

01 R Language Basics

June 22, 2020

Contents

1	Introduction	3
1.1	Hello World	3
2	Variables and Basic Data Types	3
2.1	Variables	3
2.1.1	Variable Naming	3
2.1.2	Value Assignment	3
2.1.3	Variable Removal	4
2.2	Basic Data Types	4
3	Objects	5
3.1	Vector	5
3.1.1	Create a vector	5
3.1.2	Getting the Length of a Vector	5
3.1.3	Create a sequence	5
3.1.4	Accessing vector elements	6
3.1.5	Vector Manipulation	6
3.1.6	Sorting a Vector	7
3.2	List	7
3.2.1	Create a list	7
3.2.2	Giving Names to Elements	8
3.2.3	Accessing Elements	8
3.2.4	Adding and Removing Elements	8
3.2.5	Combine Two Lists	9
3.2.6	Convert a List to a Vector	10
3.3	Matrix	10
3.3.1	Create a Matrix	10
3.3.2	Accessing Elements	11
3.3.3	Matrix Addition & Subtraction	11
3.4	Array	12
3.4.1	Create an Array	12
3.4.2	Name Dimensions	13
3.4.3	Accessing Elements	13
3.4.4	Array Addition	14
3.4.5	Calculations Across Array Elements	15
3.5	Factor	16

3.5.1	Create a Vector	16
3.5.2	Generating a Factor	16
3.6	Data Frames	17
3.6.1	Create a Data Frame	17
3.6.2	Structure of the Dataframe	17
3.6.3	Summer of the Dataframe	17
3.6.4	Making a Sub-dataframe	18
3.6.5	Update the Dataframe	18
3.6.6	Add New Entries by <code>cbind</code>	19
3.6.7	Merge Dataframes	20
3.6.8	Melting the Dataframe	20
3.6.9	Casting the Dataframe	21
3.7	String	22
3.7.1	Define a String	22
3.7.2	Combine Strings	22
3.7.3	Format String	22
3.7.4	Number of Chars	23
3.7.5	Change the Case	23
3.7.6	Extracting Parts of a String - <code>substr</code>	24
3.7.7	Split a String	24
4	Operations	24
4.1	Arithmetic Operators	24
4.2	Relational Operators	25
4.3	Logical Operators	25
4.4	Miscellaneous Operators	26
4.4.1	<code>:</code> Create a Vector Containing a Sequence of Numbers	26
4.4.2	<code>%in%</code> Check if a Variable Belongs to a Vector	27
4.4.3	<code>%%</code> Matrix Multiplication	27
5	Conditional Statements	28
5.1	<code>if ... else</code> statement	28
5.2	<code>if ... else if ... else</code> statement	28
5.3	<code>switch</code> statement	29
6	Loops	29
6.1	<code>repeat</code> Loop	29
6.2	<code>while</code> Loop	30
6.3	<code>for</code> Loop	30
6.4	<code>break</code> and <code>next</code>	31
7	Functions	31
7.1	Define a Function	31
7.2	<code>return</code> Statement	32

1 Introduction

This is a syntax and usage summary for R. The content is mainly from <https://www.tutorialspoint.com/r/index.htm>. This notes is for personal use only. Here I assume readers are experienced in programming.

1.1 Hello World

```
[1]: my_str <- "Hello World!"  
     print(my_str)
```

```
[1] "Hello World!"
```

```
[2]: cat(my_str, 123)
```

```
Hello World! 123
```

2 Variables and Basic Data Types

2.1 Variables

2.1.1 Variable Naming

- Letters, numbers, dot and underscore are allowed.
 - e.g. abc, .abc, abc., abc_
- Starting with a number are **not** allowed.
 - e.g. 1abc
- Starting with a dot is allowed
 - e.g. .abc
- Starting with a dot followed by a number is **not** allowed
 - e.g. .2abc
- Starting with an underscore is not allowed.
 - e.g. _tmp

2.1.2 Value Assignment

```
[3]: a <- 3  
  
     print(a)
```

```
[1] 3
```

```
[4]: a <<- 5
```

```
print(a)
```

```
[1] 5
```

```
[5]: a = 7
```

```
print(a)
```

```
[1] 7
```

```
[6]: 10 -> a
```

```
print(a)
```

```
[1] 10
```

2.1.3 Variable Removal

```
[7]: # Assign 10 to a
```

```
a <- 10
```

```
# Print a
```

```
print(a)
```

```
rm(a) # a is removed
```

```
[1] 10
```

2.2 Basic Data Types

```
[8]: v <- TRUE
```

```
print(class(v))
```

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

```
[9]: v <- 23.5
```

```
print(class(v))
```

```
[1] "numeric"
```

```
[10]: v <- 2L
```

```
print(class(v))
```

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

```
[11]: v <- 2+5i
```

```
print(class(v))
```

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

```
[12]: v <- "TRUE"  
      print(class(v))
```

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

```
[13]: v <- charToRaw("Hello")  
      print(class(v))
```

```
[1] "raw"
```

3 Objects

3.1 Vector

- An ordered collection of variables of **same** types.

3.1.1 Create a vector

```
[14]: # Create a vector.  
      v <- c(0,1,2,3)  
      print(v)  
  
      # Get the class of the vector.  
      print(class(v))
```

```
[1] 0 1 2 3
```

```
[1] "numeric"
```

3.1.2 Getting the Length of a Vector

```
[15]: # Print the length of a vector  
      print(length(v))
```

```
[1] 4
```

3.1.3 Create a sequence

```
[16]: # Create a vector of a sequence  
      v <- 1:10  
      print(v)  
  
      v <- 1.2:10.1
```

```
print(v)
```

```
[1] 1 2 3 4 5 6 7 8 9 10  
[1] 1.2 2.2 3.2 4.2 5.2 6.2 7.2 8.2 9.2
```

3.1.4 Accessing vector elements

- One should note that **indices in R starting from 1**.

```
[17]: # Accessing vector elements using position.  
t <- c("Sun", "Mon", "Tue", "Wed", "Thurs", "Fri", "Sat")  
u <- t[c(2,3,6)]  
print(u)  
  
# Accessing vector elements using negative indexing.  
# Negative index means dropping the index, not the same as that in Python!  
x <- t[c(-2,-5)]  
print(x)  
  
# Accessing vector element by an index  
print(t[2])
```

```
[1] "Mon" "Tue" "Fri"  
[1] "Sun" "Tue" "Wed" "Fri" "Sat"  
[1] "Mon"
```

3.1.5 Vector Manipulation

```
[18]: # Vector arithmetic  
u <- 1:6  
v <- 3:8  
  
print(u+v)
```

```
[1] 4 6 8 10 12 14
```

```
[19]: # Vector Element Recycling  
  
v1 <- c(3,8,4,5,0,11)  
v2 <- c(4,11)  
# V2 becomes c(4,11,4,11,4,11)  
  
add.result <- v1+v2  
print(add.result)  
  
sub.result <- v1-v2
```

```
print(sub.result)
```

```
[1]  7 19  8 16  4 22  
[1] -1 -3  0 -6 -4  0
```

3.1.6 Sorting a Vector

```
[20]: # Create a size-10 vector by sampling from 1:100 with replacement  
v <- sample(1:100, 10, replace=TRUE)  
print(v)  
  
v_inc <- sort(v, decreasing = FALSE)  
print(v_inc)  
  
v_dec <- sort(v, decreasing = TRUE)  
print(v_dec)
```

```
[1] 92 68  8  8 95 23 92 97 98 18  
[1]  8  8 18 23 68 92 92 95 97 98  
[1] 98 97 95 92 92 68 23 18  8  8
```

3.2 List

- An ordered collection of variables of **different** types.

3.2.1 Create a list

```
[21]: # Create a list.  
list1 <- list(c(1,2,3),18L,"ABC")  
  
# Print the list.  
print(list1)  
  
# Print the length of a list  
print(length(list1))
```

```
[[1]]  
[1] 1 2 3  
  
[[2]]  
[1] 18  
  
[[3]]  
[1] "ABC"
```

```
[1] 3
```

3.2.2 Giving Names to Elements

```
[22]: # Providing names to elements
names(list1) <- c("A vector", "The integer", "A_string")

print(list1)
```

```
$`A vector`
[1] 1 2 3
```

```
$`The integer`
[1] 18
```

```
$A_string
[1] "ABC"
```

3.2.3 Accessing Elements

```
[23]: # By indices
print(list1[2])

# By names
print(list1$`A vector`)
```

```
$`The integer`
[1] 18
```

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

3.2.4 Adding and Removing Elements

```
[24]: # Add an element
list1[length(list1)+1] <- "HI."

print(list1)
```

```
$`A vector`
[1] 1 2 3
```

```
$`The integer`
[1] 18
```



```
$A_string  
[1] "ABC"
```

```
[[4]]  
[1] "HI."
```

```
[25]: # Remove an element  
list1[3] <- NULL  
  
print(list1)
```

```
$`A vector`  
[1] 1 2 3
```

```
$`The integer`  
[1] 18
```

```
[[3]]  
[1] "HI."
```

3.2.5 Combine Two Lists

- Function `c` here is **combine**. Check `?c` for more detail.

```
[26]: # Define a new list  
list2 <- list("Sun", "Moon", 3.14, 20L)  
  
# Combine two lists using function c  
combined_list <- c(list1, list2)  
  
print(combined_list)
```

```
$`A vector`  
[1] 1 2 3
```

```
$`The integer`  
[1] 18
```

```
[[3]]  
[1] "HI."
```

```
[[4]]  
[1] "Sun"
```

```
[[5]]
```

```
[1] "Moon"
```

```
[[6]]
```

```
[1] 3.14
```

```
[[7]]
```

```
[1] 20
```

3.2.6 Convert a List to a Vector

```
[27]: # Define a List
list3 <- list(1,2,3,4,5)

v <- unlist(list3)

print(v)
```

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

3.3 Matrix

3.3.1 Create a Matrix

```
[28]: # Create a matrix.
M1 = matrix( c(1,2,3,4,5,6), nrow = 2, ncol = 3, byrow = TRUE)
print(M1)
```

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

```
[29]: # Create a matrix with dimnames
dimlabel1 = list("A", "B")
dimlabel2 = list("C", "D", "E")
M2 = matrix( c(1,2,3,4,5,6), nrow = 2, ncol = 3, byrow = TRUE, dimnames =
  ↪list(dimlabel1, dimlabel2))
print(M2)
```

```
      C D E
A 1 2 3
B 4 5 6
```

3.3.2 Accessing Elements

```
[30]: # Getting particular element  
print(M1[2,3])
```

```
[1] 6
```

```
[31]: # Getting particular column  
print(M1[,3])
```

```
[1] 3 6
```

```
[32]: # Getting particular row  
print(M1[1,])
```

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

3.3.3 Matrix Addition & Subtraction

- The rule for element-wise multiplication $*$ and division $/$ is the same.

```
[33]: # Re-define the M1 and M2 matrices  
M1 = matrix(1:6, nrow = 2, ncol = 3, byrow = TRUE)  
M2 = matrix(1:6, nrow = 2, ncol = 3, byrow = TRUE)  
  
M2 = M2 + 1  
print(M2)
```

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

```
[34]: M3 = M2 - M1  
print(M3)
```

```
      [,1] [,2] [,3]  
[1,]    1    1    1  
[2,]    1    1    1
```

```
[35]: M4 = M2 - 4*M1  
print(M4)
```

```
      [,1] [,2] [,3]  
[1,]   -2   -5   -8  
[2,]  -11  -14  -17
```

3.4 Array

3.4.1 Create an Array

```
[36]: # Create an array.  
a <- array(c(1,2,3,4),dim = c(3,3,2))  
print(a)
```

, , 1

	[,1]	[,2]	[,3]
[1,]	1	4	3
[2,]	2	1	4
[3,]	3	2	1

, , 2

	[,1]	[,2]	[,3]
[1,]	2	1	4
[2,]	3	2	1
[3,]	4	3	2

```
[37]: # Create an array from two vectors.  
a <- array(c(c(1,2,3,4),c(5,6)),dim = c(3,3,2))  
print(a)
```

, , 1

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

, , 2

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

3.4.2 Name Dimensions

```
[38]: # Define dimnames
dimname1 <- c("A", "B", "C")
dimname2 <- c("D", "E", "F")
dimname3 <- c("G", "H")

# Create an array from two vectors with dimnames.
a <- array(c(c(1,2,3,4),c(5,6)),dim = c(3,3,2), dimnames = list(dimname1,
  ↪dimname2, dimname3))
print(a)
```

, , G

 D E F
A 1 4 1
B 2 5 2
C 3 6 3

, , H

 D E F
A 4 1 4
B 5 2 5
C 6 3 6

3.4.3 Accessing Elements

```
[39]: # Getting a particular element
print(a[1,2,1])
```

[1] 4

```
[40]: # Getting a particular column
print(a[,2,1])
```

A B C
4 5 6

```
[41]: # Getting a particular matrix
print(a[, ,2])
```

 D E F
A 4 1 4
B 5 2 5
C 6 3 6

3.4.4 Array Addition

- Same as other element-wise operations, e.g. $-$, $*$, $/$

```
[42]: # Define two arrays
a1 <- array(1:3, dim = c(2,2,2))
a2 <- array(4:6, dim = c(2,2,2))

cat("Array a1:\n")
print(a1)

cat("Array a2:\n")
print(a2)

# Calculate the element-wise sums
a_result <- a1 + a2

cat("Array a_result:\n")
print(a_result)
```

Array a1:

, , 1

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

, , 2

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

Array a2:

, , 1

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

, , 2

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

Array a_result:

, , 1

```

      [,1] [,2]
[1,]    5    9
[2,]    7    5

```

```
, , 2
```

```

      [,1] [,2]
[1,]    7    5
[2,]    9    7

```

3.4.5 Calculations Across Array Elements

- We use function `apply()` to calculate across array elements

```
[43]: # Define an array
a1 <- array(1:3, dim = c(2,2,2))

cat("array a1:\n")
print(a1)

# Calculate means of elements along 1st axis
result1 <- apply(a1, MARGIN = c(1), mean)
cat("\narray result1:\n")
print(result1)

# Calculate sums of elements along 1st and 2nd axes
result2 <- apply(a1, MARGIN = c(1,2), sum)
cat("\narray result2:\n")
print(result2)
```

```
array a1:
, , 1
```

```

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

```

```
, , 2
```

```

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

```

```
array result1:
```

```
[1] 1.75 2.00
```

```
array result2:  
      [,1] [,2]  
[1,]    3    4  
[2,]    5    3
```

3.5 Factor

3.5.1 Create a Vector

```
[44]: # Create a vector.  
staff_genders <- array(c("M","M","F","M","F","F","M"))  
  
# Create a factor.  
staff_gender_factor <- factor(staff_genders)  
  
# Print the factor  
print(staff_gender_factor)  
print(nlevels(staff_gender_factor))
```

```
[1] M M F M F F M  
Levels: F M  
[1] 2
```

3.5.2 Generating a Factor

- Function `gl` is used to generate labels
 - Syntax: `gl(n, k, labels)`
 - `n` : number of levels.
 - `k` : the number of replications.
 - `labels` : a vector of labels.

```
[45]: # Create labels  
v <- gl(3, 4, labels = c("Apple", "Orange", "Banana"))  
print(v)
```

```
[1] Apple Apple Apple Apple Orange Orange Orange Orange Banana Banana  
[11] Banana Banana  
Levels: Apple Orange Banana
```


3.6 Data Frames

3.6.1 Create a Data Frame

```
[46]: # Create a data frame.
staff_table <- data.frame(
  ID=c(1L,2L,3L),
  Name=c("Tom", "Ann", "Peter"),
  Gender=c("M","F","M"),
  Age=c(32L,36L,29L)
)

# Print the data frame.
print(staff_table)
```

	ID	Name	Gender	Age
1	1	Tom	M	32
2	2	Ann	F	36
3	3	Peter	M	29

3.6.2 Structure of the Dataframe

```
[47]: # Get the structure of the data frame.
str(staff_table)

'data.frame':  3 obs. of  4 variables:
 $ ID      : int  1 2 3
 $ Name    : Factor w/ 3 levels "Ann","Peter",...: 3 1 2
 $ Gender  : Factor w/ 2 levels "F","M": 2 1 2
 $ Age     : int  32 36 29
```

3.6.3 Summer of the Dataframe

```
[48]: # Get the summer of the data frame
print(summary(staff_table))
```

	ID	Name	Gender	Age
Min.	:1.0	Ann :1	F:1	Min. :29.00
1st Qu.	:1.5	Peter:1	M:2	1st Qu.:30.50
Median	:2.0	Tom :1		Median :32.00
Mean	:2.0			Mean :32.33
3rd Qu.	:2.5			3rd Qu.:34.00
Max.	:3.0			Max. :36.00

3.6.4 Making a Sub-dataframe

```
[49]: # Making a sub-dataframe
staff_table_gender = data.frame(staff_table$Name, staff_table$Gender)

print(staff_table_gender)
```

```
  staff_table.Name staff_table.Gender
1             Tom                  M
2             Ann                  F
3           Peter                  M
```

```
[50]: # Getting first two rows
staff_table_12 <- staff_table[1:2,]

print(staff_table_12)
```

```
  ID Name Gender Age
1  1  Tom      M  32
2  2  Ann      F  36
```

```
[51]: # Getting particular cols and rows
staff_table_parti <- staff_table[c(1,3),c(2,4)]

print(staff_table_parti)
```

```
  Name Age
1  Tom  32
3 Peter  29
```

3.6.5 Update the Dataframe

```
[52]: # Add a new column
staff_table$Salary = c(30000,32000,29000)

print(staff_table)
```

```
  ID  Name Gender Age Salary
1  1   Tom      M  32  30000
2  2   Ann      F  36  32000
3  3 Peter      M  29  29000
```

```
[53]: # Add a new row
new_staff = data.frame(
  ID = 4L,
  Name = "Ken",
  Gender = "M",
```

```

    Age = 30L,
    Salary = 31000.0
)

staff_table <- rbind(staff_table, new_staff)

print(staff_table)

```

	ID	Name	Gender	Age	Salary
1	1	Tom	M	32	30000
2	2	Ann	F	36	32000
3	3	Peter	M	29	29000
4	4	Ken	M	30	31000

3.6.6 Add New Entries by cbind

- One should note that, `cbind` returns a vector containing elements of a **same type**. All numeric entries are converted to be character.

```

[54]: # Create a new row using cbind
new_staff = cbind(
  ID = c(5L, 6L),
  Name = c("Amy", "Zoe"),
  Gender = c("F", "F"),
  Age = c(29L, 31L),
  Salary = c(30000, 29500)
)

# Append the new row using rbind
staff_table <- rbind(staff_table, new_staff)

print(staff_table)

```

	ID	Name	Gender	Age	Salary
1	1	Tom	M	32	30000
2	2	Ann	F	36	32000
3	3	Peter	M	29	29000
4	4	Ken	M	30	31000
5	5	Amy	F	29	30000
6	6	Zoe	F	31	29500

3.6.7 Merge Dataframes

```
[55]: # New dataframe to merge
staff_office = cbind(
  ID = c(2L, 3L, 4L, 5L, 6L, 1L),
  Office = c("HK", "HK", "HK", "TW", "TW", "TW")
)

# Merge two dataframes according to "ID"
staff_table_w_office = merge(x = staff_table, y = staff_office, by.x = c("ID"),
  ↪by.y = c("ID"))

print(staff_table_w_office)
```

	ID	Name	Gender	Age	Salary	Office
1	1	Tom	M	32	30000	TW
2	2	Ann	F	36	32000	HK
3	3	Peter	M	29	29000	HK
4	4	Ken	M	30	31000	HK
5	5	Amy	F	29	30000	TW
6	6	Zoe	F	31	29500	TW

3.6.8 Melting the Dataframe

- Here a library namely “reshape” is required.

```
[56]: # Get a dataframe without Names and IDs
staff_meta_data <- data.frame(
  Gender = staff_table_w_office$Gender,
  Office = staff_table_w_office$Office,
  Age = as.numeric(staff_table_w_office$Age),
  Salary = as.numeric(staff_table_w_office$Salary))

cat("A sub-dataframe\n\n")
print(staff_meta_data)

cat("\nThe melten dataframe\n\n")
# Melt the dataframe
library(reshape) # Here we need the reshape library
staff_meta_data_melt <- reshape::melt(staff_meta_data, id = c("Gender",
  ↪"Office"))
print(staff_meta_data_melt)
```

A sub-dataframe

	Gender	Office	Age	Salary
1	M	TW	32	30000

2	F	HK	36	32000
3	M	HK	29	29000
4	M	HK	30	31000
5	F	TW	29	30000
6	F	TW	31	29500

The melten dataframe

	Gender	Office	variable	value
1	M	TW	Age	32
2	F	HK	Age	36
3	M	HK	Age	29
4	M	HK	Age	30
5	F	TW	Age	29
6	F	TW	Age	31
7	M	TW	Salary	30000
8	F	HK	Salary	32000
9	M	HK	Salary	29000
10	M	HK	Salary	31000
11	F	TW	Salary	30000
12	F	TW	Salary	29500

3.6.9 Casting the Dataframe

```
[57]: # Casting here means calculating means of age and salary according to id
      ↪ "Gender" and "Office"
casted_staff_meta_data <- reshape::cast(staff_meta_data_melt,
      ↪ Gender+Office~variable, mean)

cat("Means of age and salary (average over gender and office location)\n\n")
print(casted_staff_meta_data)

# Casting here means calculating means of age and salary according to id
      ↪ "Office"
casted_staff_meta_data <- reshape::cast(staff_meta_data_melt, Office~variable,
      ↪ mean)

cat("\nMeans of age and salary (average over office location)\n\n")
print(casted_staff_meta_data)
```

Means of age and salary (average over gender and office location)

	Gender	Office	Age	Salary
1	F	HK	36.0	32000
2	F	TW	30.0	29750
3	M	HK	29.5	30000

```
4      M      TW 32.0  30000
```

Means of age and salary (average over office location)

	Office	Age	Salary
1	HK	31.66667	30666.67
2	TW	30.66667	29833.33

3.7 String

3.7.1 Define a String

```
[58]: # Assign a string
str1 <- "I got Tom's lunchbox."
print(str1)
```

```
[1] "I got Tom's lunchbox."
```

3.7.2 Combine Strings

```
[59]: str1 <- "I "
str2 <- "Like"
str3 <- "Hamburger."

# Combine strings with default settings
result <- paste(str1, str2, str3)
print(result)
```

```
[1] "I Like Hamburger."
```

```
[60]: # Combine strings and connect them using "#"
result <- paste(str1, str2, str3, sep="#")
print(result)
```

```
[1] "I #Like#Hamburger."
```

```
[61]: # Cross combining strings using "#" and "%%"
result <- paste(c(str1, str2, str3), c("A", "B"), sep="#", collapse = "%%")
print(result)
```

```
[1] "I #A%%Like#B%%Hamburger.#A"
```

3.7.3 Format String

The format function has a general syntax:

```
format(x, digits, nsmall, scientific, width, justify = c("left", "right",  
"centre", "none"))
```

```
[62]: str1 <- format(3.14159265358, digits = 4) # Keeps at most 4 digits  
print(str1)
```

```
[1] "3.142"
```

```
[63]: str2 <- format(3.1, nsmall = 2) # Make decimal places after the floating point  
↪ to be 2  
print(str2)
```

```
[1] "3.10"
```

```
[64]: str3 <- format(3.1, scientific = TRUE)  
print(str3)
```

```
[1] "3.1e+00"
```

```
[65]: str4 <- format("Hello", width = 10)  
print(str4)
```

```
[1] "Hello      "
```

```
[66]: str5 <- format("Hello", width = 10, justify = "right")  
print(str5)
```

```
[1] "      Hello"
```

3.7.4 Number of Chars

```
[67]: # Define the string  
str1 <- "Hello"  
  
print(nchar(str1))
```

```
[1] 5
```

3.7.5 Change the Case

```
[68]: # Define the string  
str1 <- "Hello"  
  
print(toupper(str1))
```

```
[1] "HELLO"
```

```
[69]: print(tolower(str1))
```

```
[1] "hello"
```

3.7.6 Extracting Parts of a String - substr

```
[70]: # Define a string
str1 <- "The air quality is ideal for most individuals."

# Get a substring
str2 <- substr(str1, start = 5, stop = 15)
print(str2)
```

```
[1] "air quality"
```

3.7.7 Split a String

```
[71]: # Define a string
str1 <- "The air quality is ideal for most individuals."

words <- strsplit(str1, split = " ")
print(words)
```

```
[[1]]
[1] "The"          "air"          "quality"      "is"          "ideal"
[6] "for"          "most"         "individuals."
```

4 Operations

4.1 Arithmetic Operators

Arithmetic operators include +, -, *, /, %% (remainder), %/% (quotient) and ^ (exponent).

```
[72]: # Define vectors
v1 <- c(2,7,9)
v2 <- c(1,2,3)

print(v1+v2)
print(v1-v2)
print(v1*v2)
print(v1/v2)
print(v1%%v2)
print(v1%/%v2)
```



```
print(v1 ^ v2)
```

```
[1] 3 9 12
[1] 1 5 6
[1] 2 14 27
[1] 2.0 3.5 3.0
[1] 0 1 0
[1] 2 3 3
[1] 2 49 729
```

4.2 Relational Operators

Relational Operators includes <, >, ==, <=, >= and !=.

```
[73]: # Define vectors
v1 <- c(2,3,9)
v2 <- c(1,3,12)

print(v1<v2)
print(v1>v2)
print(v1==v2)
print(v1<=v2)
print(v1>=v2)
print(v1!=v2)
```

```
[1] FALSE FALSE TRUE
[1] TRUE FALSE FALSE
[1] FALSE TRUE FALSE
[1] FALSE TRUE TRUE
[1] TRUE TRUE FALSE
[1] TRUE FALSE TRUE
```

4.3 Logical Operators

```
[74]: print(TRUE || TRUE)
print(FALSE || TRUE)
print(TRUE || FALSE)
print(FALSE || FALSE)
```

```
[1] TRUE
[1] TRUE
[1] TRUE
[1] FALSE
```

```
[75]: print(TRUE && TRUE)
print(FALSE && TRUE)
```

```
print(TRUE && FALSE)
print(FALSE && FALSE)
```

```
[1] TRUE
[1] FALSE
[1] FALSE
[1] FALSE
```

```
[76]: v1 <- c(FALSE, FALSE, FALSE, TRUE)
      v2 <- c(FALSE, TRUE, TRUE, TRUE)

      print(v1&v2)
      print(v1|v2)
      print(!v1)
```

```
[1] FALSE FALSE FALSE TRUE
[1] FALSE TRUE TRUE TRUE
[1] TRUE TRUE TRUE FALSE
```

The logical operator `&&` and `||` considers only the first element of the vectors

```
[77]: v1 <- c(FALSE, FALSE, FALSE, TRUE)
      v2 <- c(TRUE, TRUE, TRUE, FALSE)
      print(v1&&v2)
      print(v1||v2)

      v3 <- c(TRUE, TRUE, TRUE, TRUE)
      v4 <- c(FALSE, FALSE, FALSE, TRUE)
      print(v2&&v3)
      print(v1&&v4)
```

```
[1] FALSE
[1] TRUE
[1] TRUE
[1] FALSE
```

4.4 Miscellaneous Operators

4.4.1 : Create a Vector Containing a Sequence of Numbers

```
[78]: v1 <- 1:10
      print(v1)
```

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

4.4.2 %in% Check if a Variable Belongs to a Vector

```
[79]: print(3 %in% v1)
      print(11 %in% v1)
```

```
[1] TRUE
[1] FALSE
```

4.4.3 %*% Matrix Multiplication

```
[80]: M1 <- matrix(1:6, nrow = 3, ncol = 2)
      M2 <- matrix(3:8, nrow = 2, ncol = 3)

      cat("\n")
      print(M1)
      cat("\n")
      print(M2)

      M_result <- M1 %*% M2

      cat("\n")
      print(M_result)
```

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

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

```
      [,1] [,2] [,3]
[1,]    19    29    39
[2,]    26    40    54
[3,]    33    51    69
```

5 Conditional Statements

5.1 if ... else statement

```
[81]: x <- 2 # Assign 2 to x

# A if-only statement
if (x < 4){
  cat("x is less than 4.\n\n")
}

x <- 5 # Assign 5 to x

# A if-else statement
if (x < 4){
  cat("x is less than 4.\n\n")
}else{
  cat("x is not less than 4.\n")
}
```

x is less than 4.

x is not less than 4.

5.2 if ... else if ... else statement

```
[82]: x <- 5 # Assign 5 to x

# A if-else statement
if (x < 4){
  cat("x is less than 4.\n\n")
}else if (x < 8){
  cat("x is not less than 4, but less than 8.\n")
}else{
  cat("x is not less 8.\n")
}
```

x is not less than 4, but less than 8.

```
[83]: x <- 10 # Assign 10 to x

# A if-else statement
if (x < 4){
  cat("x is less than 4.\n\n")
}else if (x < 8){
  cat("x is not less than 4, but less than 8.\n")
}
```

```

}else{
  cat("x is not less than 8.\n")
}

```

x is not less than 8.

5.3 switch statement

```

[84]: # Running a switch statement
x1 <- switch(2, "One", "Two", "None Matched")
print(x1)

```

[1] "Two"

```

[85]: # Define a list of indices
idx <- list(1,2,3)
# Give names to list elements
names(idx) <- c("Apple", "Orange", "Banana")

# Determine the output by a switch
x2 <- switch(idx$"Apple", "Apple", "Orange", "Banana")

cat("I got an", x2, " .\n")

```

I got an Apple .

6 Loops

6.1 repeat Loop

```

[86]: # Set the counter to be 0
count <- 0L

# Start the repeat loop
repeat{
  count <- count + 1L # Increase the counter by 1
  cat("This is iteration #", count, " .\n", sep = "")
  if (count >= 5L)
  {
    break # Break the loop if counter >= 5
  }
}

```

This is iteration #1.

This is iteration #2.

```
This is iteration #3.  
This is iteration #4.  
This is iteration #5.
```

6.2 while Loop

```
[87]: # Set the counter to be 0  
count <- 0L  
  
# Start the while loop  
while (count < 5)  
{  
  count <- count + 1L # Increase the counter by 1  
  cat("This is iteration #", count, ".\n", sep = "")  
}
```

```
This is iteration #1.  
This is iteration #2.  
This is iteration #3.  
This is iteration #4.  
This is iteration #5.
```

6.3 for Loop

```
[88]: # Generate integer sequence using seq.int  
# seq.int(1,10,2) returns a vector starting from 1, stopping at/before 10, and  
#   ↪ increasing by 2  
all_x <- seq.int(1,10,2)  
print(all_x)  
  
cat("\n")  
  
# Start the for loop  
for (x in all_x){  
  print(x)  
}
```

```
[1] 1 3 5 7 9
```

```
[1] 1  
[1] 3  
[1] 5  
[1] 7  
[1] 9
```

6.4 break and next

```
[89]: # Start the for loop
for (x in all_x){
  if (x > median(all_x)){ # Break the loop if x is larger than the median
    break
  }
  print(x)
}
```

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

```
[90]: # Start the for loop
for (x in all_x){
  if (x == all_x[2]){ # Skip the iteration if x is equal to all_x[2]
    next
  }
  print(x)
}
```

```
[1] 1
[1] 5
[1] 7
[1] 9
```

7 Functions

7.1 Define a Function

```
[91]: # Define a function doing addition and print
add_and_print <- function(x1, x2){
  y <- x1 + x2
  print(y)
}

# Here we call the function in the loop
for (i in 1:4){
  add_and_print(i, 2*i)
}
```

```
[1] 3
[1] 6
[1] 9
[1] 12
```

7.2 return Statement

```
[92]: # Define a function doing addition and **return**  
add_and_print <- function(x1, x2){  
  y <- x1 + x2  
  return(y)  
}  
  
# Here we print the return of the called function in the loop  
for (i in 1:4){  
  print(add_and_print(i,2*i))  
}
```

```
[1] 3  
[1] 6  
[1] 9  
[1] 12
```