
> sources("fileName.r",echo=TRUE)  # read codes from .r file along with commands in the file
```

Data frame
```
> names(dataFrame)                 # return variable name stored in the dataFrame
> head(data,n)                     # display top n (default 6) row of data; tail(data,n) display from the end
> tapply(df$data1,df$data2,function)   # apply function to data1 (1 vector) grouped by data2
> by(df[,c(…)],df$data2,function)      # apply function to data1 (multiple vectors) by data2
> split(df$data1,df$data2)             # separate data1 by data2
> table(df$data)                       # return the frequency of each value in data
```

Converting logical data to numeric
```
> TRUE+0          # convert TRUE to 1; FALSE+0 will convert to 0
```

Scatter plot
```
> plot(data,…,main="title")        # plot a scatter plot with variable data,…
> abline(lsfit(data,…))            # fit a least squares line between data,…
```

File Input and output
```
> read.table("fileName.dat",header=TRUE)          # read .dat with header row
> write.table(df,"fileName.dat",row.names=FALSE)  # write .dat without row index
> read.csv("fileName.csv",header=TRUE)            # read .csv with header row
> write.csv(df,"fileName.csv",row.names=FALSE)    # write .csv without row index
```

Basic Data Visualisation

Multiple graphics

```
> par(mfrow=c(n,m))      # set up a n by m matrix for graphics; mfrow fill in row-wise; mfcol fill in column-wise
```

Histogram

```
> hist(data,freq=TRUE,main=)        # produces a histogram in frequency; FALSE for density instead
> x=seq(lb,ub,0.1)                  # set up a range of data
> lines(x,mean(data),sd(data),lty=2)    # plot a normal density of data along with histogram
```

Type of lty	0.blank	1.solid	2.dashed	3.dotted	4.dotdash'	5.longdash	6.twodash
		———	– – –	.....	–·–·–·	— — —	—– —–

Pei chart

```
> pie(data,labels=,cex=,main=)      # produces a pie chart of data; cex control density of labels
```

Bar chart

```
> barplot(data,horiz=,main=)        # produces a bar chart of data
> barplot(data,beside=)             # produces a grouped bar chart of data, which has at least 2 column
> t(data)                           # to transpose a matrix; may use to have different variable in plotting
> prop.table(data,margin=1)         # return row proportion; 2 for column proportion
```

Normal probability plot (Normal-QQ plot)

```
                                    # check normality assumption if a close to 45 degree straight line
> qqnorm(data,main=)                # produces a normal QQ-plot of data
> qqline(data,col=)                 # plot a reference line of the normal QQ-plot
```

Box plot

```
> boxplot(data,horiz=,main=)        # produces a box plot of data
> boxplot(data1~data2)              # produces a box plot of data1 grouped by data2
```

Scatter plot

```
                                    # check 2 variables if they are linearly related
> plot(data1,data2,pch=,bg=,col=,main=)     # produces a scatter plot of data1 verse data2
# pch for type of dot; bg for fill with colour; col for scale with colour
> text(data1-0.1,data2+0.1,cex=)    # add case # to the top left corner of each dot
> abline(intercept,slope)           # plot a reference line with intercept and slope
```

Matrix scatter plot

```
> pairs(data,pch=,bg=,main=)        # produces a matrix scatter plot of data, which are column variables
```

Time series plot

```
> plot(as.ts(data),main=)           # produces a time series of data
> plot(cbind(data1,data2,…))        # plot multiple time series on one plot
> matplot(cbind(data1,…),type="l")  # plot multiple time series on one plot with the same y -axis
> grid()                            # add a nx by ny rectangular grid lines to an existing plot
> lag()                             # return next value; only applicable in time series object
```

Mathematical function plot

```
> curve(formula,xLB,xUB,main=)      # produces a curve of formula, which is defined in all real number
# if a function is undefined in some points, plot by separate the defined parts and undefined parts
```


Programming in R

Function

```
> functionName<-function(argument1,argument2=,…) {  # argument from global variable(s)
                                                    # assign default value by in argument2 when it is declared
>     statements                                    # perform declaration of local variable(s) or expression
>     return(value)                                 # no return will return the last expression by default
> }                                                 # local variable is only accessible within the function
> variable(argument1,…)                             # call the customised function
```

Logical expression

```
> expression1&expression2           # vectorised operator and process all elements of involving vector
> expression1&&expression2          # only execute the first element and return 1 logical value
> any(vector)                       # test if vector has at least one TRUE value
```

If-else statement (standard version)

```
> if(condition)                     # if the condition is evaluated to be true, it processes the statement1
>     statement1
> else if                           # an alternative of if-clause when above condition evaluated to be false
>     statement2
> else                              # unlimited use of else-clause
>     statement3
```

If-else statement

```
> ifelse(condition,TrueOutcome,FalseOutcome)        # NA will not be evaluated
```

Switch statement

```
> switch(expression,case1,case2,…)          # each
```

Two looping control statement

```
> break             # immediately terminate the innermost enclosing loop statement
> next              # ignore the following statements of the loop and return to the very beginning of the loop
```

For loop

```
> for(name in vector){              # loop name from vector, a range of repetitions
>     statements                    # statements to be executed for a known of repetitions
> }                                 # name is only accessible within the for loop
```

While loop

```
> while(condition){     # always declare the variable involved in condition first; condition to continue
>     statements        # include modification of condition affecting factor or variable
> }                     # for unknown of repetitions
```

Infinitely loop

```
> repeat{               # loop infinitely until it reach break
>     statements        # include a if-clause to break out the infinitely loop ; a condition to terminate
> }
```


Useful R syntax

(A) List of operators

Priority	Operator	Meaning
1	$	component selection
2	[] [[]]	subscripts, elements
3	^	exponentiation
4	-	unary minus
5	:	sequence operator
6	%% %/% %*%	modulus, integer division, matrix multiplication
7	* /	multiplication, division
8	+ -	addition, subtraction
9	< > <= >= == !=	comparison
10	!	logical negation
11	& | && ||	vectorized and or, control and or
12	<- ->	assignments

(B) Control flow

Syntax	Description
if (cond) expr	evaluate cond; if TRUE, evaluate expr
if (cond) expr1 else expr2	evaluate cond; if TRUE, eval. expr1, if FALSE, expr2
ifelse(cond, expr1, expr2)	a vectorized version of if-else
switch(expr, …)	evaluate expr and compare it to arguments
break	terminate current loop and jump out
next	terminate current iteration and immediately start next iteration of the loop
return(expr)	terminate current function and immediately return the value of expr
stop(message)	terminate evaluation of the current function and display message
while (cond) expr	evaluate cond; if TRUE evaluate expr, until cond is FALSE
repeat expr	repeat expr indefinitely, some breaks should be included inside expr
for (name in expr1) expr2	evaluate expr2 once for each name in expr1

(C) Statistical Distributions

Distribution	R name	Additional arguments
Binomial	binom	size, prob
Normal	norm	mean, sd
Uniform	unif	min, max

Adding the letter d, p, q or r in front of the R's name represents the density, cdf, quantile and pseudo-random number respectively.

(D) Built-in Functions

Name	Operations
ceiling(x)	smallest integer greater than or equal to x
floor(x)	largest integer less than or equal to x
trunc(x)	ignore the decimal part
round(x)	round up for positive x and round down for negative x
sqrt(x), abs(x)	return square root and absolute value of x
sample(v,n,r,replace=F, prob=p)	generate n discrete random numbers which take value v with prob p
sort(x,ascending=T)	sort the vector in ascending or descending order
order(x), rank(x)	return the order and rank of numbers in x
sum(x), prod(x)	return the sum and product of all the elements in a vector
cumsum(x), cumprod(x)	cumulative sum and product of elements of a vector
min(x), max(x)	return the smallest, largest values and sample mean of x
mean(x)	
var(x), sd(x)	return the sample variance and standard deviation of a vector
seq(a,b,by=c)	generate a sequence of number from a to b with step c
rep(x,n)	replicate elements in a vector n times
apply(d,n,f)	apply a function f to a matrix d, n=1 row, n=2 column
length(x)	return the length of a vector
list(x,y,…)	construct list of items x, y,… etc
sample(x, size, replace, prob)	takes a sample of the specified size from the elements of x using either with or without replacement
plot(x,y,pch=,bg=)	plot (x,y), pch is print character, bg is background color
lines(x,y)	add lines or curves with coordinates (x,y) in the plot
abline(h= , v=)	add a horizontal or vertical line in the plot
matrix(data, nrow, ncol, byrow)	create a matrix from the given set of values in data
cbind(a,b), rbind(a,b)	binding two matrices column-wise and row-wise
dim(A), nrow(A), ncol(A)	return the row and column dimension of a matrix A

Input, Output and Other Programming Issues

Error handling

```
> stop("…")                         # terminate a function with error message "…"
```

Customise operator

```
> "operator"<-function(argument,…) {        # define operator using function()
>     statements                            # calculation statement that will perform
> }
> x operator y                              # use the operator with out ""
```

Flexible number of arguments

```
> functionName<-function(…) {       # "…" does not specific the number of arguments which is variable
>     arg=list(…)                   # select all arguments from the arguments
> }
```

Editing function

```
> fix(functionName)                 # it will pop up a window in editing functionName
```

Return a vector from function

```
> functionName<-function(…) {
>     statements
>     return(c(…))                  # return a vector; it is possible to rename at the same time
> }
```

Operation to vectors

```
> outer(vectorY,vectorX,"operator")     # a matrix for vectorY and vectorX with operation
                                        # vectorY is colum-wise; vectorX is row-wise
```

Floating point control

```
> sprintf("%n.mf",data)     # n refers to number of reversation; m refers to number of decimal places;
                            # f refers to floating point; can replace with e to display in scientific notation
```

Output function

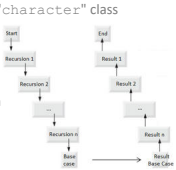
```
> cat(…,…,…)                # output "…" in "NULL" class; print() return "character" class output
```

Input function

```
> readline()        # pause the programme and wait for user's input; inputted data are in "character" class
```

Recursive function

```
> functionName<-function(…) {
>     statements                    # contain a base case to terminate the function
>     return(functionName(…))       # recursion of function by calling itself in return
> }
```



Measure running time

```
> startTime<-proc.time()            # store the time at the start of programme
> proc.time()-startTime             # calculate the duration of time
```

Check the existence of functions

```
> exists("functionName")            # return a logical value about the existence of functionName
```

Debugging function

```
> traceback()                       # display how each code statement is executed
```