POČÍTAČOVÉ PRAKTIKUM (PROGRAM R)

navazující magisterské studium FP

1. ročník, zimní semestr

Tereza Šimková, KAP FP TUL

(naposledy upraveno 25. listopadu 2020)

Požadavky na udělení zápočtu:

Vypracování a odevzdání semestrální práce do stanoveného termínu.

Materiály:

Materiály a informace ke cvičení jsou k dispozici na E-learningovém portálu TUL v kurzu KAP/PPR (2020)

`https://elearning.tul.cz/`

Konzultace:

Prostřednictvím e-mailu či Google Meet.

R software

- volně dostupný programovací jazyk
- vytvořen R. Ihakaem a R. Gentlemanem (University of Auckland, NZ)
- prostředí pro statistické výpočty a grafické výstupy
- ke stažení na stránkách `http://www.r-project.org`
- lze snadno rozšiřovat pomocí tzv. packages (balíků) `https://cran.r-project.org/`
- RStudio poskytuje uživatelsky příjemnější prostředí (ke stažení na stránkách `http://www.rstudio.com`)

Základní ovládání

- po spuštění jsou otevřena zpravidla 3 okna
- do konzolového okna se vkládají objekty a výpočty, spouštějí se funkce, zobrazují se numerické výstupy (tzv. Console)
- v grafickém okně se zobrazují grafy (lze přepnout na aktuální adresář, seznam balíků nebo nápovědu)
- v dalším okně jsou zobrazeny všechny aktivní objekty (lze přepnout na seznam všech zadaných příkazů do konzole)
- další okno (*File* $\rightarrow$ *New File* $\rightarrow$ *R Script*) slouží k tvorbě vlastních funkcí

Základní ovládání

- znak > oznamuje, že je konzole připravena k zápisu
- R lze používat k numerickým výpočtům jako kalkulačku – zapíšeme výraz a stiskneme ENTER
- vstupní řádky modře

  ```
  > 5+3
  ```

- klávesy $\uparrow, \downarrow$ se používají k listování v již zadaných příkazech
- klávesy $\leftarrow, \rightarrow$ se používají k pohybu karetu

Základní ovládání

- znak # se používá pro komentář (vše za znakem # se v daném řádku ignoruje)

```
> 3+5 # soucet dvou cisel
[1] 8
```

- příkazem <- či = je do objektu přiřazena hodnota/výraz

```
> x <- 5
> x = 5
```

- hodnotu uloženou do objektu lze zjistit zapsáním názvu objektu do konzole

```
> x
[1] 5
```

nebo zapsáním celého příkazu do kulatých závorek při zapisování do konzole

```
> (x <- 5)
[1] 5
```

Základní ovládání

- názvy objektů se mohou skládat z písmen, číslic, tečky a podtržítka
- jsou rozlišována velká a malá písmena

```
> x
[1] 5
> X
Error: object 'X' not found
```

- používá se desetinná tečka, nikoliv čárka

```
> y <- 1,25
Error: unexpected ',' in "y <- 1,"
> y <- 1.25
```

- objeví-li se na dalším řádku znak +, většinou není zápis dokončen

```
> x <- (3-1.5
+
```

Základní ovládání

- matematické operátory a funkce uvedené v tabulce aplikované na vektor či matici jsou prováděny po prvcích

MATEMATICKÉ OPERÁTORY	
+	součet
-	rozdíl
*	součin
/	podíl
^ nebo **	mocnina
%%	modulo
%/%	celočíselné dělení

Základní ovládání

ZÁKLADNÍ FUNKCE	
`abs(·)`	absolutní hodnota
`sqrt(·)`	druhá odmocnina
`log(·)`	přirozený logaritmus
`log10(·)`	dekadický logaritmus
`exp(·)`	exponenciální funkce
`sin(·)`	sinus
`cos(·)`	kosinus
`tan(·)`	tangens
`ceiling(·)`	horní celá část
`floor(·)`	dolní celá část
`round(·, digits=n)`	zaokrouhlení na `n` desetinných míst

Základní ovládání

KONSTANTY	
`pi`	Ludolfovo číslo π
`i`	imaginární jednotka i

Základní ovládání

- příkaz `getwd()` zjistí cestu k aktuálnímu adresáři, do kterého právě ukládáme nebo z něj načítáme soubory

```
> getwd()
[1] "C:/Users/Tereza Šimková/Documents"
```

- příkaz `setwd()` změní adresář, kde v závorce je uvedena nová cesta v uvozovkách, např.

```
> setwd("C:/Users/Tereza Šimková/Documents/VÝUKA")
```

nebo lze cestu nastavit v okně vpravo dole na záložce *Files* → *More* → *Set As Working Directory*

Datové typy

1. numerický (numerical)
   ```
   > a <- -2.514
   ```
2. znakový (character)
   – skládá se z řetězců, zadává se do "" nebo ’’, pro textové hodnoty
   ```
   > b <- "univerzita"
   ```
3. logický (logical)
   – jen hodnoty TRUE (nebo T), FALSE (nebo F)
   ```
   > a > 0
   ```
4. komplexní (complex)
   – numerická hodnota doplněná o imaginární jednotku i $(i^2 = -1)$
   ```
   > d <- 6-2i
   ```

Datové typy

- `mode` zjistí typ objektu

```
> mode(b)
[1] "character"
```

- speciální hodnoty:
  `NA` (z angl. Not Available, chybějící nebo neznámá hodnota)
  `Inf` (z angl. Infinity, nekonečno)
  `NaN` (z angl. Not a Number, výsledek nedefinovaných numerických výpočtů)
  `NULL` (v objektu není nic uloženo)

Datové struktury

- umožňují více hodnot specifikovat jako jeden objekt
  1. vektory
  2. faktory
  3. matice
  4. pole
  5. datová tabulka
  6. seznam

Vektory

- `c()` vytvoří vektor (c od angl. combine), kde do závorky zapíšeme složky vektoru oddělené čárkami

```
> v <- c(-5, 1, 0.6, 17, 1124)
> v
[1] -5.0 1.0 0.6 17.0 1124.0
```

- lze tvořit numerické, logické, komplexní i textové vektory
- `length()` určí délku vektoru, který je uvedený v závorce

```
> length(v)
[1] 5
```

Vektory

– matematické operátory a funkce aplikované na vektory se provádějí po složkách, např.

```
> v/2
```
(tj. každá složka je vydělena 2)
```
[1] -2.5  0.5  0.3  8.5 562.0
> v^5
```
(tj. každá složka je umocněna na 5.)
```
[1] -3.125000e+03 1.000000e+00 7.776000e-02 1.419857e+06 1
> w <- c(1, 0.5, 2, 3, 12)
> w
[1] 1.0 0.5 2.0 3.0 12.0
> v/w
```
(tj. 1. složka vektoru v je vydělena 1. složkou vektoru w, 2. složka vektoru v je vydělena 2. složkou vektoru w, atd.)
```
[1] -5.000000 2.000000 0.300000 5.666667 93.666667
> v*w
```
(tj. 1. složka vektoru v je vynásobena 1. složkou vektoru w, 1. složka vektoru v je vynásobena 1. složkou vektoru w, atd.)
```
[1] -5.0  0.5  1.2 51.0 13488.0
```

Vektory

- pro skalární součin vektorů použijeme %*%

```
> v%*%w
[,1]
[1,] 13535.7
```

- a:b vytvoří aritmetickou posloupnost od a do b s krokem 1

```
> (z <- 1:6)
[1] 1 2 3 4 5 6
```

- seq(from=a, to=b, by=k) vytvoří aritmetickou posloupnost od a do b s krokem k

```
> seq(from=1,to=6,by=1)
[1] 1 2 3 4 5 6
> seq(from=2,to=10,by=2)
[1] 2 4 6 8 10
> seq(from=1,to=10,by=2.4)
[1] 1.0 3.4 5.8 8.2
> seq(from=-5,to=1,by=1)
[1] -5 -4 -3 -2 -1 0 1
```

Vektory

- `rep(o, n)` vytvoří vektor, ve kterém se zadaný objekt o opakuje `n`-krát

```
> rep(1.5,3)
[1] 1.5 1.5 1.5
```

- chceme-li vypsat i-tou složku vektoru, použijeme název vektoru a do hranaté závorky uvedeme danou pozici, tj. i (podobně pro skupinu složek)

```
> v <- c(-5, 1, 0.6, 17, 1124)
> v[3]          (je vypsána 3. složka vektoru v)
[1] 0.6
> v[2:4]          (je vypsána 2. až 4. složka vektoru v)
[1] 1 0.6 17
> v[c(1,4,5)]          (je vypsána 1., 4. a 5. složka vektoru v)
[1] -5 7 24
```

Vektory

– chceme-li vymazat i-tou složku vektoru, použijeme název vektoru a do hranaté závorky uvedeme danou pozici opatřenou znaménkem minus, tj. $-i$ (podobně pro skupinu složek)

```
> v <- v[-4]
[1] -5 1 0.6 1124
> v <- v[-c(1,2)]
[1] 1 0.6 1124.0
```

Faktory

- speciální případ vektorů pro práci s nominálními nebo ordinálními daty
- navíc obsahují informaci `Levels` (konečná množina hodnot, kterých proměnná může nabývat)

```
> factor(c("zena","zena","muz","zena","muz","zena"))
[1] zena zena muz zena muz zena
Levels: muz zena
```

- argument `labels` lze použít k definici popisků, pokud jsou hodnoty vyjádřeny číselně

```
> factor(c(1,1,0,1,0,1),labels=c("muz","zena"))
[1] zena zena muz zena muz zena
Levels: muz zena
```

- na faktory nelze aplikovat matematické operátory a elementární funkce

SPECIÁLNÍ FUNKCE	
`sum(·)`	součet prvků
`cumsum(·)`	kumulativní součet prvků
`prod(·)`	součin prvků
`cumprod(·)`	kumulativní součin prvků
`sort(·)`	uspořádání prvků do neklesající posloupnosti
`rank(·)`	pořadí prvků v neklesající posloupnosti
`which(·)`	vrátí pozici prvku, který vyhovuje dané podmínce
`min(·)`	minimální prvek
`max(·)`	maximální prvek

Matice

- matice je dvoudimenzionální struktura složená z řádků (rows) a sloupců (columns)
- `matrix(v, n, m)` uspořádá prvky vektoru `v` do matice o `n` řádcích a `m` sloupcích

```
> v <- c(2, 4, 0, -1, 3, 5, -5, 10, 0, 1, -4, 1)
> (M <- matrix(v,4,3))
```

- argument `byrow=TRUE` udává zaplňování matice po řádcích

```
> matrix(v,4,3,byrow=TRUE)
```

- `dim()` zjistí rozměry matice uvedeného v závorce (nefunguje pro vektor)

```
> dim(M)
[1] 4 3
```

- alternativní způsob zadání matice pomocí `dim()`

```
> v <- c(2, 4, 0, -1, 3, 5, -5, 10, 0, 1, -4, 1)
> dim(v) <- c(4,3)
> v
```

Matice

- řádky a sloupce matice lze pojmenovat pomocí `rownames` a `colnames` (nebo zjistit, jak jsou řádky či sloupce již pojmenované)

```
> rownames(M) <- c("Adam", "Bara", "Cyril", "Eva")
> colnames(M) <- c("utery", "streda", "ctvrtek")
```

- `M[i,j]` vypíše prvek matice `M` na pozici `i,j`

```
> M[2,3]
```

- `M[i, ]`, resp. `M[ ,j]` vypíše `i`-tý řádek, resp. `j`-tý sloupec matice `M`

```
> M[3,]
> M[,2]
```

- výpis prvků lze provádět přes názvy řádků a sloupců

```
> M["Bara", ]
> M[ , "streda"]
> M["Bara", "streda"]
```

Matice

- operátory a funkce aplikované na matici jsou opět prováděny po prvcích

```
> K <- matrix(1:12,4,3))
> M+K
> M-K
> -3*M                (tj. každý prvek je vynásoben -3)
> M/2                 (tj. každý prvek je vydělen 2)
> M^2                 (tj. každý prvek je umocněn na 5.)
> M/K
> M*K                 (součin po prvcích!)
```

- pokud bychom chtěli součin 2 matic, použijeme místo obyčejného znaménka pro součin (tj. *) následující %*%

```
> M%*%K
```

Matice

- `cbind()`, resp. `rbind()` složí vektory uvedené v závorce do matice po sloupcích, resp. po řádcích

```
> v <- c(1,3,9)
> w <- c(0,-1,6)
> cbind(v,w)
> rbind(v,w)
```

- pro transponovaní matice slouží příkaz `t()`

```
> t(M)
```

Pole

- pole je k-dimenzionální struktura
- `array(v,c(index1, index2, ..., indexk))` vytvoří z vektoru `v` pole o rozměrech
  `index1`$\times$`index2`$\times\cdots\times$`indexk`
  ```
  > (A <- array(v,c(2,3,2)))
  ```
- `dim()` zjistí rozměry pole uvedeného v závorce
  ```
  > dim(A)
  [1] 2 3 2
  ```

Datová tabulka

- datová tabulka je tabulka dat, kde sloupce označují sledované znaky a řádky označují jednotlivá pozorování
- vhodná pro ukládání dat a jejich další zpracování
- `data.frame(sloupec1=data1, sloupec2=data2, ...)` vytvoří datovou tabulku s definovanými názvy sloupců (sloupec1, sloupec2, ...) a jejich hodnotami (data1, data2, ...)

```
> (studenti <- data.frame(fakulta=c("FT","FP","FS",
"FS","FT"), vek=c(23,28,25,24,24)))
   fakulta vek
1       FT  23
2       FP  28
3       FS  25
4       FS  24
5       FT  24
```

Datová tabulka

- přidaný argument `row.names=c("","",...)` udává názvy řádků, tj. zkoumaných jednotek

```
> studenti <- data.frame(fakulta=c("FT","FP","FS","FS","FT
row.names=c("Dana","Klara","Jan","Petr","Martina"))
        fakulta vek
Dana         FT  23
Klara        FP  28
Jan          FS  25
Petr         FS  24
Martina      FT  24
```

- `names()`, kde v závorce je uveden název datové tabulky, vypíše názvy sloupců, tj. sledovaných znaků

```
> names(studenti)
[1] "fakulta""vek"
```

Datová tabulka

– chceme-li k datové tabulce přidat další sloupec, tj. další sledovaný znak, napíšeme za název datové tabulky znak dolaru $ a název nového sloupce (vše bez mezer) a do něj uložíme vektor s příslušnými hodnotami

```
> studenti$rocnik <- c(2,3,3,1,2)
> studenti
        fakulta vek rocnik
Dana         FT  23      2
Klara        FP  28      3
Jan          FS  25      3
Petr         FS  24      1
Martina      FT  24      2
```

Datová tabulka

- chceme-li naopak sloupec z datové tabulky vymazat, přiřadíme do tohoto sloupce NULL

```
> studenti$vek <- NULL
> studenti
        fakulta rocnik
Dana         FT      2
Klara        FP      3
Jan          FS      3
Petr         FS      1
Martina      FT      2
```

- na sloupce datové tabulky lze opět aplikovat vhodné funkce (pokud to dává smysl)

Seznam

- nejobecnější datová struktura
- sdružuje několik různých datových typů/struktur do jednoho velkého objektu, jednotlivé části (tj. datové typy/struktury) se nazývají složky
- seznam je výstupem mnoha funkcí v R, např. histogramu

```
> histogram <- hist(1:10)
> histogram
```

Grafické výstupy

- high-level funkce: vytváří graf jako takový plus osy a popisky
- low-level funkce: přidává do již existujícího grafu další prvky (např. legendu, další body, mřížku, ...)
- `plot(x,y)`, kde `x` a `y` jsou vektory o stejné délce, vytvoří bodový graf

```
> x <- c(-1, 5, 9, 0, 6, 3)
> y <- c(0, 2, -6, 7, 9, -1)
> plot(x,y) # bodovy graf
```

- původní graf se vždy přepíše novým!

```
> z <- seq(from=-5,to=5,by=0.01)
> plot(z,sin(z)) # graf funkce sinus od -5 do 5
```

Grafické výstupy

- vybrané argumenty funkce `plot`:
  - `main`: název grafu, v uvozovkách ""
    ```
    > plot(z,sin(z),main="Sinus")
    ```
  - `sub`: podnázev grafu, v uvozovkách "", umístěn pod grafem, menší písmo
    ```
    > plot(z,sin(z),main="Sinus",sub="y=sin(x)")
    ```
  - `type`: typ grafu, v uvozovkách "", např. `type="p"` vykresluje body, `type="l"` vykresluje křivku, `type="b"` vykresluje obojí
    ```
    > plot(z,sin(z),main="Sinus",sub="y=sin(x)",type="l")
    ```
  - `xlab, ylab`: názvy os, v uvozovkách ""
    ```
    > plot(z,sin(z),main="Sinus",sub="y=sin(x)",type="l",
    xlab="x",ylab="y")
    ```
  - `xlim, ylim`: hodnoty os (maximum a minimum), dvousložkové vektory
    ```
    > plot(z,sin(z),main="Sinus",sub="y=sin(x)",type="l",
    xlab="x",ylab="y",xlim=c(-10,10),ylim=c(-2,2))
    ```

Grafické výstupy

- vybrané argumenty funkce `plot` - pokračování:
  - `col`: barva grafu, v uvozovkách "", např. `col="red"`, `col="blue"`, `col="green"`

    ```
    > plot(z,sin(z),main="Sinus",sub="y=sin(x)",type="l",
    xlab="x",ylab="y",xlim=c(-10,10),ylim=c(-2,2),col="green")
    ```
  - `pch`: typ vykreslených bodů, např. `pch=0` vykresluje prázdné čtverečky, `pch=1` vykresluje prázdná kolečka, `pch=17` vykresluje plné trojúhelníčky

    ```
    > plot(x,y,pch=17)
    ```
  - `cex`: velikost vykreslených bodů

    ```
    > plot(x,y,pch=17,cex=2)
    ```
  - `lty`: typ čáry, např. `lty=1` vykresluje nepřerušovanou čáru, `lty=2` vykresluje přerušovanou čáru, `lty=3` vykresluje tečkovanou čáru

    ```
    > plot(z,sin(z),main="Sinus",sub="y=sin(x)",type="l",
    xlab="x",ylab="y",xlim=c(-10,10),ylim=c(-2,2),col="green",
    lty=2)
    ```
  - `lwd`: tloušťka čáry

    ```
    > plot(z,sin(z),main="Sinus",sub="y=sin(x)",type="l",
    xlab="x",ylab="y",xlim=c(-10,10),ylim=c(-2,2),col="green",
    lty=2,lwd=3)
    ```

Grafické výstupy

Další významné high-level funkce

- curve()
  `> curve(sin)` ... vykreslí graf funkce $f : y = \sin x$ na intervalu $\langle 0, 1 \rangle$
  `> curve(sin,from=0,to=2*pi)` ... vykreslí graf funkce $f : y = \sin x$ na intervalu $\langle 0, 2\pi \rangle$
  `> curve(x^2-3*x)` ... vykreslí graf funkce $f : y = x^2 - 3x$ na intervalu $\langle 0, 1 \rangle$
- barplot()

```
> x <- factor(c(1,1,0,1,0,1))
> barplot(table(x))
```

  - horizontálně uspořádané sloupce: argument `horiz=TRUE`
  - názvy kategorií: argument `names.arg=c()`

Grafické výstupy

- pie()

```
> pie(table(x))
```

  - lze přidat i procenta: `> kategorie <- c("muz","zena")`

```
> procenta <- round(table(x)/length(x)*100)
> kategorie <- paste(kategorie, procenta)
> kategorie <- paste(kategorie, "znak procento", sep="")
> pie(table(x), labels=kategorie)
```

  - lze vykreslit i 3D koláčový graf (potřeba balíku `plotrix`): `pie3D()`

- hist()

```
> x <- c(3,5,7,2,4,3,1,0,7,4,4,2)
> hist(x)
```

  - argument `breaks`: volba počtu intervalů, implicitně nastaveno na výpočet pomocí Sturgesova pravidla $k = 1 + 3.3 \log_{10} n$, kde n je počet pozorování
  - argument `freq`: vykresleny absolutní četnosti, pokud `freq=TRUE`, jinak relativní četnosti
  - argument `right`: intervaly zprava uzavřené a zleva otevřené, pokud `right=TRUE`; pokud `right=FALSE`, naopak
  - argument `labels`: nad sloupec je vypsána četnost, pokud `labels=TRUE`

Grafické výstupy

Vybrané low-level funkce

- `points(x, y, col, pch, cex, ...)`: přidá do již existujícího grafu body

```
> x <- c(-5,2.5)
> y <- c(0,1)
> points(x,y) # prida do grafu 2 body
```

- `lines(x, y, col, lty, lwd, ...)`: přidá do grafu lomenou čáru mezi body o souřadnicích `x, y`

```
> w <- seq(from=-10,to=10,by=0.01)
> lines(w,cos(w),type="l",col="red",lwd=2) # prida graf
funkce kosinus od -10 do 10
```

- `segments(x1, y1, x2, y2, col, lty, lwd)`: přidá do grafu úsečku s krajními body [x1, y1] a [x2, y2]

```
> segments(0,0,0,1,col="blue",lwd=2) # prida usecku
spojujici body [0,0] a [0,1]
```

Grafické výstupy

Vybrané low-level funkce - pokračování

- `abline(a, b, ...)`: přidá do grafu přímku o rovnici $y = a + bx$

```
> abline(1,-2,col="yellow",lwd=2) # primka o rovnici y=1-2
```

- `abline(h=a)`, resp. `abline(v=a)`: přidá do grafu vodorovnou, resp. svislou přímku o rovnici $y = a$, resp. $x = a$

```
> abline(h=3,col="green",lwd=2) # primka o rovnici y=3
> abline(v=1.5,col="red",lwd=2) # primka o rovnici x=1.5
```

- `grid(n1, n2)`: přidá do grafu mřížku, $n1 \times n2$ udává počet obdélníků, na které je graf rozdělen

```
> grid(10,10)
```

- `legend(x, y, nazvy, ...)`: přidá do grafu legendu, horní pravý roh boxu s legendou má souřadnice $[x, y]$

```
> legend(2,2,c("sinus","kosinus"),col=c("green","red"),
lty=c(1,1),lwd=c(2,2))
```

- `text(x, y, text, ...)`: přidá do grafu text, střed textu má souřadnice $[x, y]$

```
> text(-5,-1.5,"komentář")
```

Grafické výstupy

- `split.screen(c(n,m))` rozdělí grafické okno na $n \times m$ částí uspořádané do n řádků a m sloupců
- `screen(cislo)` aktivuje část grafického okna číslo `cislo`

Podmíněné příkazy

RELAČNÍ OPERÁTORY	
==	rovná se
!=	nerovná se
<	menší
>	větší
<=	menší nebo rovno
>=	větší nebo rovno
LOGICKÉ OPERÁTORY	
&	a
|	nebo

Podmíněné příkazy

`if`

- syntaxe: `if(podminka) {prikaz}`
- je-li splněna podmínka `podminka`, pak se provede příkaz `prikaz`, v opačném případě se nic neprovede
- *Je (v reálných číslech) definována druhá odmocnina čísla* x*?*

```
> x <- 2
> if (x >= 0) {"Druha odmocnina je definovana."}
[1] "Druha odmocnina je definovana."
> x <- -5
> if (x >= 0) {"Druha odmocnina je definovana."}
```

- nelze použít na vektor (a ani žádné následující podmíněné příkazy)

Podmíněné příkazy

`if else`

- syntaxe: `if(podminka) {prikaz1} else {prikaz2}`
- je-li splněna podmínka `podminka`, pak se provede příkaz `prikaz1`, v opačném případě se provede příkaz `prikaz2`
- *Je (v reálných číslech) definována druhá odmocnina čísla* x*?*

```
> x <- 2
> if (x >= 0) {"Druha odmocnina je definovana."} else {"Dr
[1] "Druha odmocnina je definovana."
> x <- -5
> if (x >= 0) {"Druha odmocnina je definovana."} else {"Dr
[1] "Druha odmocnina neni definovana."
```

Podmíněné příkazy

`ifelse`

- alternativa k if else příkazu
- syntaxe: `ifelse(podminka, prikaz1, prikaz2)`
- je-li splněna podmínka `podminka`, pak se provede příkaz `prikaz1`, v opačném případě se provede příkaz `prikaz2`
- *Je (v reálných číslech) definována druhá odmocnina čísla* x*?*

```
> x <- 2
> ifelse(x >= 0, "Druha odmocnina je definovana.", "Druha
[1] "Druha odmocnina je definovana."
> x <- -5
> ifelse(x >= 0, "Druha odmocnina je definovana.", "Druha
[1] "Druha odmocnina neni definovana."
```

Podmíněné příkazy

`if else if else ... else`

- pokud rozlišujeme více než dva stavy
- syntaxe:

```
if (podminka1) {prikaz1} else if
(podminka2) {prikaz2} else if
(podminka3) {prikaz3} else if
⋮
else {prikazk}
```

- je-li splněna podmínka `podminka1`, pak se provede příkaz `prikaz1`, je-li splněna podmínka `podminka2`, pak se provede příkaz `prikaz2`, atd., není-li splněna žádná z uvedených podmínek, provede se příkaz `prikazk`
- *Je číslo x kladné, záporné, nebo nula?*

```
> x <- -0.2
> if (x > 0) {"Kladne cislo."} else if
+ (x < 0) {"Zaporne cislo."} else
+ {"Nula."}
```

Příkazy cyklů

- pro opakované provedení příkazů

`for`

- používáme, pokud víme počet opakování
- syntaxe: `for (promenna in rozsah) {prikazy}`
  `promenna` ... indexační proměnná
  `rozsah` ... hodnoty proměnné `promenna` (zpravidla vektor)
  `prikazy` ... posloupnost příkazů, které mají být vykonány; ve složených závorkách; odděleny středníkem nebo každý příkaz na jeden řádek (doporučeno)
- vytištění hodnoty pomocí příkazu `print()`
- *Určete druhé mocniny přirozených čísel od 1 do 10.*

```
> for (i in 1:10) {print(i*i)}
```

Příkazy cyklů

`while`

- používáme, pokud nevíme počet opakování - závisí na jisté podmínce
- syntaxe: `while (podminka) {prikazy}`
- dokud je splněna podmínka `podminka`, provádějí se příkazy `prikazy`
- *Vypište takové druhé mocniny přirozených čísel, které jsou menší než 650.*

```
> i <- 1
> while (i**2 < 650) {print(i**2); i <- i+1}
```

Příkazy cyklů

`repeat`

- alternativa k `while`
- pro ukončení se používá příkaz `break`, kde v závorce je uvedena ukončovací podmínka
- syntaxe:
  `repeat {prikazy`
  `if (podminka) {break}}`
- dokud není splněna podmínka `podminka`, provádějí se příkazy `prikazy`
- *Vypište takové druhé mocniny přirozených čísel, které jsou menší než 650.*

```
> i <- 1
> repeat {print(i**2)
+ i <- i+1
+ if (i**2>=650) {break}
```

Poznámka: For cyklus lze obejít použitím funkcí `apply()`, `lapply()`, `sapply()` a `tapply()`.

Tvorba funkcí

- v R možnost vytvářet vlastní funkce
- syntaxe funkce je vždy ve tvaru
  `nazev <- function(arg1, arg2, ...){telo}`
  `nazev` ... název funkce (vyhýbat se názvům již existujících funkcí)
  `arg1, arg2` ... posloupnost vstupních hodnot, odděleny čárkou
  `telo` ... posloupnost příkazů, které mají být vykonány; ve složených závorkách; odděleny středníkem nebo každý příkaz na jeden řádek (doporučeno)
- funkce lze psát rovnou do konzole nebo ji lze vytvořit v editoru, tj. *File* $\rightarrow$ *New File* $\rightarrow$ *R Script*
- výstupní hodnota je vrácena pomocí příkazu `return()`

Tvorba funkcí

- *Vytvořte funkci pro výpočet objemu kvádru, jsou-li zadány délky stran* a, b, c.

```
> kvadr.objem <- function(a,b,c){
+ V <- a*b*c
+ return(V)}
```

- pokud je výstupních hodnot více, je potřeba sdružit tyto hodnoty do seznamu, tj. `return(list=c(...))`
- *Vytvořte funkci pro výpočet objemu a povrchu kvádru, jsou-li zadány délky stran* a, b, c.

```
> kvadr <- function(a,b,c){V <- a*b*c
+ S <- 2*(a*b+b*c+a*c)
+ return(list=c(objem=V,povrch=S))}
```

- zapsáním názvu funkce do konzole je vypsána definice funkce

```
> kvadr
```

- zapsáním názvu funkce do konzole včetně konkrétních vstupních hodnot v závorce je vypsán výsledek/výsledky funkce

```
> kvadr(4,8,3)
```

Tvorba funkcí

- pokud je vytvořena funkce v editoru, je potřeba ji uložit jako soubor s příponou `.R` a následně spustit pomocí příkazu `source()`, kde v závorce je uveden název funkce v uvozovkách (POZOR! Nejprve je potřeba nastavit správný adresář.)
- příkaz `warning()` vytiskne upozorňující hlášku

```
> kvadr <- function(a,b,c){
+ V <- a*b*c
+ S <- 2*(a*b+b*c+a*c)
+ warning("Delky stran museji byt kladna cisla!")
+ return(list=c(objem=V,povrch=S))}
```

- příkaz `stop()` zastaví provádění příkazů a vytiskne chybovou hlášku, když je splněna podmínka

```
> kvadr <- function(a,b,c){
+ if (a<=0 | b<=0 | c<=0) stop("Delky stran museji byt kla
+ V <- a*b*c
+ S <- 2*(a*b+b*c+a*c)
+ return(list=c(objem=V,povrch=S))}
```