Section 2.4 and 2.5

Chammika De Mel

7/12/2021

Section 2.4

Question 7

```
a <- c("MON", "TUES", "WED", "THUR", "FRI")
typeof(a)
```

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

```
class(a)
```

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

Both Class and Type: character

```
b <- c(1, 2, 3, 4, 5)
typeof(b)
```

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

```
class(b)
```

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

Here Class is numeric but type is double. Class is a broader group that the data belongs to (or the inherited class) and typeof() shows the internal representation of data. Though integers are entered they are represented in "double" format.

```
c <- c(1L, 2L, 3L, 4L, 5L)
typeof(c)
```

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

```
class(c)
```

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

The "L" next to numbers tells R to take the numbers as integers. The class and type both are integer.

```
d <- c(TRUE, FALSE, TRUE, TRUE)
typeof(d)
```

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

```
class(d)
```

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

Both Class and Type: logical

```
e <- c(2+3i, 1+2i, 5+3i)
typeof(e)
```

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

```
class(e)
```

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

Both Class and Type: complex

```
f <- c("MON", TRUE, 1, 1L)
typeof(f)
```

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

```
class(f)
```

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

Here we have entered a mix of character, logical, double and integer data. Vectors can store only data of a single type. Therefore the data is converted to a single type by force (We call this situation: "Data are coerced").

Data are coerced to the most compatible type of entered data. Here all those can be coerced to characters. See below.

```
f
```

```
## [1] "MON"  "TRUE" "1"    "1"
```

Hence both Class and Type: character

Question 8

```
a1 <- vector("numeric", 8)
a2 <- vector("complex", 8)
a3 <- vector("logical", 8)
a4 <- vector("character", 8)

a1;a2;a3;a4
```

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

```
## [1] 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
## [1] "" "" "" "" "" "" "" ""
```

a1, a2, a3 and a4 are vectors of length 8 with default values of numeric, complex, logical and charactor respectively.

```
b1 <- numeric(8)
b2 <- complex(8)
b3 <- logical(8)
b4 <- character(8)

b1;b2;b3;b4
```

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

```
## [1] 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
## [1] "" "" "" "" "" "" "" ""
```

Above code does the same process as the previous code chunk for a1-a4.

Section 2.5

Question 9

First we will check what x is.

```
x <- 1:10
x
```

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

```
x[3]
```

```
## [1] 3
```

Returns the 3rd element of vector x.

```
x[c(2, 4)]
```

```
## [1] 2 4
```

Returns 2nd and 4th elements of vector x.

```
x[-1]
```

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

Returns all elements except the 1st element.

```
# x[c(2, -4)]
```

This causes an error. There is no method defined for these kind of inputs.

```
x[c(2.4, 3.54)]
```

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

The indexes are rounded down to the nearest integer. This is the function of "floor".

```
set.seed(1762021)
st_normal <- rnorm(100)
st_normal
```

```
##   [1] -0.97448987 -1.27461307  0.48801800  0.49141295 -0.67692483  1.34014254
##   [7]  2.18583063 -0.17553252 -1.63483137  0.03024727 -2.11800235  0.18654600
##  [13]  1.39701400  0.14609591 -1.83088596 -2.36168175 -2.02598608  0.67525542
##  [19]  1.15826732 -0.37223351 -0.08631292 -0.46437630 -0.78819679  0.04924904
##  [25] -0.78357858  1.05806879  0.13313826  0.76659792  0.12350731 -0.55793273
##  [31]  1.03673831  1.38740632 -1.21604544 -1.27585619 -1.27849850 -0.46260147
##  [37]  1.11404898 -0.60577065 -0.05419039  0.82942191 -0.11990169 -1.14907057
##  [43] -1.06150768 -0.26883482  1.43786263 -0.76171195 -0.14256348  0.79500907
##  [49] -0.09093005  1.68983869  0.48986626 -0.49533952 -0.76212444 -0.45888922
##  [55] -0.64235313  0.32436572 -0.86661285  0.12504993 -1.30756345 -1.20334635
##  [61] -0.59564966  1.62625440  0.59231311  1.42604105 -0.23446921 -0.82578278
##  [67] -0.79852124  0.88041040  0.65535406  0.20698931 -0.99832265 -2.12806683
##  [73]  1.43832441 -0.18321060 -0.34885211 -0.76699277  0.75013339  0.29268751
##  [79] -1.08067845  0.87199830 -1.11618678 -0.60850387 -0.30167388 -0.89210264
##  [85] -1.64258734 -0.67589617  0.45925549  0.54833515 -0.17458499 -1.01837339
##  [91]  1.26295194  0.07384001  0.75025707 -0.33222339 -0.14743703 -0.46438730
##  [97] -1.24863294  1.48870732  0.71412848 -0.71444642
```

```
B <- seq(10,length(st_normal),10)
st_normal[-B]
```

```
##  [1] -0.97448987 -1.27461307  0.48801800  0.49141295 -0.67692483  1.34014254
##  [7]  2.18583063 -0.17553252 -1.63483137 -2.11800235  0.18654600  1.39701400
## [13]  0.14609591 -1.83088596 -2.36168175 -2.02598608  0.67525542  1.15826732
## [19] -0.08631292 -0.46437630 -0.78819679  0.04924904 -0.78357858  1.05806879
## [25]  0.13313826  0.76659792  0.12350731  1.03673831  1.38740632 -1.21604544
## [31] -1.27585619 -1.27849850 -0.46260147  1.11404898 -0.60577065 -0.05419039
## [37] -0.11990169 -1.14907057 -1.06150768 -0.26883482  1.43786263 -0.76171195
## [43] -0.14256348  0.79500907 -0.09093005  0.48986626 -0.49533952 -0.76212444
## [49] -0.45888922 -0.64235313  0.32436572 -0.86661285  0.12504993 -1.30756345
## [55] -0.59564966  1.62625440  0.59231311  1.42604105 -0.23446921 -0.82578278
## [61] -0.79852124  0.88041040  0.65535406 -0.99832265 -2.12806683  1.43832441
## [67] -0.18321060 -0.34885211 -0.76699277  0.75013339  0.29268751 -1.08067845
## [73] -1.11618678 -0.60850387 -0.30167388 -0.89210264 -1.64258734 -0.67589617
## [79]  0.45925549  0.54833515 -0.17458499  1.26295194  0.07384001  0.75025707
## [85] -0.33222339 -0.14743703 -0.46438730 -1.24863294  1.48870732  0.71412848
```

Checking if it worked properly

```
A <- st_normal[-B]
which(!(st_normal%in%A))
```

```
##  [1]  10  20  30  40  50  60  70  80  90 100
```

set.seed() ensures reproducibility of of the results.