

Úvod, instalace R a Rstudio, základní datové typy a struktury operace, čísla, vektory a práce s nimi

—
17VSADR – Skriptování a analýza dat v jazyce R

Lubomír Štěpánek^{1, 2}



¹Oddělení biomedicínské statistiky & výpočetní techniky
Ústav biofyziky a informatiky
1. lékařská fakulta
Univerzita Karlova v Praze



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

(2017) Lubomír Štěpánek, CC BY-NC-ND 3.0 (CZ)



Dílo lze dále svobodně šířit, ovšem s uvedením původního autora a s uvedením původní licence. Dílo není možné šířit komerčně ani s ním jakkoliv jinak nakládat pro účely komerčního zisku. Dílo nesmí být jakkoliv upravováno. Autor neručí za správnost informací uvedených kdekoli v předložené práci, přesto vynaložil nezanedbatelné úsilí, aby byla uvedená fakta správná a aktuální, a práci sepsal podle svého nejlepšího vědomí a svých „nejlepších“ znalostí problematiky.

Obsah

- 1 Organizace
- 2 Úvod
- 3 Začínáme
- 4 Datové typy, jejich vlastnosti
- 5 Datové struktury
- 6 Vektory a operace s nimi

Organizace předmětu

- volitelný předmět (2 kreditové body)
- zakončen zápočtem
- rozsah 0 + 2 (vyučovacích hodin týdně)
- neoficiální účast na bázi dobrovolnosti vítána

Literatura



Karel Zvára. *Základy statistiky v prostředí R*. Praha, Česká republika: Karolinum, 2013. ISBN: 978-80-246-2245-3.



Hadley Wickham. *Advanced R*. Boca Raton, FL: CRC Press, 2015. ISBN: 978-1466586963.

Proč bych se právě já měl učit s R

- s vědou (nejen) v biomedicině to myslím vážně
- chtěl bych se věnovat postgraduálnímu doktorskému studiu, kde jistojistě budu potřebovat statistiku
- jsem postgraduální student a jednou bych chtěl dokončit postgraduální doktorské studium, k čemuž tu zatracenou statistiku opravdu potřebuji
- jsem lékař a hledám konečně efektivní nástroj pro analýzy svých výzkumů
- komerční statistické programy pro mě nejsou dostupné, nebo nejsou dobře použitelné
- sbírám opakovaně data stejného charakteru a rád bych si jejich (před)zpracování automatizoval

Proč bych se právě já měl učit s R

- publikuji v odborných časopisech a rád bych do statě článku s Methodology and Statistical Analysis pravdivě psal, že „... all statistical analyses were performed using R language for statistical computing and graphics...“, protože existují důkazy, že citování R či jiných volných statistických nástrojů mnohdy zvyšuje pravděpodobnost citování takového článku
- tuším, že věda 2.0 v biomedicině se bude provozovat nejen formou experimentů na živém (in vivo) či v laboratořích (in vitro), ale budou ji tvořit ze značné části počítačové modely a simulace (in silico)
- uvědomuji si, že MS Excel v základním rozhraní neumí dotted vykreslit krabicový diagram

Proč bych se právě já měl učit s R

- zpracování dat v tabulkových procesorech a spoléhání se jen na ně je spjato s různými problémy, chybné výstupy z tabulkových procesorů dokonce vyvolaly některé vědecké skandály
- data již nějakou dobu (sám) analyzuji a přemýšlím, který nástroj pro analýzu (s kvalitní dokumentací a živou podporou a komunitou) se začít učit

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*

Stručná historie R

- R vychází z jazyka a prostředí S, které vyvinuto v Bellových laboratořích v letech 1975-1976 (prof. Johne Chambers)
- přerod v R v roce 1992 na Acklandské univerzitě na Novém Zélandu (prof. Ross Ihaka a Robert Gentleman)
- R je tedy akcentem S
- v roce 1994 první verze prostředí R pro volné použití, poté postupně vzniká řada dalších verzí, v září 2017 poslední verze R 3.4.3 („Short Summer“)
- za otce moderního R považován Hadley Wickham (leader analyst v RStudios, adj. prof. na Aucklandské univerzitě)

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 RStudio

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

<https://www.rstudio.com/>

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

- poté instalujme dle instrukcí do předvolené složky

Další software

- může (a bude) se časem hodit i
 - obecný textový editor, např. Notepad++

<https://notepad-plus-plus.org/>

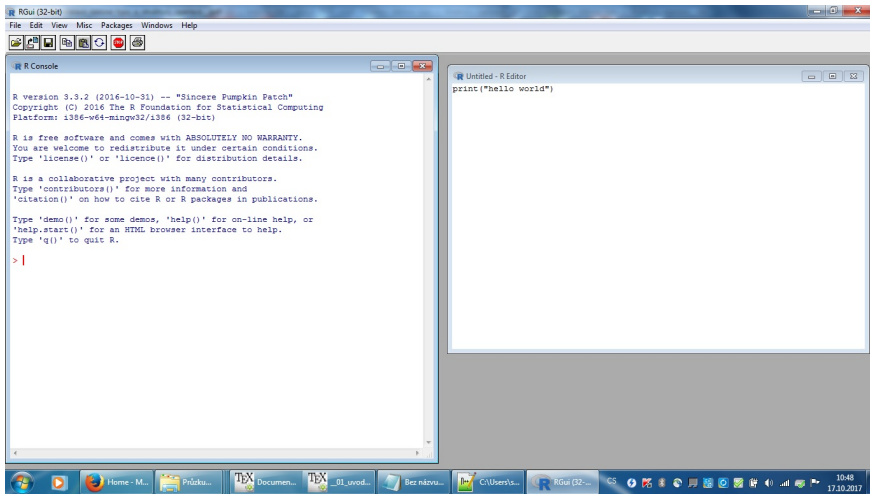
- univerzální konvertor dokumentů Pandoc

<https://pandoc.org/>

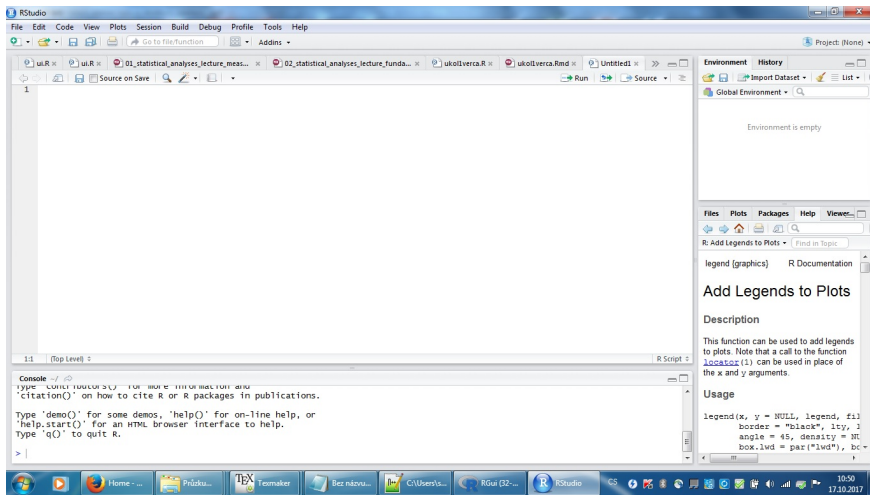
- typografický sázeč $\text{T}_{\text{E}}\text{X}$

<https://miktex.org/>

První spuštění R jádra



První spuštění RStudio



Ahoj světe!

- do skriptu či konzole napíšme

```
|| 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

Přístup k práci v R

- ❶ interaktivní práce mezi skriptem a konzolí (*shell programming*)
- ❷ (dávkové) volání skriptů
 - (i) vytvoření funkčního skriptu s příponou .R
 - (ii) uložení skriptu do pracovní složky, kterou zjistíme pomocí příkazu `getwd()`
 - (iii) zavolání a exekuce skriptu v konzoli pomocí příkazu `source(nazev_skriptu.R)`

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)

Komplexní číslo

- v R jako `complex`
- libovolné $x \in \mathbb{C}$ takové, že $x = a + bi$, kde $a, b \in \mathbb{R}$ a $i^2 = -1$
- např.

```
1 + 2i; 0 + 1i
```

- zda je hodnota typu `complex`, zjistíme pomocí

```
is.complex(1 + 2i)      # TRUE
class(0 + 1i)           # "complex"
class(sqrt(-1 + 0i))    # "complex"
class(sqrt(-1))         # Warning message:
                        # NaNs produced
```

Logická hodnota

- v R jako `logical`
- libovolné booleovské $x \in \{\text{TRUE}, \text{FALSE}\}$
- 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 dvojíty 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 $\{\text{NaN}\} \subset \{\text{NA}\}$
- např.

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

Atributy každého objektu

- každý objekt (daného datového typu) má svou třídu a délku
- třída (class) charakterizuje datový typ
- délka (length) vrátí počet atomických podobjektů objektu daného datového typu
- např.

```
class("ahoj")      # "character"
class(NaN)         # "numeric"
class(NA)          # "logical"
class(class(NA))   # "character"
length("123")      # 1
length(123)        # 1
length(NaN)        # 1
length(NA)         # 1
```

Přiřazení hodnoty k proměnné

- přiřadit hodnotu nějaké proměnné lze pomocí jednoduchého rovnítka

```
x = 5
```

- nebo pomocí orientované šipky

```
x <- 5  
5 -> x      # totéž
```

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

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

to se hodí zejména u dynamického iterování (viz později)

Intermezzo

- zkusme apriorně (bez ověření v R) vyslovit, o jaké datové typy jde v následujících případech

```

1.8
is.logical(is.numeric(-5000))
sqrt(4)           # sqrt() je druhá odmocnina
sqrt(4L)
TRUE
"FALSE"
asin(2)           # asin() je arcus sinus
1 / Inf
-2 / INF
class(TRUE)
class(class(is.complex(1 + 1i)))
"357L"
as.integer("357L")
as.integer("357")
length(12)

```


Datové struktury

- vektor (`vector`)
- faktor (`factor`)
- matice (`matrix`)
- pole (`array`)
- 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

```
x <- c(3, 1, 2)
y <- c("a", "b", "c")
names(x) <- y      # pojmenuje prvky
                   # vektoru x

x
unnname(x)         # zbaví prvky vektoru
                   # x jeho jejich jmen

setNames(x, y)      # opět pojmenuje
                   # prvky vektoru x
```

Subvektory, indexování, adresace

- vektor obsahující celočíselnou aritmetickou řadu lze s výhodou vytvořit následovně

```
x <- 1:10      # vektor o prvcích 1, 2, ..., 10
y <- 5:1       # vektor o prvcích 5, 4, ..., 1
z <- seq(from = 2, to = 10, by = 2)
              # vektor o prvcích 2, 4, 6, 8, 10
w <- seq(2, 10, 2)
              # totéž
```

toho lze využít při indexování

- R indexuje vektory od 1, nikoliv od 0 (první prvek má index 1, druhý index 2, apod.)

Subvektory, indexování, adresace

- 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")
```

Intermezzo

- vypišme z vektoru `x` každou třetí a pátou hodnotu

```
x <- c(34, 65, 4, 0, 56, 23, 54, 17,  
      4, 8, 5, 44, 84, -5, 4444, 49,  
      37, 86, 45, 65, 36, 72, 54, 36,  
      56, 74, 26, 88, 36, 76, 46,  
      17, 84, 57, 25, -75, 634, 5578,  
      -6, 46, 44, 743, 577, 466,  
      645, 33, 64, 67)
```


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

- lze je použít jako operátory mezi dvěma (či více) hodnotami, ale i mezi dvěma (či více) vektory shodných délek
- lze užít tzv. *prefix notaci* ve tvaru ‘operátor’(..., ...) vhodnou pro dynamické iterování
- mezi aritmetické operátory patří

operace	operátor	prefix notace	příklad
sčítání	+	‘+’(...)	2 + 3; ‘+’(2, 3)
odčítání	-	‘-’(...)	2 - 3; ‘-’(2, 3)
násobení	*	‘*’(...)	2 * 3; ‘*’(2, 3)
dělení	/	‘/’(...)	2 / 3; ‘/’(2, 3)
mocnění	^ či **	‘^’(...)	2 ^ 3; ‘^’(2, 3); či 2 ** 3
modulo ¹	%%	‘%%’(...)	7 %% 3; ‘%%’(7, 3)
celočíselné dělení	%%/	‘%%/’(...)	7 %%/ 3; ‘%%/’(7, 3)

Aritmetické operace

• sčítání

```

2 + 3                                # 5
15 + 25 + 35                         # 75
c(1, 2) + c(10, 20)                  # c(11, 22)
'+'(2, 3)                             # 5
'+'(15, 25, 35)                       # Error: operator needs
                                      # one or two arguments
'+'('+'(15, 25), 35)                  # 75
'+'(c(1, 2), c(10, 20))              # c(11, 22)

```

Aritmetické operace

• odčítání

```

12 - 3                      # 9
35 - 25 - 15                # -5
c(12, 25) - c(3, 6)         # c(9, 19)
'-'(12, 3)                  # 9
'-'(35, 25, 15)             # Error: operator needs
                             # one or two arguments
'-'('-'(35, 25), 15)        # -5
'-'(c(12, 25), c(3, 6))    # c(9, 19)

```

Aritmetické operace

• násobení

```

12 * 3                                # 36
35 * 25 * 15                          # 13125
c(12, 25) * c(3, 6)                   # c(36, 150)
'*'(12, 3)                            # 36
'*'(35, 25, 15)                       # Error: operator needs
                                       # one or two arguments
'*'('*'(35, 25), 15)                   # 13125
'*'(c(12, 25), c(3, 6))               # c(36, 150)

```

Aritmetické operace

• dělení

```

12 / 3 # 4
45 / 5 / 3 # 3
c(12, 25) / c(3, 5) # c(4, 5)
'/'(12, 3) # 4
'/'(45, 5, 3) # Error: operator needs
               # one or two arguments
'/'('/'(45, 5), 3) # 3
'/'(c(12, 25), c(3, 5)) # c(4, 5)

```

Aritmetické operace

• mocnění

```

2 ^ 3                                # 8
2 ** 3                              # 8; Python-like notace
4 ^ 3 ** 2                          # 262144
4 ^ (3 ** 2)                       # 262144
(4 ^ 3) ** 2                        # 4096; pozor na
                                   # uzávorkování !!!
c(25, 36) ^ 0.5                     # c(5, 6); odmocňování
c(5, 3) ^ c(2, 3)                   # c(25, 27)
c(5, 3) ** c(2, 3)                  # c(25, 27)
'^'(2, 3)                           # 8
'^^(2, 3)                           # Error: could not find
                                   # function "^^"
'^'(4, 3, 2)                        # Error: operator needs
                                   # one or two arguments
'^'('^'(4, 3), 2)                    # 4096
'^'(c(5, 3), c(2, 3))               # c(25, 27)

```


Aritmetické operace

- modulo (zbytek po celočíselném dělení)
 - operace $m \% n$ vrací takové k , aby bylo $\frac{|n|}{n} \cdot k \in \{0, 1, \dots, n-1\}$ a současně $\frac{m-k}{n} \in \mathbb{Z}$, kde $m \in \mathbb{Z}$ a $n \in \mathbb{Z} \setminus \{0\}$ jsou dané parametry
 - formálně též $m \equiv k \pmod{n}$
 - čteme „ m je kongruentní s k modulo n “

```

12 %% 3           # 0
10 %% 3           # 1
10 %% -3          # -2
5  %% 0           # NaN; n nesmí
                  # být nula !
17 %% 23          # 17

```

Aritmetické operace

• celočíselné dělení

- operace $m \% n$ vrací takové z , aby platilo $m = zn + k$ a současně $\frac{|n|}{n} \cdot k \in \{0, 1, \dots, n - 1\}$, kde $m \in \mathbb{Z}$, $n \in \mathbb{Z} \setminus \{0\}$ jsou dané parametry
- víme-li tedy, že pro nějaká $m \in \mathbb{Z}$ a $n \in \mathbb{Z} \setminus \{0\}$ je $m \equiv k \pmod{n}$, pak výsledkem celočíselného dělení $m \% n$ bude $\frac{m-k}{n}$

```

12 %/% 3           # 4
10 %/% 3           # 3
10 %/% -3          # -2
5 %/% 0            # Inf
17 %/% 23          # 0
23 %/% 17          # 1
17 %/% 5           # 3; celočíselné
                   # dělení
(17 - 17 %/% 5) / 5 # 3; explicitní
                   # výpočet
(17 %/% 5) * 5 + (17 %/% 5) # 17

```

Aritmetické operace

- uživatelem definované operátory
 - soubor vestavěných aritmetických operátorů lze rozšířit o vlastní, uživatelem definované operátory (více později u uživatelem-definovaných funkcí)

```
# definuji vlastní operátor
'očaruj_pomocí%' <- function(x, y){x^2 + y}

5 %očaruj_pomocí% 4          # 5 ^ 2 + 4 = 29
6 %očaruj_pomocí% -3        # 6 ^ 2 - 3 = 33

# definuji operátor, který vrací TRUE, právě když
# je první číslo celočíselným dělitelem druhého
'%d%' <- function(d, n){n %% d == 0}

3 %d% 9          # TRUE
4 %d% 9          # FALSE
'%d%'(7, 21)    # TRUE; prefix notation
```

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

```
c(FALSE, FALSE, TRUE, TRUE) &  
c(FALSE, TRUE, FALSE, TRUE)  
# c(FALSE, FALSE, FALSE, TRUE)
```

- operace *long* AND (operátor `&&`, „AND-AND“)
 - použitelná také pro vektory
 - vyhodnocuje pouze nutný počet výroků (možná optimalizace) a vrací jen jednu hodnotu

```
c(FALSE, FALSE, TRUE, TRUE) &&  
c(FALSE, TRUE, FALSE, TRUE)  
# FALSE
```

Logické operace

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

```
c(FALSE, FALSE, TRUE, TRUE) |  
c(FALSE, TRUE, FALSE, TRUE)  
# c(FALSE, TRUE, TRUE, TRUE)
```

- operace *long* OR (operátor `||`, „OR-OR“)
 - použitelná také pro vektory, vyhodnocuje pouze nutný počet výroků (možná optimalizace) a vrací jen jednu hodnotu

```
c(FALSE, FALSE, TRUE, TRUE) ||  
c(FALSE, TRUE, FALSE, TRUE)  
# FALSE, protože první dvojice vrací FALSE
```

- operace XOR (funkce `xor()`, „vylučovací OR“)

```
xor(c(FALSE, FALSE, TRUE, TRUE),  
c(FALSE, TRUE, FALSE, TRUE))  
# c(FALSE, TRUE, TRUE, FALSE)
```

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()`)

```
2 == 3                                # FALSE
'==' ('a', 'a')                       # TRUE; prefix notace
all.equal(c(1, 2), c(1, 2 + 1e-13),
          tolerance = 1e-12)          # TRUE; porovnává vektory
                                      # volitelnou danou tolerancí
identical(c(1, 2), c(1, 2 + 1e-13))  # FALSE, porovnává objekty
                                      # a vrací TRUE jen při úplné
                                      # 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
'>'(12, 11) # TRUE; prefix notace
FALSE >= TRUE # FALSE; porovnává hodnotu
# v booleovské aritmetice
# (TRUE := 1, FALSE := 0)
```

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

```
2 != 3 # TRUE
TRUE != FALSE # TRUE
'!='(FALSE, FALSE) # FALSE; prefix notace
```


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
```

Množinové operace

- sjednocení množin typu $A \cup B$

```
|| union(c(1, 2, 3), c(5, 1)) # c(1, 2, 3, 5)
```

- průnik množin typu $A \cap B$

```
|| intersect(c(1, 2, 3), c(5, 1)) # c(1)
```

- asymetrický rozdíl množin typu $A - B$

```
|| setdiff(c(1, 2, 3), c(5, 1)) # c(2, 3)
```

- symetrický rozdíl množin typu $A \div B = (A - B) \cup (B - A)$

```
|| union(setdiff(c(1, 2, 3), c(5, 1)),
||       setdiff(c(5, 1), c(1, 2, 3)))
|| # c(2, 3, 5)
```

- jsou si množiny rovny, tedy $A \subseteq B \wedge B \subseteq A \iff A = B$?

```
|| setequal(c(1, 2, 3), c(5, 1)) # FALSE
|| setequal(c(1, 2, 3), c(3, 2, 1)) # 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"
```

Konstanty

- některé matematické a textové konstanty jsou součástí jádra R
- např.

```
pi                # 3.141593; Ludolfovo číslo
exp(1)           # 2.718282; Eulerovo číslo
Inf              # nekonečno
-Inf             # mínus nekonečno
letters          # malá písmena anglické abecedy
LETTERS          # velká písmena anglické abecedy
month.name       # názvy měsíců
month.abb        # zkratky názvů měsíců
weekdays(Sys.Date())
                 # vrátí aktuální den v týdnu
weekdays(seq(as.Date("2017-10-23"),
               as.Date("2017-10-29"), 1))
                 # vypíše názvy dnů v týdnu
```

- fyzikální konstanty jsou dostupné v balíčku constants

Děkuji za pozornost!

lubomir.stepanek@lf1.cuni.cz

lubomir.stepanek@fbmi.cvut.cz

github.com/LStepanek/17VSADR_Skriptovani_a_analyza_dat_v_jazyce_R