Variables, Data types & Operators

Elements of the R language

Marcin Kierczak

Introduction

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

Contents

- **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 is simply assigning a name to some structure that stores data...

```
7 + 9
a <- 7
a
b <- 9
b
c <- a + b
c
```

```
## [1] 16
## [1] 7
## [1] 9
## [1] 16
```

Variables cted.

We are not constrained to numbers...

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

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

Is `<-` equivalent to `=`? Which one shall I use?

- `val <- 3`, `val = 3` and `3 -> val` are three equivalent ways of assigning in R, But, it's best to use only `<-` to avoid possible confusion. The equal sign `=` should be used when setting function arguments ie; `f(a = 3)`.

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

- `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

- So, how to name variables?
- 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,
- 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

A *numeric* stores numbers of different *types*:

```
x <- 41.99 # assign 41.99 to x
typeof(x)
```

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

Classes, types, and modes

- `class` what type of object is it for R,
- `typeof()` what R thinks it is,
- `mode()` how S language would see it (backward compatibility),
- `storage.mode()` how is it stored in the memory; useful when talking to C or Java,

```
x <- 1:3
class(x)
typeof(x)
mode(x)
storage.mode(x)
```

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

Type casting

By default, any *numeric* is stored as *double* !

```
y=12 # now assign an integer value to y
class(y) # still numeric
typeof(y) # an integer, but still a double!
```

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

But we can explicitly **cast it** to integer:

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

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

> We need **casting** because sometimes a function requires data of some type!

More on type casting

Be careful when casting!

```
pi <- 3.1415926536 # assign approximation of pi to pi
pi
pi <- as.integer(pi) # not-so-careful casting
pi
pi <- as.double(pi) # trying to rescue the situation
pi
```

```
## [1] 3.141593
## [1] 3
## [1] 3
```

Casting is not rounding!

```
as.integer(3.14)
as.integer(3.51)
```

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

Ceiling, floor and a round corner

```
floor(3.51) # floor of 3.51
ceiling(3.51) # ceiling of 3.51
round(3.51, digits=1) # round to one decimal
```

```
## [1] 3
## [1] 4
## [1] 3.5
```

What happens if we cast a string to a number

```
as.numeric('4.5678')
as.double('4.5678')
as.numeric('R course is cool!')
```

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

Special values

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

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

and also:

```
112345^67890  # Also infinity for R
1/2e78996543  # Zero for R
Inf - Inf # Not a Number
```

```
## [1] Inf
## [1] 0
## [1] NaN
```

Complex number type

Core R supports complex numbers.

```
z = 7 + 4i # create a complex number
z
class(z)
typeof(z)
is.complex(z)
```

```
## [1] 7+4i
## [1] "complex"
## [1] "complex"
## [1] TRUE
```

```
sqrt(-1) # not treated as cplx number
sqrt(-1 + 0i) # now a proper cplx number
sqrt(as.complex(-1)) # an alternative way
```

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

Logical type

```
a <- 7 > 2
b <- 2 >= 7
a
b
class(a)
typeof(a)
```

```
## [1] TRUE
## [1] FALSE
## [1] "logical"
## [1] "logical"
```

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

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

	TRUE	FALSE	NA
TRUE	TRUE	FALSE	NA
FALSE	FALSE	FALSE	FALSE
NA	NA	FALSE	NA

Logical type cted.

```
x <- TRUE
x
x <- T # also valid
x
is.logical(x)
typeof(x)
```

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

- Observe that in R the logical type is also a numeric!

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

```
## [1] 1
## [1] 2
## [1] 0
```

A trap set up for you

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

```
F <- T
T
F
```

```
## [1] TRUE
## [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
character
class(character)
typeof(text) # also of 'character' type
```

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

Character type

```r
number <- 3.14
number.text <- as.character(number) # cast to char
number.text
class(number.text)
as.numeric(number.text) # and the other way round
```

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

Basic string operations

```
text1 <- "John had a yellow "
text2 <- "submarine"
result <- paste(text1, text2, ".", sep='')
result
sub("submarine", "cab", result)
substr(result, start=1, stop=5)
```

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

See you at the next lecture!

Graphics from freepik.com

Created: 09-Aug-2023 • Roy Francis • SciLifeLab • NBIS