Elements of the R programming language – 1

Marcin Kierczak with Thomas Källman (labs)

21 September 2016

Introduction

Today, we will talk about various elements of a programming language and see how they are realized in R.

Contents of the lecture

- **variables and their types**
- **operators**
- vectors
- numbers as vectors
- strings as vectors
- matrices
- lists
- data frames
- objects
- repeating actions: iteration and recursion
- decision taking: control structures
- functions in general
- variable scope
- core functions

Variables

Creating a variable == assigning a name to data...

```
7 + 9
```

```
## [1] 16
```

```
a <- 7
a
```

```
## [1] 7
```

```
b <- 9
b
```

```
## [1] 9
```

```
c <- a + b
c
```

Variables cted.

We are not constrained to numbers. . .

```r
text1 <- 'a'
text2 <- 'qwerty'
text1
```

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

```r
text2
```

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

Variables – naming conventions

How to write variable names?

- What is legal/valid?
- What is a good style?

A syntactically valid name consists of letters, numbers and the dot or underline characters and starts with a letter or the dot not followed by a number.

Names such as ".2way" are not valid, and neither are the so-called *reserved words*.

Reserved words

Reserved words, are:

`if`, `else`, `repeat`, `while`, `function`, `for`, `in`, `next`, `break`, `TRUE`, `FALSE`, `NULL`, `Inf`, `NaN`, `NA`, `NA_integer_`, `NA_real_`, `NA_complex_`, `NA_character_`

and you also **cannot** use: `c`, `q`, `t`, `C`, `D`, `I`

and you **should not** use: `T`, `F`

Variables – good style

- make them informative, e.g. `genotypes` instead of `fsjht45jkhsdf4`,
- use consistent notation across your code – the same *naming convention*,
- camelNotation vs. dot.notation vs. dash_notation
- I used to use the camelNotation and the dot.notation and I'm still hesitating :-),
- do not `give.them.too.long.names`,
- in the dot notation avoid `my.variable.2`, use `my.variable2` instead,
- there are certain customary names: `tmp` - for temporary variables; `cnt` for counters; `i,j,k` within loops, `pwd` - for password. . .

Variables have types

We have already discussed the system of types in general. Now, time to look at the types system in R.

A numeric that stores numbers of different *types*:

```
x = 41.99 # assign 41.99 to x
class(x)
```

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

```
mode(x) # representation
```

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

```
typeof(x)
```

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

Class, type, representation and soorage mode

1. *class* is the point of view of object-oriented programming in R.

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

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

any generic function that has an "integer" method can be used.

2. `typeof()` gives the "type" of object from R's point of view.
3. `mode()` gives the "type" of object from the point of view of the S language.
4. `storage.mode()` is useful when passing R objects to compiled code, e.g. C.

Variables have types cted.

```r
y = 12 # now assign an integer value to y
class(y) # still numeric
```

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

```r
typeof(y) # an integer, but still a double!
```

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

Even integers are stored as double by default.
Numeric == double == real.

Variables have types cted.

```r
x <- as.integer(x) # type conversion, casting
typeof(x)
```

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

```r
class(x)
```

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

```r
is.integer(x)
```

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

One rarely works explicitly with integers though...

Be careful when casting

```r
pi <- 3.1415926536 # assign approximation of pi to pi
pi
```

```
## [1] 3.141593
```

```r
pi <- as.integer(pi) # not-so-careful casting
pi
```

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

```r
pi <- as.double(pi) # trying to rescue the situation
pi
```

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

Casting is not rounding

```r
as.integer(3.14)
```

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

```r
as.integer(3.51)
```

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

Ceiling, floor and a round corner

```r
floor(3.51) # floor of 3.51
```

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

```r
ceiling(3.51) # ceiling of 3.51
```

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

```r
round(3.51, digits = 1) # round to one decimal
```

```
## [1] 3.5
```

What happens if we cast a string to a number

```r
as.numeric('4.5678')
```

```
## [1] 4.5678
```

```r
as.double('4.5678')
```

```
## [1] 4.5678
```

```r
as.numeric('R course is cool!')
```

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

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

Special values

```r
-1/0  # Minus infinity
```

```
## [1] -Inf
```

```r
1/0 # Infinity
```

```
## [1] Inf
```

Special values cted.

```r
112345^67890  # Also infinity for R
```

```
## [1] Inf
```

```r
1/2e78996543  # Zero for R
```

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

```r
Inf - Inf # Not a Number
```

```
## [1] NaN
```

Complex number type

Core R supports complex numbers.

```
z = 7 + 4i # create a complex number
z
```

```
## [1] 7+4i
```

```
class(z)
```

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

```
typeof(z)
```

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

```
is.complex(z)
```

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

Complex number type cted.

```r
sqrt(-1) # not treated as cplx number
```

```
## Warning in sqrt(-1): NaNs produced
```

```
## [1] NaN
```

```r
sqrt(-1 + 0i) # now a proper cplx number
```

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

```r
sqrt(as.complex(-1)) # an alternative way
```

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

Logical type

```
a <- 7 > 2
b <- 2 >= 7
a
```

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

```
b
```

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

```
class(a)
```

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

```
typeof(a)
```

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

Logical type cted.

R has three logical values: TRUE, FALSE and NA.

```
x <- c(NA, FALSE, TRUE)
names(x) <- as.character(x)
outer(x, x, "&") # AND table
```

```
##        <NA> FALSE  TRUE
## <NA>     NA FALSE    NA
## FALSE FALSE FALSE FALSE
## TRUE     NA FALSE  TRUE
```

Logical type cted.

```
x <- TRUE
x
```

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

```
x <- T # also valid
x
```

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

```
is.logical(x)
```

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

```
typeof(x)
```

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

Logical as number

It is **very important** to remember that **logical type is also a numeric**!

```
x <- TRUE
y <- FALSE
x + y
```

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

```
2 * x
```

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

```
x * y
```

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

A trap set up for you

Never ever use variable names as T or F. Why?

```
F <- T
T
```

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

```
F
```

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

Maybe applicable in politics, but not really in science...

Character type

It is easy to work with characters and strings:

```
character <- 'c'
text <- 'This is my first sentence in R.'
text
```

```
## [1] "This is my first sentence in R."
```

```
character
```

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

```
class(character)
```

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

```
typeof(text) # also of 'character' type
```

Character type

```
number <- 3.14
number.text <- as.character(number) # cast to char
number.text
```

```
## [1] "3.14"
```

```
class(number.text)
```

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

```
as.numeric(number.text) # and the other way round
```

```
## [1] 3.14
```

Basic string operations

```r
text1 <- "John had a yellow "
text2 <- "submarine"
result <- paste(text1, text2, ".", sep='')
result
```

```
## [1] "John had a yellow submarine."
```

```r
sub("submarine", "cab", result)
```

```
## [1] "John had a yellow cab."
```

```r
substr(result, start = 1, stop = 5)
```

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

Basic printing

```
txt <- "blue"
val <- 345.78
sprintf("The weight of a %s ball is  %g g", txt, val)
```

```
## [1] "The weight of a blue ball is  345.78 g"
```