Introductory Programming in R

By Asef Nazari

asef.nazari@monash.edu

Faculty of IT

Monash university

2. Data Types

2.1 Main Data Classes

R has five basic or “atomic” classes of objects:

- numeric:
  - double (real numbers): values like 2.3, 3.14, -5.7634 , ...
  - integer: values like 0,1,2, -4, ...
- character: values like "GDDS", 'exe'
- logical: TRUE and FALSE (always capital letters)
- complex: we have nothing to do with it in this unit.

In [32]:

```
typeof(2) # numbers by default are double
typeof(2L) # to force to be integer
typeof(3.14)
typeof(TRUE)
typeof("TRUE")
```

'double'

'integer'

'double'

'logical'

'character'

2.2 Vectors

The most basic type of R objects is a vector. All the objects we used so far are vectors of length 1. Vectors are variables with one or more values of the same type, e.g., all are of numeric class. For example, a numeric vector might consist of the numbers (1.2, 2.3, 0.2, 1.1).

- Vectors are created by c() function (concatenatation function)
- Also, they ca be created by vector() function: v <- vector("numeric", length=5)
- should contain objects of the same class

- if you put objects from different classes, an implicit coercion (the calss of value would be changed) will happen
- Creating variables using seq and rep functions.

In [33]:

```
v1 <- c(5,7,9) # a vector called v1 is created.
```

In [34]:

```
v1
```

```
5 7 9
```

In [35]:

```
print(v1)
#this says v1 is a vector, or a sequence of objects, and the first one is 5.
```

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

In [36]:

```
v2 <- 3:35 # a sequence of consecutive integers are put in v2. The sequence starts
print(v2)
# the first item is 3 and the 26th item is 28.
```

```
 [1]  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
25 26 27
[26] 28 29 30 31 32 33 34 35
```

In [37]:

```
v3 <- c("Helo", "Hi", "Bye") # a vector of characters
print(v3)
```

```
[1] "Helo" "Hi"   "Bye"
```

In [38]:

```
v4 <- c(TRUE, TRUE, FALSE, TRUE, TRUE) # a vector of logical values
v4
```

```
TRUE TRUE FALSE TRUE TRUE
```

In [39]:

```
length(v4) # gives the length of a vector
```

```
5
```

In [40]:

```
v5 <- seq(2,8) #another way of making a vector of consecutive numbers. Same as 2:8
v5
```

```
2 3 4 5 6 7 8
```

In [41]:

```
v6 <- seq(from=3, to=10, by=2) # equally you can write seq(3,10,2)
print(v6)
```

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

In [42]:

```
#learn more about seq() function by typing ? seq
?seq
```

In [43]:

```
v7 <- vector(mode="numeric", length=5) # another way of creating a vector
print(v7)
```

```
[1] 0 0 0 0 0
```

In [44]:

```
v8 <- c(5, "a", 2) #different types, so a coercion happens. Be very careful about t
print(v8)
```

```
[1] "5" "a" "2"
```

In [45]:

```
#accessing elements of a vector
v8[1]
print(v8[2])
```

```
'5'
```

```
[1] "a"
```

In [46]:

```
vv <- c(1,2,3)
vv
vv[2] #prints the second item
vv[2] <- 257 # changes the value stored in the second element
vv
```

```
1 2 3
```

```
2
```

```
1 257 3
```

In [47]:

```
#to choose more than one element from a vector
x <- c(12.2, 52.3, 10.2, 11.1)
x[1] # only the first element
x[c(1,3)] # the first and third elemment
```

```
12.2
```

```
12.2 10.2
```

In [48]:

```
# Adding an element to the end of a list
v <- c(1,2,3)
print(v)
```

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

In [49]:

```
v <- c(v, 100) # 100 is added to the end of a vector
print(v)
```

```
[1]   1   2   3 100
```

In [50]:

```
# Create sequential data
x1 <- 0:10  # Assigns number 0 through 10 to x1
x2 <- 10:0  # Assigns number 10 through 0 to x2
x3 <- seq(10)  # Counts from 1 to 10
x4 <- seq(30, 0, by = -3)  # Counts down by 3
```

In [51]:

```
x <- c(1,3,6,9,0)
x
x[-2] # all the elements except the second element
x[3] <- 200 #modify an element
x
# to delete a vector
x <- NULL
x
```

```
1  3  6  9  0
```

```
1  6  9  0
```

```
1  3  200  9  0
```

```
NULL
```

In [52]:

```
x <- c(2, 9, 7)
x
y <- c(x, x, 10)
y
```

```
2  9  7
```

```
2  9  7  2  9  7  10
```

In [53]:

```
round(seq(1,3,length=10), 2)
```

```
1  1.22  1.44  1.67  1.89  2.11  2.33  2.56  2.78  3
```

In [54]:

```
seq(from = 2, by = -0.1, length.out = 4)
```

```
2  1.9  1.8  1.7
```

In [55]:

```
x <- rep(3,4)
x
```

```
3  3  3  3
```

In [56]:

```
rep(1:5,3)
```

```
1  2  3  4  5  1  2  3  4  5  1  2  3  4  5
```

In [57]:

```
x <- c(7,3,5,2,0,1)
y <- x[-3]
y
```

```
7  3  2  0  1
```

In [58]:

```
y <- x[-length(x)] # always delets the final element
y
```

```
7  3  5  2  0
```

2.3 Lists

Other basic object in R is a list. A list is very similar to a vector, but it could contain objects from different classes. You can create a list using list() function. The main functionality of lists in putting outputs of functions inside. Later we will see an important example of lm() functions.

In [59]:

```
L1 <- list(5, "a", 2)
print(L1)
# L1 has 3 elements, and each element is considered as a vector
#pay attention to double brackets. It shows the elements of the list
```

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

[[2]]
[1] "a"

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

In [60]:

```
L1 #auto printing
```

1. 5
2. 'a'
3. 2

In [61]:

```
length(L1)
```

3

In [62]:

```
L2 <- list(c(1,2,3), c("One", "Two"), TRUE)
print(L2)
```

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

[[2]]
[1] "One" "Two"

[[3]]
[1] TRUE
```

In [63]:

```
L2
```

1. 1 2 3
2. 'One' 'Two'
3. TRUE

In [64]:

```
L1[1]
```

1. 5

In [65]:

```
print(L2[1])
```

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

In [66]:

```
print(L2[[1]])
```

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

2.4 Numbers

Numbers in R are considerd as **numeric**, (as real numbers with double precision) . If you want an integer, you need to explicitly add L to the end of the number, otherwise it is a double.

Special numbers:

- Inf, infinity, for $\frac{1}{0}$
- NaN, not a number, for $\frac{0}{0}$
- NA can be thought as a missing value

In [67]:

```
x <- 1
print(x)
class(x)
typeof(x)
y <- 1L
print(y)
class(y)
typeof(y)
```

```
[1] 1
```

'numeric'

'double'

```
[1] 1
```

'integer'

'integer'

In [68]:

```
c1 <- "Heloo" # character variable
c2 <- "The World!" # another character variable
paste(c1, c2)
print(c(c1, c2))
```

'Heloo The World!'

```
[1] "Heloo"      "The World!"
```

In [69]:

```
sqrt(-2)  #NaN stands for “not a number”
```

```
Warning message:
In sqrt(-2): NaNs produced
```

NaN

2.5 Changing Class of a Value

You saw that a vector contains values of only one class. If different classes mixed together by having valuesw ith different classes in a vector, an implicit coercion happens. It means R will convert all the values to a class that are the same. However, sometimes we want to change the type of a value ourselves, so we implemenet an explicit coercion by as.SomeClass() functions.

- as.numeric() to change the type into numeric if it is possibel

- as.logical() to change into logical if it is possible
- as.character()
- as.complex()
- as.integer()

Sometimes R cannot convert one type to another, and gives NA. Also, you will get warning from R.

In [70]:

```
x <- 1:5 #sequence of numbers
class(x)
y <- as.numeric(x)
class(y)
z <- as.logical(x)
print(z)
class(z)
u <- as.character(z)
print(u)
class(u)
```

'integer'

'numeric'

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

'logical'

```
[1] "TRUE" "TRUE" "TRUE" "TRUE" "TRUE"
```

'character'

In [71]:

```
t <- as.numeric(u)
t
class(t)
```

```
Warning message:
In eval(expr, envir, enclos): NAs introduced by coercion
```

NA NA NA NA NA

'numeric'

In [72]:

```
#list does not have any problm with mixing data types. Very poerful!
x <- list(14, "Hello", TRUE, list(23, "Hi", TRUE, FALSE))
x
```

1. 14
2. 'Hello'
3. TRUE
4. A. 23
   B. 'Hi'
   C. TRUE
   D. FALSE

In [73]:

```
print(x)
#elements of list has double brackets around them. Other objects have single bracke
```

```
[[1]]
[1] 14

[[2]]
[1] "Hello"

[[3]]
[1] TRUE

[[4]]
[[4]][[1]]
[1] 23

[[4]][[2]]
[1] "Hi"

[[4]][[3]]
[1] TRUE

[[4]][[4]]
[1] FALSE
```

2.6 Factors

Categorical data in R are represented using factors. We will learn a lot about this type of data soon. Factors are stored as integers, but they are assigned labels. R sorts factors in alphabetical oredr. Factors can be ordered or unordered. R considers factors as nominal categorical variables, and "ordered" as ordinal categorical variables.

In [74]:

```
x <- factor(c("male", "fmale", "male", "male", "fmale", "male")) #create a factor o
print(x)
```

```
[1] male  fmale male  male  fmale male
Levels: fmale male
```

In [75]:

```
levels(x) #alphabetical order
```

'fmale' 'male'

In [76]:

```
nlevels(x)
```

2

In [77]:

```
unclass(x)
```

```
    2  1  2  2  1  2
```

In [78]:

```
table(x) #gives frequency count
```

```
x
fmale  male
    2     4
```

In [79]:

```
levels(x)
```

```
    'fmale'   'male'
```

In [80]:

```
summary(x)
```

```
fmale
2
male
4
```

In [81]:

```
#change the order of levels
#this is important in linear regression. The first level is used as the baseline le
x <- factor(c("male", "fmale", "male", "male", "fmale", "male"), levels=c("male", "
print(x)
```

```
[1] male  fmale male  male  fmale male
Levels: male fmale
```

In [82]:

```
d <- c(1,1,2,3,1,3,3,2)
d[1]+d[2] # integers
fd <- factor(d)
print(fd)
fd[1]+fd[2] #factors, you will get warning
unclass(fd) # bring down to integer vector
```

```
2
```

```
[1] 1 1 2 3 1 3 3 2
Levels: 1 2 3
```

```
Warning message:
In Ops.factor(fd[1], fd[2]): '+' not meaningful for factors
```

```
[1] NA
```

```
    1  1  2  3  1  3  3  2
```

In [83]:

```
rd <- factor(d, labels=c("A", "B", "C")) # factor is as an integer vector where eac
print(rd)
```

```
[1] A A B C A C C B
Levels: A B C
```

In [84]:

```
levels(rd) <- c("AA", "BB", "CC")
print(rd)
```

```
[1] AA AA BB CC AA CC CC BB
Levels: AA BB CC
```

In [85]:

```
is.factor(d)
is.factor(fd)
```

FALSE

TRUE

In [86]:

```
#ordered factor variable
x1 <- factor(c("low", "high", "medium", "high", "low", "medium", "high"))
print(x1)
x1f <- factor(x1, levels = c("low", "medium", "high"))
print(x1f)
```

```
[1] low    high   medium high   low    medium high
Levels: high low medium
[1] low    high   medium high   low    medium high
Levels: low medium high
```

In [87]:

```
x1o <- ordered(x1, levels = c("low", "medium", "high"))
print(x1o)
```

```
[1] low    high   medium high   low    medium high
Levels: low < medium < high
```

In [88]:

```
min(x1o) ## works!
```

low

In [89]:

```
is.factor(x1o)
```

TRUE

In [90]:

```
attributes(x1o)
```

$levels
'low' 'medium' 'high'
$class
'ordered' 'factor'

By using the gl() function, we can generate factor levels . It takes two integers as input which indicates how many levels and how many times each level.

- gl(n, m, labels)
- n is the number of levels
- m is the number of repeatitions
- labels is a vector of labels

In [91]:

```
v <- gl(3, 4, labels = c("H1", "H2","H3"))
print(v)
```

```
 [1] H1 H1 H1 H1 H2 H2 H2 H2 H3 H3 H3 H3
Levels: H1 H2 H3
```

In [92]:

```
class(v)
```

'factor'

2.7 Missing Values

A variable might not have a value, ot its value might missing. In R missing values are displayed by the symbol NA (not avaiable).

- NA, not available
- Makes certain calculations impossible
- is.na()
- is.nan()
- NA values have class

In [93]:

```
x1 <- c(4, 2.5, 3, NA, 1)
summary(x1)  # Works with NA
mean(x1)  # Doesn't work
mean(x1, na.rm=TRUE)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  1.000   2.125   2.750   2.625   3.250   4.000       1
```

```
[1] NA
```

2.625

In [94]:

```
is.na(x1)
```

FALSE FALSE FALSE TRUE FALSE

In [95]:

```
# To find missing values
which(is.na(x1))  # Give index number
```

4

In [96]:

```
# Ignore missing values with na.rm = T
mean(x1, na.rm = T)
```

2.625

In [97]:

```
# Replace missing values with 0 (or other number)
# In data wrangling you will learn a lot about this.
x2 <- x1
x2[is.na(x2)] <- 0
x2
```

4 2.5 3 0 1

2.8 Subsetting

- [] always returns an object of the same class
- [[]] is used to extract elements from a list fo dataframe. It always return a single element.
- $ to extract elements from a list or dataframe unsing a name

In [98]:

```
x <- c("a1", "a2", "a3", "a4", "a5", "a6")
```

In [99]:

```
x[1] #extracts the first item. it's a vector
x[2:5] # extracts a sequence. it's a vector
```

'a1'

'a2' 'a3' 'a4' 'a5'

In [100]:

```
x <- list(prime=c(2,3,5,7), even=c(0,2,4,6), odd=c(1,3,5,7), digit=3.14)
```

In [101]:

```
print(x)
```

```
$prime
[1] 2 3 5 7

$even
[1] 0 2 4 6

$odd
[1] 1 3 5 7

$digit
[1] 3.14
```

In [102]:

```
print(x[1]) #extracts the first element of the list, and it is a list
class(x[1])
```

```
$prime
[1] 2 3 5 7
```

'list'

In [103]:

```
print(x[[1]]) #extracts the first element and returns a vector.
```

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

In [104]:

```
print(x[4])
```

```
$digit
[1] 3.14
```

In [105]:

```
print(x[[4]])
```

```
[1] 3.14
```

In [106]:

```
x$digit
```

3.14