Introduction to R

Instructor: Selina Baldauf

Freie Universität Berlin - Theoretical Ecology

2021-08-01 (updated: 2022-08-26)

R as a calculator

Arithmetic operators

Addition	+
Subtraction	-
Multiplication	*
Division	/
Modulo	%%
Power	^

```r
# Addition
2 + 2
# Subtraction
5.432 - 34234
# Multiplication
33 * 42
# Division
3 / 42
# Modulo (Remainder)
2 %% 2
# Power
2^2
# Combine operations
((2 + 2) * 5)^(10 %% 10)
```

R as a calculator

Relational operators

Equal to	`==`
Not equal to	`!=`
Less than	`<`
Greater than	`>`
Less or equal than	`<=`
Greater or equal than	`>=`

```
2 == 2
## [1] TRUE
2 != 2
## [1] FALSE
33 <= 32
## [1] FALSE
20 < 20
## [1] FALSE
```

R as a calculator

Logical operators

Not	`!`

```
!TRUE
## [1] FALSE
!(3 < 1)
## [1] TRUE
```

R as a calculator

Logical operators

Not	`!`
And	`&`

```
(3 < 1) & (3 == 3) # FALSE & TRUE = FALSE
## [1] FALSE
(1 < 3) & (3 == 3) # TRUE & TRUE = TRUE
## [1] TRUE
(3 < 1) & (3 != 3) # FALSE & FALSE = FALSE
## [1] FALSE
```

R as a calculator

Logical operators

Not	`!`
And	`&`
Or	`|`

```
(3 < 1) | (3 == 3) # FALSE | TRUE = TRUE
## [1] TRUE
(1 < 3) | (3 == 3) # TRUE | TRUE = TRUE
## [1] TRUE
(3 < 1) | (3 != 3) # FALSE | FALSE = FALSE
## [1] FALSE
```

Comments in R

```
# Reading and cleaning the data ------------------

data <- read_csv("data/my-data.csv")
# clean all column headers
# (found on https://stackoverflow.com/questions/68177507/)
data <- janitor::clean_names(data)

# Analysis ---------------------------------------
```

- Everything that follows a `#` is a comment
- Comments are not evaluated
- Notes that make code more readable or add information
- Comments can be used for
  - Explanation of code (if necessary)
  - Include links, names of authors, ...
  - Mark different sections of your code (💡 try `Ctrl/Cmd + Shift + R`)

Objects and data types in R

Variables

- Store values under meaningful names **to reuse** them
- A variable has a name and value and is created using the assignment operator

radius <- 5

- Variables are available in the global environment
- R is case sensitive: **r**adius != **R**adius
- Variables can hold any R objects, e.g.
    - Simple numbers
    - Tables with data
    - ...
- Choose meaningful variable names
    - Make your code easier to read

Variables

```r
# create a variable
radius <- 5
# use it in a calculation and save the result
# pi is a built-in variable that comes with R
circumference <- 2 * pi * radius
# change value of variable radius
radius <- radius + 1
```

```r
# just use the name to print the value to the console
radius
```

Atomic data types

There are 6 so-called **atomic data types** in R. The 4 most important are:

Numeric: There are two numeric data types:

- Double: can be specified in decimal (`1.243` or `-0.2134`), scientific notation (`2.32e4`) or hexadecimal (`0xd3f1`)
- Integer: numbers that are not represented by fraction. Must be followed by an `L` (`1L`, `2038459L`, `-5L`)

Logical: only two possible values `TRUE` and `FALSE` (abbreviation: `T` or `F` - but better use non-abbreviated form)

Character: also called string. Sequence of characters surrounded by quotes (`"hello"` , `"sample_1"`)

Vectors

Vectors

Vectors are data structures that are built on top of atomic data types.

Imagine a vector as a **collection of values** that are all **of the same data type**.

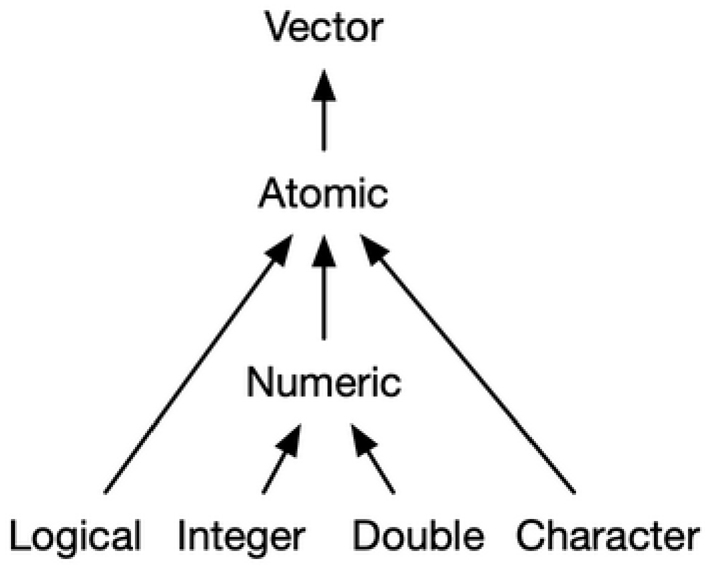

Image from Advanced R book

Creating vectors: `c()`

Use the function `c()` to *combine* values into a vector

```
lgl_var <- c(TRUE, TRUE, FALSE)
dbl_var <- c(2.5, 3.4, 4.3)
int_var <- c(1L, 45L, 234L)
chr_var <- c("These are", "just", "some strings")
```

You can also combine multiple vectors into one:

```
# Combine multiple vectors
v1 <- c(1,2,3)
v2 <- c(800, 83, 37)
v3 <- c(v1, v2)
```

Creating vectors: `c()`

Be aware of implicit **type conversion** when combining vectors of different types

```
# integer + logical -> integer (same with double + logical)
c(int_var, lgl_var)
## [1]   1  45 234   1   1   0

# integer + character -> character (same with double + character)
c(int_var, chr_var)
## [1] "1"            "45"           "234"          "These are"    "just"         "some strings"

# logical + character -> character
c(lgl_var, chr_var)
## [1] "TRUE"         "TRUE"         "FALSE"        "These are"    "just"         "some strings"
```

Creating vectors: `:` and `seq()`

The `:` operator creates a sequence between two numbers with an increment of (-)1

```
1:10 # instead of c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
##  [1]  1  2  3  4  5  6  7  8  9 10
```

The `seq()` function creates a sequence of values

```
seq(from = 1, to = 10, by = 1) # specify increment of sequence with by
##  [1]  1  2  3  4  5  6  7  8  9 10
seq(from = 1, to = 10, length.out = 20) # specify desired length with length.out
##  [1]  1.000000  1.473684  1.947368  2.421053  2.894737  3.368421  3.842105  4.315789  4.789474
## [10]  5.263158  5.736842  6.210526  6.684211  7.157895  7.631579  8.105263  8.578947  9.052632
## [19]  9.526316 10.000000
```

Creating vectors: `rep()`

Repeat values multiple times with `rep()`

```
rep("hello", times = 5)
## [1] "hello" "hello" "hello" "hello" "hello"
```

Working with vectors

Let's create some vectors to work with.

```
# list of 10 biggest cities in Europe
cities <- c("Istanbul", "Moscow", "London", "Saint Petersburg", "Berlin", "Madrid", "Kyiv",
"Rome", "Bucharest", "Paris")

population <- c(15.1e6, 12.5e6, 9e6, 5.4e6, 3.8e6, 3.2e6, 3e6, 2.8e6, 2.2e6, 2.1e6)

area_km2 <- c(2576, 2561, 1572, 1439,891,604, 839, 1285, 228, 105 )
```

We can check the length of a vector using the `length()` function:

```
length(cities)
## [1] 10
```

Data from Wikipedia

Working with vectors

Divide the population and area vector to calculate the population density in each city:

```
population / area_km2
##  [1]  5861.801  4880.906  5725.191  3752.606  4264.871  5298.013  3575.685  2178.988  9649.123
## [10] 20000.000
```

The operation is performed **separately for each element of the two vectors** and the result is a vector.

Same, if a **vector is divided by vector of length 1** (i.e. a single number). Result is always a vector.

```
mean_population <- mean(population) # calculate the mean of population vector
mean_population
## [1] 5910000
population / mean_population # divide population vector by the mean
##  [1] 2.5549915 2.1150592 1.5228426 0.9137056 0.6429780 0.5414552 0.5076142 0.4737733 0.3722504
## [10] 0.3553299
```

Working with vectors

We can also work with relational and logical operators

```
population > mean_population
##  [1]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

The result is a vector containing `TRUE` and `FALSE`, depending on whether the city's population is larger than the mean population or not.

Logical and relational operators can be combined

```
# population larger than mean population OR population larger than 3 million
population > mean_population | population > 3e6
##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE
```

Working with vectors

Check whether elements occur in a vector:

```
cities == "Istanbul" | cities == "Berlin" | cities == "Madrid"
##  [1]  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
```

The `%in%` operator checks whether *multiple* elements occur in a vector.

```
# for each element of cities, checks whether that element is contained in to_check
to_check <- c("Istanbul", "Berlin", "Madrid")
cities %in% to_check
##  [1]  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
```

```
# for each element of to_check, check whether that element is contained in cities
to_check %in% cities
## [1] TRUE TRUE TRUE
```

`%in%` always returns a vector of the same length as the vector on the left side

Indexing vectors

You can use square brackets `[]` to access specific elements from a vector.

The basic structure is:

vector [vector of indexes to select]

```
cities[5]
## [1] "Berlin"
```

```
# the three most populated cities
cities[1:3] # same as cities[c(1,2,3)]
## [1] "Istanbul" "Moscow"   "London"
```

```
# the last entry of the cities vector
cities[length(cities)] # same as cities[10]
## [1] "Paris"
```

Indexing vectors

Change the values of a vector at specified indexes using the assignment operator `<-`

Imagine for example, that the population of

- Istanbul (index 1) increased to 20 Million
- Rome (index 8) changed but is unknown
- Paris (index 10) decreased by 200,000

```
# first copy the original vector to leave it untouched
population_new <- population
# Update Istanbul (1) and Rome(8)
population_new[c(1, 8)] <- c(20e6, NA) # NA means missing value
# Update Paris (10)
population_new[10] <- population_new[10] - 200000
```

```
population_new
##  [1] 20000000 12500000  9000000  5400000  3800000  3200000  3000000       NA  2200000  1900000
```

Indexing vectors

You can also index a vector using logical tests. The basic structure is:

vector [logical vector of same length]

```
mega_city <- population > mean_population
mega_city
##  [1]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

Which are the mega cities?

```
cities[mega_city] # or short: cities[population > mean_population]
## [1] "Istanbul" "Moscow"   "London"
```

Index only the cities that return `TRUE` for the comparison of their population against the mean population

Indexing vectors

We also use `%in%` for logical indexing.

```
population[ cities %in% c("Berlin", "Paris", "Stockholm", "Madrid")]
## [1] 3800000 3200000 2100000
```

- Returns only 3 values for population, because Stockholm is not a city in our vector
  - No city in `cities` returns `TRUE` for the comparison with `"Stockholm"`

```
cities %in% c("Berlin", "Paris", "Stockholm", "Madrid")
##  [1] FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE
```

Summary I

- Variables have a name and a value and are created using the assignment operator `<-`, e.g.

```
radius <- 5
```

- Vectors are a collection of values of the same data type:
  - character (`"hello"`)
  - numeric: integer (`23L`) and double (`2.23`)
  - logical (`TRUE` and `FALSE`)
- Be careful about implicit type conversion:
  - numeric to character
  - logical to character
  - logical to numeric

Summary II

Data types

```
# check the data type of a variable
typeof("hello")

# check if a variable is of a certain data type
is.*()
is.integer(1L)

# convert a variable into a certain data type
as.*()
as.logical("hello")
```

Summary III

Create vectors

```
# combine objects into vector
c(1,2,3)

# create a sequence of values
seq(from = 3, to = 6, by = 0.5)
seq(from = 3, to = 6, length.out = 10)
2:10

# repeat values from a vector
rep(c(1,2), times = 2)
rep(c("a", "b"), each = 2)
```

Summary IV

Indexing and subsetting vectors

```
# By index
v[3]
v[1:4]
v[c(1,5,7)]

# Logical indexing with 1 vector
v[v > 5]
v[v != "bird" | v == "rabbit"]
v[v %in% c(1,2,3)] # same as v[v == 1 | v == 2 | v == 3]

# Logical indexing with two vectors of same length
v[y == "bird"] # return the value in v for which index y == "bird"
v[y == max(y)] # return the value in v for which y is the maximum of y
```

Summary V

Working with vectors

```
# length
length(v)
# rounding numbers
round(v, digits = 2)
# sum
sum(v)
# mean
mean(v)
# median
median(v)
# standard deviation
sd(v)
# find the min value
min(v)
# find the max value
```

Now you

Task 2: Data types and vectors (35 min)

Find the task description here