

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

—  
B03128 – Úvod do skriptovacího jazyka R

Lubomír Štěpánek<sup>1, 2</sup>



<sup>1</sup>Oddělení biomedicínské statistiky  
Ústav biofyziky a informatiky  
1. lékařská fakulta  
Univerzita Karlova v Praze



<sup>2</sup>Katedra biomedicínské informatiky  
Fakulta biomedicínského inženýrství  
České vysoké učení technické v Praze

(2020) 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 (3 kreditové body)
- zakončen seminární prací
- rozsah 0 + 30 (vyučovacích hodin), tj. 2 vyučovací hodiny týdně
- neoficiální účast na bázi dobrovolnosti vítána

# Zápočet

- zápočet bude udělen za
  - (i) alespoň rozumnou účast na cvičeních
  - (ii) odevzdání alespoň rozumného množství domácích úkolů (bude zadáno nejspíš 4–8 úkolů)
  - (iii) vypracování seminární práce
    - 2–5 stránek
    - založena na výběru vhodných dat (klidně vlastních) a jejich analýze v R



# 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

- chystám se aktivně účastnit studentské vědecké konference 1. LF UK
- 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 doted' 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. John 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/>

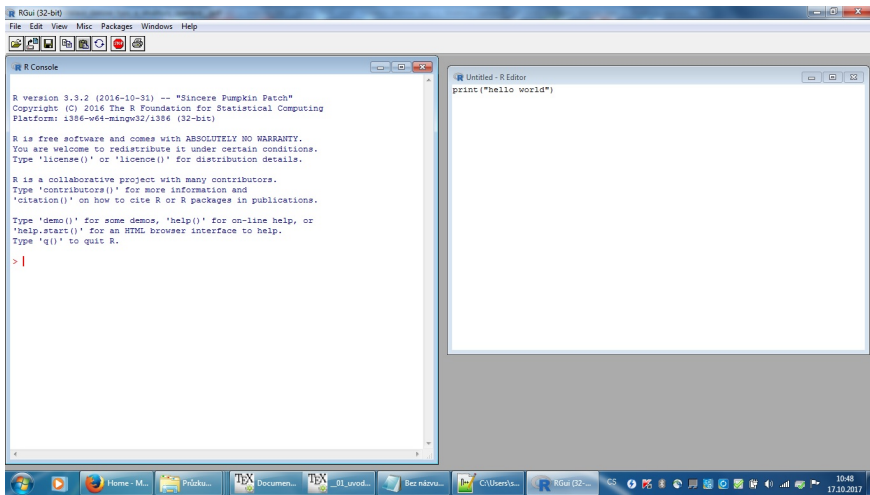
- univerzální konvertor dokumentů Pandoc

<https://pandoc.org/>

- typografický sázeč T<sub>E</sub>X

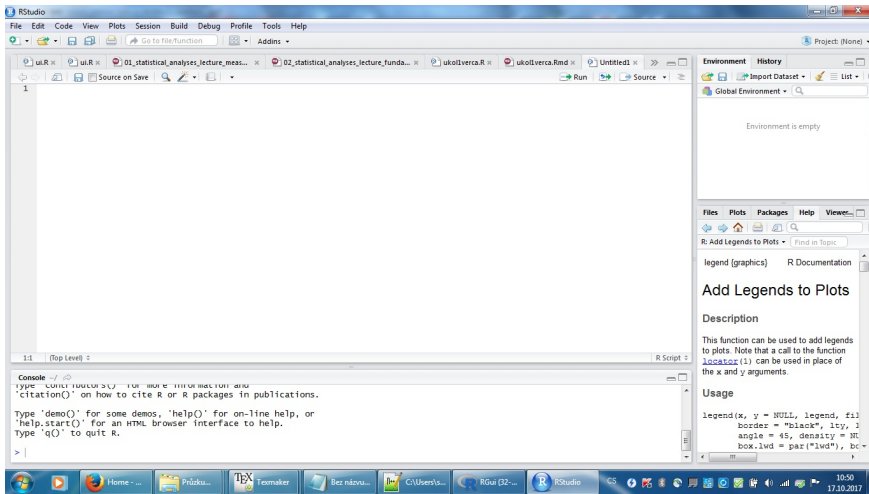
<https://miktex.org/>

# První spuštění R jádra





# První spuštění RStudio



# Ahoj světe!

- do skriptu či konzole napíšme

```
1 || print("hello world")
```

- dostaneme

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

# Práce s nápovědou

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

```
1 || help(print)
```

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

```
1 || ?print
```

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

```
1 || ??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)`



---

- ```
install.packages(  
  "nazev_balicku",  
  dependencies = TRUE,  
  repos = "http://cran.us.r-project.org"  
)
```
- poslední argument repos není nutný, ale je šikovný, vybere doménu, z které balíček stáhnout, aniž by ji musel uživatel vybrat kliknutím v pop-up okně (vhodné pro úplnou automatizaci kódu)
  - zkusme nainstalovat balíčky ShinyItemAnalysis a swirl!

# Datové typy

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





## Celé číslo

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

$$1 \parallel 5L; 13L, -5L$$

- zda je hodnota typu integer, zjistíme pomocí

```
1 is.integer(-13L)      # TRUE
2 class(-13L)           # "integer"
3 is.integer(-13)       # FALSE
4 class(-13)            # "numeric"
```

- přetypování celého čísla na reálné (numeric) pomocí

```
1 || 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)

# 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 || 1 + 2i; 0 + 1i
```

- zda je hodnota typu `complex`, zjistíme pomocí

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

# Logická hodnota

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

```
1 || TRUE; FALSE; T; F
```

- zda je hodnota typu `logical`, zjistíme pomocí

```
1 || is.logical(TRUE)      # TRUE
2 || class(FALSE)         # "logical"
3 || class("TRUE")        # "character"
4 || class(T)             # "logical"
5 || class(F)             # "logical"
```

# Textový řetězec

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

```
1 || "ahoj"; 'xweiwogw23425ng'; ""
```

- zda je hodnota typu character, zjistíme pomocí

```
1 || is.character("ahoj")      # TRUE
2 || class("bla bla")         # "character"
3 || class("123")             # "character"
4 || class(123)                # "numeric"
5 || is.numeric(Inf)           # TRUE
6 || is.numeric("Inf")         # FALSE
```

- na textový řetězec lze převést libovolnou jinou hodnotu pomocí

```
1 || 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  $\{\text{NaN}\} \subseteq \{\text{NA}\}$
- 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

```

# Atributy každého objektu

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

```
1 class("ahoj")      # "character"
2 class(NaN)         # "numeric"
3 class(NA)          # "logical"
4 class(class(NA))   # "character"
5 length("123")      # 1
6 length(123)        # 1
7 length(NaN)        # 1
8 length(NA)         # 1
```

# 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)

# Intermezzo

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

```
1      1.8
2      is.logical(is.numeric(-5000))
3      sqrt(4)                # sqrt() je druhá odmocnina
4      sqrt(4L)
5      TRUE
6      "FALSE"
7      asin(2)                # asin() je arcus sinus
8      1 / Inf
9      -2 / INF
10     class(TRUE)
11     class(class(is.complex(1 + 1i)))
12     "357L"
13     as.integer("357L")
14     as.integer("357")
15     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.

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

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

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

- vektor tedy lze prodloužit libovolně o jednu či více hodnot

16



# Subvektory, indexování, adresace

- vektor obsahující celočíselnou aritmetickou řadu lze s výhodou vytvořit následovně

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

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

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

# Intermezzo

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

```
1 | x <- c(34, 65, 4, 0, 56, 23, 54, 17,  
2 |       4, 8, 5, 44, 84, -5, 4444, 49,  
3 |       37, 86, 45, 65, 36, 72, 54, 36,  
4 |       56, 74, 26, 88, 36, 76, 46,  
5 |       17, 84, 57, 25, -75, 634, 5578,  
6 |       -6, 46, 44, 743, 577, 466,  
7 |       645, 33, 64, 67)
```



# Intermezzo

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

```
1 | x <- c(34, 65, 4, 0, 56, 23, 54, 17,
2 |       4, 8, 5, 44, 84, -5, 4444, 49,
3 |       37, 86, 45, 65, 36, 72, 54, 36,
4 |       56, 74, 26, 88, 36, 76, 46,
5 |       17, 84, 57, 25, -75, 634, 5578,
6 |       -6, 46, 44, 743, 577, 466,
7 |       645, 33, 64, 67)
```

- řešením může být

```
1 | x[c(F, F, T, F, T)]
2 |
3 | # 4, 56, 17, 8, 84, 4444, 86, 65, 54,
4 | # 56, 88, 76, 84, 25, 5578, 46, 577,
5 | # 645, 67
```

# Faktory

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

```

1  x <- factor(
2      c("muž", "žena", "muž", "muž")
3  )      # pořadí kategorií je defaultně
4          # abecední
5  x <- factor(
6      c("muž", "žena", "muž", "muž"),
7      levels = c("žena", "muž")
8  )      # zde si pořadí kategorií
9          # určíme sami
    
```

- nad faktory snadno vytvoříme kontingenční tabulku

```

1  table(x)      # x
2                  # žena muž
3                  # 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 <sup>1</sup>	%%	‘%%’(...)	7 %% 3; ‘%%’(7, 3)
celočíselné dělení	%%/	‘%%/’(...)	7 %%/ 3; ‘%%/’(7, 3)

# Aritmetické operace

## • sčítání

```

1      2 + 3                                # 5
2      15 + 25 + 35                         # 75
3      c(1, 2) + c(10, 20)                  # c(11, 22)
4      '+'(2, 3)                             # 5
5      '+'(15, 25, 35)                       # Error: operator needs
6                                              # one or two arguments
7      '+'(+'(15, 25), 35)                   # 75
8      '+'(c(1, 2), c(10, 20))              # c(11, 22)

```

# Aritmetické operace

## • odčítání

```
1 12 - 3 # 9
2 35 - 25 - 15 # -5
3 c(12, 25) - c(3, 6) # c(9, 19)
4 '-'(12, 3) # 9
5 '-'(35, 25, 15) # Error: operator needs
6 # one or two arguments
7 '-'('-(35, 25), 15) # -5
8 '-'(c(12, 25), c(3, 6)) # c(9, 19)
```

# Aritmetické operace

- násobení

```
1 12 * 3 # 36
2 35 * 25 * 15 # 13125
3 c(12, 25) * c(3, 6) # c(36, 150)
4 '*'(12, 3) # 36
5 '*'(35, 25, 15) # Error: operator needs
6 # one or two arguments
7 '*'('*(35, 25), 15) # 13125
8 '*'(c(12, 25), c(3, 6)) # c(36, 150)
```

# Aritmetické operace

## • dělení

```

1 12 / 3 # 4
2 45 / 5 / 3 # 3
3 c(12, 25) / c(3, 5) # c(4, 5)
4 '/'(12, 3) # 4
5 '/'(45, 5, 3) # Error: operator needs
6 # one or two arguments
7 '/'('/'(45, 5), 3) # 3
8 '/'(c(12, 25), c(3, 5)) # c(4, 5)

```

# Aritmetické operace

## • mocnění

```

1      2 ^ 3                      # 8
2      2 ** 3                     # 8; Python-like notace
3      4 ^ 3 ** 2                 # 262144
4      4 ^ (3 ** 2)              # 262144
5      (4 ^ 3) ** 2              # 4096; pozor na
6                                  # uzávorkování !!!
7      c(25, 36) ^ 0.5            # c(5, 6); odmocňování
8      c(5, 3) ^ c(2, 3)         # c(25, 27)
9      c(5, 3) ** c(2, 3)        # c(25, 27)
10     '^'(2, 3)                 # 8
11     '**'(2, 3)                # Error: could not find
12                               # function "*"
13     '^'(4, 3, 2)              # Error: operator needs
14                               # one or two arguments
15     '^'('^'(4, 3), 2)         # 4096
16     '^'(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  $n$ “

1	12	%%	3	# 0
2	10	%%	3	# 1
3	10	%%	-3	# -2
4	5	%%	0	# NaN; <i>n nesmí</i>
5				# <i>být nula !</i>
6	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  $\frac{m-k}{n}$

```

1      12 %/% 3                # 4
2      10 %/% 3                # 3
3      10 %/% -3               # -2
4      5 %/% 0                 # Inf
5      17 %/% 23               # 0
6      23 %/% 17               # 1
7      17 %/% 5                # 3; celočíselné
8                                   # dělení
9      (17 - 17 %% 5) / 5      # 3; explicitní
10                                   # výpočet
11     (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)

```

1      # definuji vlastní operátor
2      '%očaruj_pomocí%' <- function(x, y){x^2 + y}
3
4      5 %očaruj_pomocí% 4           # 5 ^ 2 + 4 = 29
5      6 %očaruj_pomocí% -3          # 6 ^ 2 - 3 = 33
6
7      # definuji operátor, který vrací TRUE, právě když
8      # je první číslo celočíselným dělitelem druhého
9      '%d%' <- function(d, n){n %% d == 0}
10
11     3 %d% 9                        # TRUE
12     4 %d% 9                        # FALSE
13     '%d%' (7, 21)                 # TRUE; prefix notace

```

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

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



# Logické operace

- operace NOT (operátor !)

- vrací výroku opačnou logickou hodnotu, než které výrok nabývá

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

- funkce all()

- vrací pro vektor výroků TRUE právě tehdy, jsou-li všechny výroky rovné TRUE

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

```
1 || any(c(3 < 2, 7 %% 3 <= 0, FALSE)) # FALSE
2 || 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      '=='('a', 'a')             # TRUE; prefix notace
3      all.equal(c(1, 2), c(1, 2 + 1e-13),
4                      tolerance = 1e-12)
5                      # TRUE; porovnává vektory
6                      # volitelnou danou tolerancí
7      identical(c(1, 2), c(1, 2 + 1e-13))
8                      # FALSE, porovnává objekty
9                      # a vrací TRUE jen při úplné
10                     # shodě
```

# Operace porovnávání (komparace)

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

```
1 | 2 < 3 # TRUE
2 | "b" <= "a" # FALSE; porovnává pořadí
3 | # v abecedě
4 | '>'(12, 11) # TRUE; prefix notace
5 | FALSE >= TRUE # FALSE; porovnává hodnotu
6 | # v booleovské aritmetice
7 | # (TRUE := 1, FALSE := 0)
```

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

```
1 | 2 != 3 # TRUE
2 | TRUE != FALSE # TRUE
3 | '!='(FALSE, FALSE) # FALSE; prefix notace
```



- ```
1 || isTRUE(3^2 > 2^3) # TRUE
```



## Vestavěné matematické funkce

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

```
1      abs(), sign()
2      acos(), asin(), atan()
3      sin(), cos(), tan()
4      ceiling(), floor(), trunc()
5      exp(), log(), log10(), log2(), sqrt()
6      max(), min(), prod(), sum()
7      cummax(), cummin(), cumprod(), cumsum(),
      diff()
8      pmax(), pmin()
9      range()
10     mean(), median(), cor(), sd(), var()
11     rle()
```

# Zaokrouhlování, formátování čísel

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

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

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

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

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

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

# Konstanty

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

```
1      pi                # 3.141593; Ludolfovo číslo
2      exp(1)           # 2.718282; Eulerovo číslo
3      Inf              # nekonečno
4      -Inf            # mínus nekonečno
5      letters          # malá písmena anglické abecedy
6      LETTERS          # velká písmena anglické abecedy
7      month.name       # názvy měsíců
8      month.abb        # zkratky názvů měsíců
9      weekdays(Sys.Date())
10                                # vrátí aktuální den v týdnu
11      weekdays(seq(as.Date("2018-10-01"),
12                  as.Date("2018-10-07"), 1))
13                                # 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

[https://github.com/LStepanek/B03128\\_Uvod\\_do\\_skriptovaciho\\_jazyka\\_R](https://github.com/LStepanek/B03128_Uvod_do_skriptovaciho_jazyka_R)