SLOVAK UNIVERSITY OF
TECHNOLOGY IN BRATISLAVA

Data Types in R

Dr.Sc. Oleksii Yehorchenkov

Department of Spatial Planning

Objects

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

- character
- numeric (real numbers)
- integer
- complex
- logical (True/False)

The most basic object is a **vector**

A vector can only contain objects of the same class BUT: The one exception is a list, which is represented as a vector but can contain objects of different classes (indeed, that’s usually why we use them)

Empty vectors can be created with the `vector()` function.

Numbers

- Numbers in R a generally treated as numeric objects (i.e. double precision real numbers)
- If you explicitly want an integer, you need to specify the `L` suffix
- Ex: Entering 1 gives you a numeric object; entering `1L` explicitly gives you an integer.
- There is also a special number `Inf` which 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 (more on that later).

Attributes

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.

Entering Input

```
1 x <- 1
2 print(x)
```

```
[1] 1
```

```
1 x
```

```
[1] 1
```

```
1 msg <- "hello"
2 msg
```

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

The grammar of the language determines whether an expression is complete or not.

```
1 x <-  ## Incomplete expression
```

The # character indicates a comment. Anything to the right of the # (including the # itself) is ignored.

Evaluation

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

```
x <- 5   ## nothing printed
x        ## auto-printing occurs
```

```
[1] 5
```

```
print(x)   ## explicit printing
```

```
[1] 5
```

The `[1]` indicates that `x` is a vector and `5` is the first element.

Printing

```
1  x <- 1:100
2  x
```

```
  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
 [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
 [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
 [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
 [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
 [91]  91  92  93  94  95  96  97  98  99 100
```

The `:` operator is used to create integer sequences.

Creating Vectors

The `c()` function can be used to create vectors of objects.

```
x <- c(0.5, 0.6)        ## numeric
x <- c(TRUE, FALSE)     ## logical
x <- c(T, F)            ## logical
x <- c("a", "b", "c")   ## character
x <- 9:29               ## integer
x <- c(1+0i, 2+4i)      ## complex
```

Using the `vector()` function

```
x <- vector("numeric", length = 10)
x
```

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

Mixing Objects

What about the following?

```
y1 <- c(1.7, "a")   ## character
y2 <- c(TRUE, 2)    ## numeric
y3 <- c("3", TRUE)    ## character
```

```
[1] "1.7" "a"
```

```
[1] 1 2
```

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

When different objects are mixed in a vector, coercion occurs so that every element in the vector is of the same class.

Explicit Coercion

Objects can be explicitly coerced from one class to another using the as.* functions, if available.

```
x <- 0:10
class(x)
```

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

```
as.numeric(x)
```

```
 [1]  0  1  2  3  4  5  6  7  8  9 10
```

```
as.logical(x)
```

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

```
as.character(x)
```

```
 [1] "0"  "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10"
```

Explicit Coercion

Nonsensical coercion results in **NAs** (Not Available).

```
1 x <- c("a", "b", "c")
2 as.numeric(x)
```

```
Warning: NAs introduced by coercion
```

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

```
1 as.logical(x)
```

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

```
1 as.complex(x)
```

```
Warning: NAs introduced by coercion
```

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

Matrices

Matrices are vectors with a dimension attribute. The dimension attribute is itself an integer vector of length 2 (**nrow, ncol**)

```
m <- matrix(nrow = 2, ncol = 3)
m
```

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

```
dim(m)
```

```
[1] 2 3
```

```
attributes(m)
```

```
$dim
[1] 2 3
```

Matrices (cont'd)

Matrices are constructed column-wise, so entries can be thought of starting in the "upper left" corner and running down the columns.

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

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

Matrices (cont'd)

Matrices can also be created directly from vectors by adding a **dimension attribute**.

```
1  m <- 1:10
2  m
```

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

```
1  dim(m) <- c(2,5)
2  m
```

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

cbind-ing and rbind-ing

Matrices can be created by column-binding or row-binding with `cbind()` and `rbind()`.

```
1  x <- 1:5
2  y <- 6:10
3
4  cbind(x, y)
```

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

```
1  rbind(x, y)
```

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

Lists

Lists are a special type of vector that can contain elements of different classes. **Lists** are a very important data type in R and you should get to know them well.

```
1 x <- list(1, "a", TRUE, 1 + 4i)
2 x
```

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

[[2]]
[1] "a"

[[3]]
[1] TRUE

[[4]]
[1] 1+4i
```

Factors

Factors are used to represent categorical data. **Factors** can be *unordered* or *ordered*. One can think of a factor as an integer vector where each integer has a label.

- Factors are treated specially by modelling functions like `lm()` and `glm()`
- Using factors with labels is better than using integers because factors are self-describing; having a variable that has values `"Male"` and `"Female"` is better than a variable that has values `1` and `2`.

Factors

```
1 x <- factor(c("yes", "yes", "no", "yes", "no"))
2 x
```

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

```
1 table(x)
```

```
x
 no yes
  2   3
```

```
1 unclass(x)
```

```
[1] 2 2 1 2 1
attr(,"levels")
[1] "no"  "yes"
```

```
1 attr(x, "levels")
```

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

Factors

The order of the levels can be set using the levels argument to `factor()`. This can be important in linear modelling because the first level is used as the baseline level.

```
1 x <- factor(c("yes", "yes", "no", "yes", "no"),
2              levels = c("yes", "no"))
3 x
```

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

Missing Value

Missing values are denoted by NA or NaN for undefined mathematical operations.

```
x = 0 / 0
x
```

```
[1] NaN
```

```
y = sqrt(-2)
y
```

```
[1] NaN
```

but

```
x = 1 / 0
x
```

```
[1] Inf
```

- `is.na()` is used to test objects if they are NA;
- `is.nan()` is used to test for NaN;
- NA values have a class also, so there are integer NA, character NA, etc.;
- A NaN value is also NA but the converse is not true

Missing Value

```r
x <- c(1, 2, NA, 10, 3)

is.na(x)
```

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

```r
is.nan(x)
```

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

```r
x <- c(1, 2, NaN, NA, 4)

is.na(x)
```

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

```r
is.nan(x)
```

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

Data Frames

Data frames are used to store tabular data

- They are represented as a special type of list where every element of the list has to have the same length;
- Each element of the list can be thought of as a column and the length of each element of the list is the number of rows;
- Unlike matrices, data frames can store different classes of objects in each column (just like lists); matrices must have every element be the same class;
- Data frames also have a special attribute called `row.names`;
- Data frames are usually created by calling functions like `read.table()` or `read.csv()`; Can be converted to a matrix by calling `data.matrix()`.

Data Frames

```
1 x <- data.frame(foo = 1:4, bar = c(T, T, F, F))
2
3 x
```

```
  foo   bar
1   1  TRUE
2   2  TRUE
3   3 FALSE
4   4 FALSE
```

```
1 nrow(x)
```

```
[1] 4
```

```
1 ncol(x)
```

```
[1] 2
```

Names

R objects can also have names, which is very useful for writing readable code and self-describing objects.

```
x <- 1:3
names(x)
```

```
NULL
```

```
names(x) <- c("one", "two", "three")
x
```

```
  one   two three 
    1     2     3 
```

```
names(x)
```

```
[1] "one"   "two"   "three"
```

Names

Lists can also have names.

```
1 x <- list(a = 1, b = 2, c = 3)
2 x
```

```
$a
[1] 1

$b
[1] 2

$c
[1] 3
```

Names

And matrices.

```r
m <- matrix(1:4, nrow = 2, ncol = 2)
dimnames(m) <- list(c("a", "b"), c("c", "d"))

m
```

```
  c d
a 1 3
b 2 4
```

Names

And data frames

```
1 head(mtcars)
```

```
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
1 names(mtcars)
```

```
 [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
[11] "carb"
```

```
1 row.names(mtcars)[1:10]
```

```
 [1] "Mazda RX4"          "Mazda RX4 Wag"      "Datsun 710"
 [4] "Hornet 4 Drive"     "Hornet Sportabout" "Valiant"
 [7] "Duster 360"         "Merc 240D"          "Merc 230"
[10] "Merc 280"
```

Summary

Data Types

- atomic classes: numeric, logical, character, integer, complex
- vectors, lists
- factors
- missing values
- data frames
- names