Úvod, instalace R a Rstudia, základní datové typy a struktury, operace

—

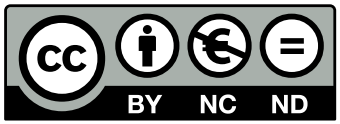
B03128 – Úvod do skriptovacího jazyka R (zkrácená online verze)

Lubomír Štěpánek[1, 2]

[1]Oddělení biomedicínské statistiky
Ústav biofyziky a informatiky
1. lékařská fakulta
Univerzita Karlova v Praze

ČVUT FBMI

[2]Katedra biomedicínské informatiky
Fakulta biomedicínského inženýrství
České vysoké učení technické v Praze



Obsah

Organizace předmětu

- volitelný předmět (3 kreditové body) v online zkrácené podobě
- zakončen seminární prací (analýza dat v R, 1–2 stránky komentáře)
- předpokládaný průběh
  - (i) instalace, datové typy a struktury, operace s nimi
  - (ii) podmínky, cykly, varování, import a export dat do a z R
  - (iii) vestavěné a uživatelské funkce
  - (iv) exploratorní analýza dat, testování hypotéz
  - (v) analýza rozptylu, korelace, lineární regrese
  - (vi) logistická regrese a další modely

githubí stránka předmětu ▸ GitHub

https://github.com/LStepanek/B03128_Uvod_do_skriptovaciho_jazyka_R

Co je R

- R je interpretovaný programovací jazyk
- kombinuje několik paradigmat
  - imperativní
  - funkcionální
  - objektové
- R je *domain specific language* – je určen pro statistickou analýzu dat a jejich grafické zobrazení
- R je open-source, konkrétně *free-as-in-beer* a *free-as-in-speech*

Stažení a instalace jádra R

- na stránkách R-projectu

  https://www.r-project.org/

  postupně **download R**, vyberme českou doménu a stáhněme desktopově
- poté instalujme dle instrukcí do předvolené složky

Stažení a instalace RStudia

- RStudio je jedním z grafických IDE (Integrated Development Environment) jazyka R
- na stránkách RStudia

https://www.rstudio.com/

postupně **Products** > **RStudio** > **Desktop** > **Open Source Edition** > **Free** > **Download**, stáhněme desktopově

- poté instalujme dle instrukcí do předvolené složky

Ahoj světe!

- do skriptu či konzole napišme

```
print("hello world")
```

- dostaneme

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

Práce s nápovědou

- nápovědu pro funkci či objekt získáme pomocí příkazu `help()`, kde argumentem je název funkce či objektu

```
help(print)
```

- nebo předsazením symbolu `?` před název funkce či objektu

```
?print
```

- předsazením symbolů `??` před název funkce či objektu prohledáme veškeré dokumenty nápovědy

```
??print
```

- vždy je zavolán HTML soubor s volnotextovou nápovědou

Datové typy

- numerická hodnota (`numeric`)
- celé číslo (`integer`)
- komplexní číslo (`complex`)
- logická hodnota (`logical`)
- textový řetězec (`character`)
- `NA`, `NULL`, `NaN`

Numerická hodnota

- v R jako `numeric`
- libovolné $x \in \mathbb{R}$ uložené s danou přesností
- odpovídá datovému typu `double` s 64 bitovou přesností, který je běžný v jiných jazycích
- např.

```
5;  -13.8,  4.5578e15
```

- zda je hodnota typu `numeric`, zjistíme pomocí

```
is.numeric(-13.8)     # TRUE
class(-13.8)          # "numeric"
class(Inf)            # "numeric"
```

- vhodná pro různorodé operace (viz dále)

Celé číslo

- v R jako `integer`
- libovolné $z \in \mathbb{Z}$ uložené s danou přesností
- např.

```
5L; 13L, -5L
```

- zda je hodnota typu `integer`, zjistíme pomocí

```
is.integer(-13L)    # TRUE
class(-13L)         # "integer"
is.integer(-13)     # FALSE
class(-13)          # "numeric"
```

- přetypování celého čísla na reálné (`numeric`) pomocí

```
as.numeric(5L)
```

- pozor! v R mají celá čísla pouze 16 bitovou přesnost
- pro práci s velkými celými čísly nutné balíčky `gmp` či `int64` (zvýší bitovou přesnost uložených celých čísel)

Logická hodnota

- v R jako `logical`
- libovolné booleovské $x \in \{\texttt{TRUE}, \texttt{FALSE}\}$
- např.

```
TRUE; FALSE; T; F
```

- zda je hodnota typu `logical`, zjistíme pomocí

```
is.logical(TRUE)        # TRUE
class(FALSE)            # "logical"
class("TRUE")           # "character"
class(T)                # "logical"
class(F)                # "logical"
```

Textový řetězec

- v R jako `character`
- libovolná sekvence znaků (extended ASCII) uzavřená mezi jednoduchými či dvojitými uvozovkami
- např.

```
"ahoj"; 'xweiwogw23425ng'; ""
```

- zda je hodnota typu `character`, zjistíme pomocí

```
is.character("ahoj")   # TRUE
class("bla bla")       # "character"
class("123")           # "character"
class(123)             # "numeric"
is.numeric(Inf)        # TRUE
is.numeric("Inf")      # FALSE
```

- na textový řetězec lze převést libovolnou jinou hodnotu pomocí

```
as.character(123)
```

NA, NULL, NaN

- `NA` je hodnota typu `Not Available`, obvykle chybějící hodnota
- `NULL` je null object, používá se pro bezhodnotovou inicializaci objektu (uvidíme později)
- `NaN` je hodnota typu `Not a Number`, obvykle nevyjádřitelný výsledek matematické operace
- množinově platí $\{\texttt{NaN}\} \subseteq \{\texttt{NA}\}$
- např.

```
1  log(-1)              # NaN
2  is.na(NaN)           # TRUE
3  is.nan(NA)           # FALSE
4  is.nan(1 / 0)        # FALSE
5  1 / 0                # Inf
```

Přiřazení hodnoty k proměnné

- přiřadit hodnotu nějaké proměnné lze pomocí jednoduchého rovnítka

```
1        x = 5
```

- nebo pomocí orientované šipky

```
1        x <- 5
2        5 -> x        # totéž
```

- anebo pomocí funkce `assign()`, kde prvním argumentem je název proměnné (tedy textový řetězec) a druhým hodnota

```
1        assign("x", 5)   # analogické k x <- 5 či x = 5
```

to se hodí zejména u dynamického iterování (viz později)

Datové struktury

- vektor (`vector`)
- faktor (`factor`)
- matice (`matrix`)
- tabulka dat (`data.frame`)
- seznam (`list`)

Tvorba vektorů a základní příkazy

- vektor je jednorozměrný výčet prvků stejného datového typu, nemá orientaci ve smyslu řádek či sloupec
- vektor je objekt typu *tuple*, tedy zachovává pořadí svých prvků (na rozdíl od objektů typu *set*)
- lze vytvořit pomocí generické funkce `c()`, neboli *concatenate*
- např.

```
c()              # prázdný vektor
length(c())      # 0
c(3, 1, 2)       # vektor o délce 3 a prvcích 3, 1, 2
c("a", "d")      # vektor o dél. 2 a prvcích "a", "d"
```

- pomocí funkce `c()` lze vektory i prodlužovat

```
c(c(3, 1, 2), 4) # vektor o prvcích 3, 1, 2, 4
c(3, 1, 2, 4)    # zkráceně totéž
```

Tvorba vektorů a základní příkazy

- vektor tedy lze prodloužit libovolně o jednu či více hodnot

```
x <- c(3, 1, 2)
length(x)          # 3
y <- 1
z <- c(2)
w <- c(5, 7)
x <- c(x, y)       # prodloužení vektoru x
                   # o hodnotu y
w <- c(w, z)       # prodloužení vektoru w
                   # o vektor z
                   # jednoprvkový vektor je
                   # skalárem, jednou hodnotou
c <- c(1, 2, 3)
c                  # vektor o prvcích 1, 2, 3
                   # byť je c referovaný termín,
                   # funkce c je zachována
                   # a vznikl vektor c
```

Vektory textových řetězců

- vektory obsahující textové hodnoty, lze je použít např. jako názvy prvků jiného vektoru

```
1  x <- c(3, 1, 2)
2  y <- c("a", "b", "c")
3  names(x) <- y      # pojmenuje prvky
4                     # vektoru x
5  x
6  unname(x)          # zbaví prvky vektoru
7                     # x jeho jejich jmen
8  setNames(x, y)     # opět pojmenuje
9                     # prvky vektoru x
```

Subvektory, indexování, adresace

- R indexuje vektory od 1, nikoliv od 0 (první prvek má index 1, druhý index 2, apod.)
- adresujeme pomocí hranatých závorek [ ]

```
x <- c(4, 2, 6, -3)
x[1]                # 4
x[1:2]              # c(4, 2)
x[5]                # NA
x[length(x)]        # -3
x[c(1, 3, 4)]       # c(4, 6, -3)
x[length(x):1]      # c(-3, 6, 2, 4)
rev(x)              # totéž, c(-3, 6, 2, 4)
```

Logické vektory

- používají se (nejen) k adresování vhodných prvků

```
y <- c(TRUE, TRUE, FALSE, TRUE) # logický
                                # vektor
x <- c(3, 1, 2, 5)
x[y]                        # (sub)vektor c(3, 1)
x[c(F, T, F, T)]            # subvektor c(1, 5)
```

- výhodný je někdy tzv. *recycling*

```
z <- c("R", "G", "E", "F", "I")
z[c(T, F)]      # vybere pouze hodnoty
                # na lichých pozicích,
                # tedy "R", "E", "I"
                # neboli vektor
                # c("R", "E", "I")
```

Faktory

- vektory textových hodnot, kde každá hodnota patří do své kategorie

```
x <- factor(
    c("muž", "žena", "muž", "muž")
)           # pořadí kategorií je defaultně
            # abecední
x <- factor(
    c("muž", "žena", "muž", "muž"),
    levels = c("žena", "muž")
)           # zde si pořadí kategorií
            # určíme sami
```

- nad faktory snadno vytvoříme kontingenční tabulku

```
table(x)    # x
            # žena muž
            # 1 3
```

Aritmetické operace

operace	operátor	příklad
sčítání	+	2 + 3
odčítání	-	2 - 3
násobení	*	2 * 3
dělení	/	2 / 3
mocnění	^ či **	2 ^ 3 či 2 ** 3
modulo[1]	%%	7 %% 3
celočíselné dělení	%/%	7 %/% 3

[1] zbytek po celočíselném dělení

Logické operace

- logické operace lze užít obecně nad výroky, tj. nad objekty s datovým typem `logical`
- operace *short* AND (operátor `&`)
  - použitelná pro vektory
  - vyhodnocuje všechny výroky vektoru a vrací jejich hodnoty

```
1  c(FALSE, FALSE, TRUE, TRUE) &
2  c(FALSE, TRUE, FALSE, TRUE)
3                    # c(FALSE, FALSE, FALSE, TRUE)
```

- operace *short* OR (operátor `|`)
  - použitelná pro vektory, vyhodnocuje všechny výroky vektoru a vrací jejich hodnoty

```
1  c(FALSE, FALSE, TRUE, TRUE) |
2  c(FALSE, TRUE, FALSE, TRUE)
3          # c(FALSE, TRUE, TRUE, TRUE)
```

Logické operace

- operace NOT (operátor `!`)
  - vrací výroku opačnou logickou hodnotu, než které výrok nabývá

```
! TRUE                    # FALSE
! 2 > 3                   # TRUE
```

- funkce `all()`
  - vrací pro vektor výroků TRUE právě tehdy, jsou-li všechny výroky rovné TRUE

```
all(c(3 > 2, 7 %% 3 == 1, 1 == 0))   # FALSE
all(c(3 > 2, 7 %% 3 == 1, 1 >= 0))   # TRUE
```

- funkce `any()`
  - vrací pro vektor výroků TRUE právě tehdy, je-li alespoň jeden z výroků roven TRUE

```
any(c(3 < 2, 7 %% 3 <= 0, FALSE))    # FALSE
any(c(3 < 2, 7 %% 3 >= 1, FALSE))    # TRUE
```

Operace porovnávání (komparace)

- pomocí operací porovnávání lze srovnat velikost či pořadí dvou objektů stejného datového typu; datový typ může přitom být v podstatě libovolný
- výsledkem operace porovnání je výrok, tedy hodnota datového typu `logical`
- porovnání typu *je rovno* (==, `all.equal()`, `identical()`)

```
1  2 == 3                          # FALSE
2  all.equal(c(1, 2), c(1, 2 + 1e-13),
3            tolerance = 1e-12)
4                                  # TRUE; porovnává vektory
5                                  # volitelnou danou tolerancí
6  identical(c(1, 2), c(1, 2 + 1e-13))
7                                  # FALSE, porovnává objekty
8                                  # a vrací TRUE jen při úplné
9                                  # shodě
```

Operace porovnávání (komparace)

- porovnání typu *je menší*, *je menší rovno*, *je větší*, *je větší rovno* (<, <=, >, >=)

```
2 < 3                    # TRUE
"b" <= "a"               # FALSE; porovnává pořadí
                         # v abecedě
FALSE >= TRUE            # FALSE; porovnává hodnotu
                         # v booelovské aritmetice
                         # (TRUE := 1, FALSE := 0)
```

- porovnání typu *není rovno*, *je různé od* (!=)

```
2 != 3                   # TRUE
TRUE != FALSE            # TRUE
```

Operace porovnávání (komparace)

- porovnání typu *je obsaženo ve* (%in%)

```
c(2, 6) %in% c(1:5)          # c(TRUE, FALSE)
"k" %in% LETTERS             # FALSE
"J" %in% letters             # FALSE
"May" %in% month.name        # TRUE
`%in%`("Jan", month.abb)     # TRUE; prefix notace
"a" %in% "abeceda"           # FALSE
```

- ekvivalentem (wrapperem) operace %in% je funkce `is.element()`

```
is.element(c(2, 6), c(1:5))
                             # c(TRUE, FALSE)
is.element(c(1:5), c(2, 6))
                             # c(FALSE, TRUE,
                             # FALSE, FALSE, FALSE)
```

- funkce typu *je pravdou* (`isTRUE()`)

```
isTRUE(3^2 > 2^3)            # TRUE
```

Vestavěné matematické funkce

- jde o funkce balíčků `base`, `stats` a dalších, je jich obrovské množství
- např.

```
abs(), sign()
acos(), asin(), atan()
sin(), cos(), tan()
ceiling(), floor(), trunc()
exp(), log(), log10(), log2(), sqrt()
max(), min(), prod(), sum()
cummax(), cummin(), cumprod(), cumsum(),
    diff()
pmax(), pmin()
range()
mean(), median(), cor(), sd(), var()
rle()
```

Zaokrouhlování, formátování čísel

- zaokrouhlení čísla `x` pomocí `round(x, digits)` na `digits` desetinných míst

```
round(1.4, digits = 0)        # 1
round(-146.655, 2)            # -146.66
```

- zaokrouhlení čísla `x` pomocí `signif(x, digits)` na `digits` platných cifer

```
signif(1.458, digits = 1)     # 1
signif(1.458, digits = 2)     # 1.5
signif(1.458, digits = 3)     # 1.46
signif(1.458, digits = 4)     # 1.458
```

- formátování čísla `x` pomocí `format(x, nsmall)` na `nsmall` fixních desetinných cifer

```
format(1.45, nsmall = 1)      # "1.45"
format(1.45, nsmall = 2)      # "1.45"
format(1.45, nsmall = 3)      # "1.450"
```

Tvorba matic a základní příkazy

- matice (`matrix`) je dvojrozměrným polem, které obsahuje prvky stejného datového typu
- všechny sloupce matice mají shodnou délku, podobně všechny řádky matice mají shodnou délku
- ať je

$$\boldsymbol{A} = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} \qquad \boldsymbol{B} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

- v R matici $\boldsymbol{A}$ a $\boldsymbol{B}$ získáme příkazy

```
1       A <- matrix(c(1, 2, 3, 4), nrow = 2,
2                   ncol = 2)
3       B <- matrix(c(1, 3, 2, 4), nrow = 2,
4                   ncol = 2)
5       B <- matrix(c(1, 2, 3, 4), nrow = 2,
6                   ncol = 2, byrow = TRUE)
7       # vždy jeden z argumentů "nrow" či "ncol" je zbytný
```

Manipulace s maticemi

- ať je

$$\boldsymbol{C} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{pmatrix}$$

- v R pak pomocí

```
C <- matrix(letters[1:12], nrow = 3,
            byrow = T)
```

- některé užitečné příkazy

```
is.matrix(C)    # TRUE
class(C)        # "matrix"
mode(C)         # "character"; datový typ pruků
str(C)          # chr [1:3, 1:4] "a" "e" "i" ...
dim(C)          # c(3, 4); rozměry matice C
```

Manipulace s maticemi

- ať je

$$\boldsymbol{C} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{pmatrix}$$

- další užitečné příkazy

```
colnames(C) <- c("c1", "c2", "c3", "c4")
rownames(C) <- c("r1", "r2", "r3")
                # přidá jmenovky sloupcům i řádkům
C <- unname(C)
                # zbaví sloupce i řádky jmenovek
dimnames(C) <- list(
                    c("r1", "r2", "r3"),
                    c("c1", "c2", "c3", "c4")
                )
                # opět přidá jmenovky sloupcům i řádkům
```

Manipulace s maticemi

- stále mějme

$$\boldsymbol{C} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{pmatrix}$$

- další užitečné příkazy

```
1  rbind(C, c("x", "x", "x", "x"))
2                  # přidání řádku c("x", "x", "x", "x")
3                  # k matici C
4  cbind(C, c("x", "x", "x"))
5                  # přidání sloupce c("x", "x", "x")
6                  # k matici C
7  C[-1, ]         # odebrání 1. řádku matici C
8  C[, -2]         # odebrání 2. sloupce matici C
```

Submatice, indexování, adresace

- ať je

$$\boldsymbol{C} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{pmatrix}$$

- v R pomocí

```
C <- matrix(letters[1:12], nrow = 3,
            byrow = T, dimnames = list(
            c("r1", "r2", "r3"),
            c("c1", "c2", "c3", "c4")))
```

- adresace

```
C[2, 3]          # "g"; prvek 2. řádku, 3. sloupce
C["r2", "c3"]    # "g"; prvek 2. řádku, 3. sloupce
C[1, ]           # c("a", "b", "c", "d");
                 # tedy vektor 1. řádku matice C
                 # s popisky
```

Submatice, indexování, adresace

- stále mějme

$$\boldsymbol{C} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{pmatrix}$$

- adresace

```
C[, 3]                  # c("c", "g", "k");
                        # tedy vektor 3. sloupce matice C
                        # s popisky
C[c(1, 3), c(2, 4)]
                        # matrix(c("b", "j", "d", "l"), 2)
                        # submatice 1. a 3. řádku,
                        # 2. a 4. sloupce matice C
                        # s popisky
C["r2", ]               # c("e", "f", "g", "h");
                        # tedy vektor 2. řádku matice C
                        # s popisky
```

Submatice, indexování, adresace

- stále mějme

$$\boldsymbol{C} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{pmatrix}$$

- adresace

```
C[dim(C)[1], dim(C)[2]]
                # "l"; obecná adresace prvku
                # vpravo dole (např. neznáme-li
                # číselné rozměry matice)
C[5]            # "f"; major-column ordering
C[c(8, 9)]      # c("g", "k")
C[13]           # NA
diag(C)         # c("a", "f", "k"); hlavní diagonála
diag(C[, dim(C)[2]:1])
                # c("d", "g", "j"); vedlejší diagonála
```

Maticová algebra

- buďte

$$\boldsymbol{A} = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} \qquad \boldsymbol{B} = \begin{pmatrix} 5 & 7 \\ 6 & 8 \end{pmatrix}$$

- v R

```
A <- matrix(c(1, 2, 3, 4), nrow = 2)
B <- matrix(c(5, 6, 7, 8), nrow = 2)
```

- Hadamardův součin (*element-wise*, *pairwise*) $\boldsymbol{A} \circ \boldsymbol{B} = \begin{pmatrix} 5 & 21 \\ 12 & 32 \end{pmatrix}$

```
A * B       # matrix(c(5, 12, 21, 32), 2)
```

- maticový součin $\boldsymbol{A} \cdot \boldsymbol{B} = \begin{pmatrix} 23 & 31 \\ 34 & 46 \end{pmatrix}$

```
A %*% B     # matrix(c(23, 34, 31, 46), 2)
```

- transpozice $\boldsymbol{A}^T = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$

Tvorba datových tabulek a základní příkazy

- datová tabulka (`data.frame`) je dvojrozměrným polem, které obsahuje v každém sloupci prvky stejného datového typu, ale jednotlivé sloupce se mohou datovým typem lišit
- všechny sloupce datové tabulky mají shodnou délku, podobně všechny řádky datové tabulky mají shodnou délku
- v R je řada vestavěných datových tabulek

```
1  mtcars
2  str(mtcars)
3  class(mtcars)              # "data.frame"
4  mode(mtcars)               # "list"
5  is.data.frame(mtcars)      # TRUE
6  str(iris)
7      # 'data.frame': 150 obs. of 5 variables
8      # ...
9  dim(iris)                  # c(150, 5)
```

Manipulace s datovými tabulkami

- užitečné příkazy

```
data <- mtcars
colnames(data)
colnames(data) <- paste("c",
                        1:dim(data)[2],
                        sep = "_")
rownames(data) <- paste("r",
                        1:dim(data)[1],
                        sep = "_")
                  # změní jmenovky sloupcům i řádkům
head(data)        # náhled na prvních 6 řádků
head(data, 10)
                  # náhled na prvních 10 řádků
tail(data)        # náhled na posledních 6 řádků
tail(data, 10)
                  # náhled na posledních 10 řádků
```

Manipulace s datovými tabulkami

- další užitečné příkazy

```
1   rbind(data, rep(0, dim(data)[2]))
2                 # přidání řádku c(0, 0, ..., 0)
3                 # k data.frameu "data"
4   cbind(data, rep(0, dim(data)[1]))
5                 # přidání sloupce c(0, 0, ..., 0)
6                 # k data.frameu "data"
7   data.frame(data,
8              "ahoj" = rep(0, dim(data)[1]))
9                 # přidání sloupce c(0, 0, ..., 0)
10                # se jménem "ahoj" k data.frameu
11                # "data"
12  data[-1, ]    # odebrání 1. řádku data.frameu
13                # "data"
14  data[, -1]    # odebrání 1. sloupce data.frameu
15                # "data"
```

Indexování, adresace

- adresace

```
data[2, 3]      # 160; prvek 2. řádku, 3. sloupce
data["r_2", "c_3"]
                # 160; prvek pod danými popisky
data[1, ]       # c(21, 6, 160, 110, ...);
                # tedy vektor 1. řádku data.frameu
                # "data" s popisky
data[, 2]       # c(6, 6, 4, 6, ...);
                # tedy vektor 2. sloupce
                # data.frameu "data" s popisky
data$c_5        # c(3.90, 3.90, 3.85, 3.08, ...);
                # tedy vektor 2. sloupce
                # data.frameu "data" s popisky
data$c_5[1]     # 3.9;
                # tedy první prvek 2. sloupce
                # data.frameu "data"
```

Indexování, adresace

- adresace

```
        data[dim(data)[1], dim(data)[2]]
                    # 2; obecná adresace pruku
                    # vpravo dole (např. neznáme-li
                    # číselné rozměry tabulky)
        data[5]     # 5. sloupec, nikoliv major-column
                    # ordering
```

Sloupcové přehledy

- může mít smysl znát agregovaný ukazatel nad všemi sloupci

```r
colSums(data)
                # součty všech sloupců
apply(data, 2, sum)
                # totéž
colMeans(data)
                # průměry všech sloupců
apply(data, 2, mean)
                # totéž
data <- rbind(data, rep(NA, dim(data)[2]))
                # přidán řádek c(NA, NA, ...)
colMeans(data)
                # c(NA, NA, ...); pro získání průměrů
                # nutné přidat argument na.rm = TRUE
colMeans(data, na.rm = TRUE)
apply(data, 2, mean, na.rm = TRUE)
                # už funguje
```

Tvorba seznamů a základní příkazy

- seznam (`list`) je výčtem prvků, které mohou být různorodého datového typu, včetně listu
- délky jednotlivých prvků seznamu se mohou lišit

```
1  my_list <- list("a" = c(1:10),
2                  "b" = mtcars,
3                  "c" = matrix(1:8, 2),
4                  "z" = "ahoj")
5  str(my_list)
6  class(my_list)          # "list"
7  mode(my_list)           # "list"
8  is.list(my_list)        # TRUE
```

Manipulace se seznamy

- užitečné příkazy

```
1   names(my_list)
2   names(my_list) <- LETTERS[
3       1:length(my_list)
4   ]               # přejmenovávám prvky listu
5
6   my_list[[length(my_list) + 1]] <- c(T, F)
7   names(length(my_list)) <- "XY"
8                   # přidání vektoru c(T, F)
9                   # k listu "my_list" pod jménem
10                  # "XY"
```

Indexování, adresace

- adresace

```
my_list[[2]]   # 2. prvek listu
my_list[["B"]]
                # prvek listu pod jmenovkou "B"
                # jde o původní datový typ
                # (data.frame)
my_list["B"]   # prvek listu pod jmenovkou "B"
                # jde (vždy) o list
my_list[c(2, 4)]
                # 2. a 4. prvek listu
my_list$C      # prvek listu pod jmenovkou "C"
my_list[[1]][2]
                # 2; 2. prvek 1. prvku listu
my_list[[2]][3, 5]
                # 3.85; z 2. prvku listu vybírám
                # prvek o souřadnicích (3, 5)
```

Indexování, adresace pomocí funkce `lapply()`

- adresace

```
1   set.seed(1)
2   my_long_list <- lapply(
3       sample(c(80:120), 100, TRUE),
4       function(x) sample(
5           c(50:150), x, replace = TRUE
6       )
7   )    # list vektorů náhodné délky
8       # generovaných z náhodných čísel
9
10  lapply(my_long_list, "[[", 14)
11      # z každého prvku listu (vektoru)
12      # vybírám jen jeho 14. prvek
```

Prvkové přehledy

- může mít smysl znát agregovaný ukazatel nad všemi prvky seznamu

```r
lapply(my_long_list, mean)
    # pro každý prvek listu (vektor)
    # vracím jeho průměr

lapply(my_long_list, length)
    # pro každý prvek listu (vektor)
    # vracím jeho délku
```

Děkuji za pozornost!

lubomir.stepanek@lf1.cuni.cz
lubomir.stepanek@fbmi.cvut.cz

▸ GitHub

https://github.com/LStepanek/B03128_Uvod_do_skriptovaciho_jazyka_R