

# Přehled datových typů a struktur, jednoduché operace

—  
Úvod do jazyka R

Lubomír Štěpánek<sup>1, 2</sup>



<sup>1</sup>Oddělení biomedicínské statistiky & výpočetní techniky  
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

(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 Datové typy, jejich vlastnosti
- 2 Přehled datových struktur
- 3 Vektory a operace s nimi
- 4 Literatura

# 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.

```
1 ||      5; -13.8, 4.5578e15
```

- zda je hodnota typu `numeric`, zjistíme pomocí

```
1 ||      is.numeric(-13.8)      # TRUE
2 ||      class(-13.8)           # "numeric"
3 ||      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.

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

- 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  $\{\text{NaN}\} \subset \{\text{NA}\}$
- 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
```

# Tvorba vektorů a základní příkazy

- vektor tedy lze prodloužit libovolně o jednu či více hodnot

```
1      x <- c(3, 1, 2)
2      length(x)           # 3
3      y <- 1
4      z <- c(2)
5      w <- c(5, 7)
6      x <- c(x, y)         # prodloužení vektoru x
7                           # o hodnotu y
8      w <- c(w, z)         # prodloužení vektoru w
9                           # o vektor z
10                          # jednoprvkový vektor je
11                          # skalárem, jednou hodnotou
12      c <- c(1, 2, 3)
13      c                   # vektor o prvcích 1, 2, 3
14                          # byť je c referovaný termín,
15                          # funkce c je zachována
16                          # 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 |
6 | x
7 | unname(x)          # zbaví prvky vektoru
8 |                   # x jeho jejich jmen
9 | setNames(x, y)      # opět pojmenuje
                       # prvky vektoru x
```

# 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)
```

# Logické vektory

- používají se (nejen) k adresování vhodných prvků

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

- výhodný je někdy tzv. *recycling*

```
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)
celočíslné 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 *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)
```

- operace *long* OR (operátor `||`, „OR-OR“)

- 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, protože první dvojice vrací FALSE
```

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

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

# Operace porovnávání (komparace)

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

```

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

```

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

```

1 | is.element(c(2, 6), c(1:5))
2 |                               # c(TRUE, FALSE)
3 | is.element(c(1:5), c(2, 6))
4 |                               # c(FALSE, TRUE,
5 |                               # FALSE, FALSE, FALSE)

```

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

```

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

```

# Množinové operace

- sjednocení množin typu  $A \cup B$

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

- průnik množin typu  $A \cap B$

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

- asymetrický rozdíl množin typu  $A - B$

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

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

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

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

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

```
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())     # vrátí aktuální den v týdnu
10                                     #
11      weekdays(seq(as.Date("2017-10-23"),
12                  as.Date("2017-10-29"), 1))
13                                     # vypíše názvy dnů v týdnu
```

- fyzikální konstanty jsou dostupné v balíčku `constants`

# Literatura



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



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

Děkuji za pozornost!

lubomir.stepanek@lf1.cuni.cz

lubomir.stepanek@fbmi.cvut.cz