KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI
www.inf.upol.cz

Michal Krupka
krupka.inf.upol.cz
michal.krupka@upol.cz
17. listopadu 12, 771 46 Olomouc

Paradigmata programování 1◇ poznámky k přednášce

1. Symbolické výrazy

verze z 14. října 2019

1 Úvod

Cílem čtyřsemestrálního kurzu Paradigmata programování je seznámit studenty s principy programování a základními přístupy a styly v programování používanými (tzv. *paradigmaty*). Budeme to dělat bez ohledu na to, jaké přístupy se v současné době zrovna používají v praxi — zkušenost říká, že ty se rychle mění a vyvíjejí, zatímco principy zůstávají mnoho desetiletí stále stejné. (A koneckonců, když se podíváme, jak mnohdy fungují softwarové produkty, které v *praxi* vznikají, nezískáme dojem, že by se zrovna z ní bylo dobré ve všem inspirovat.)

Budeme se zabývat programovací jazykem Common Lisp. Je třeba, abyste si naistalovali a správně nastavili aplikaci LispWorks (Personal Edition). Podrobnosti o tom, jak to udělat, jsou v jiném dokumentu.[1]

Pokud se vám LispWorks a jeho omezení platná pro Personal Editions nelíbí, můžete použít i jinou implementaci jazyka, například SBCL nebo Clozure Common Lisp. Tuto volbu ale doporučuji jen hodně zběhlým uživatelům; vyučující vám s problémy, které si tím případně způsobíte, nebudou pomáhat.

2 Lisp jako kalkulačka

Po spuštění LispWorks se otevře okno, kterému říkáme **Listener** (*posluchač*). V něm vstupujeme do kontaktu s programem LispWorks. (Často budu, i když nepřesně, říkat, že s jazykem Lisp.)

Okno můžeme používat třeba na počítání s čísly, podobně jako kalkulačku (ale i na další věci, o kterých se dozvíme později). Napíšeme do něj, co chceme vypočítat, a dostaneme výsledek:

```
CL-USER 1 > 1
1

CL-USER 2 > (+ 1 1)
2
```

[1] Stručně: aplikaci stáhnete z webu LispWorks. A pak přinejmenším nastavíte v předvolbách Environment -> Editor keys na Windows nebo jinou platformu, kterou používáte.

Nejprve jsme zkusili vypočítat číslo 1. To můžeme chápat jako už vypočítanou hodnotu, takže není divu, že se nám jako výsledek vrátilo číslo 1.

Jako druhý pokus jsme vypočítali číslo $1 + 1$. Museli jsme ovšem výpočet zadat ve tvaru, kterému jazyk Lisp rozumí: nejprve napsat operaci, kterou chceme, aby provedl, a pak čísla, se kterými ji má provést. Vše uzavřít do kulatých závorek.

Další příklady:

```
CL-USER 4 > (/ 4 2)
2

CL-USER 5 > (- 4 2)
2

CL-USER 6 > (* 4 2)
8

CL-USER 7 > (+ 1 2 3 4)
10

CL-USER 8 > (* 1 2 3 4)
24

CL-USER 9 > (/ 24 6 2)
2

CL-USER 10 > (+)
0

CL-USER 11 > (+ (/ -4 2) (* -4 2))
-10
```

Vidíme, že kromě sčítání můžeme i odečítat, násobit a dělit. Můžeme to dělat i s více než dvěma čísly. A můžeme i dělat složitější výpočty s více operacemi.

Kromě celých čísel můžeme pracovat i s čísly racionálními a obecně reálnými:

```
CL-USER 12 > (/ 4 6)
2/3

CL-USER 13 > (/ 1234 2345)
1234/2345

CL-USER 14 > pi
3.141592653589793D0
```

Poslední vyhodnocení nás poučí, že v Lispu můžeme používat číslo π tak, že na-

píšeme jeho anglický název. Znaků `D0` na konci čísla si zatím nemusíme všímat, za chvilku se dozvíme, co znamenají. Jinak, jak vidíme, v zápisu necelých čísel se používá desetinná tečka místo čárky.

Takto bychom tedy mohli vypočítat obvod kružnice o poloměru `r`:

```
CL-USER 15 > (* 2 pi r)
```

...kdyby ovšem Listener věděl, kolik je `r`. To on ovšem neví, takže nám nahlásí chybu:

```
Error: The variable R is unbound.
  1 (continue) Try evaluating R again.
  2 Specify a value to use this time instead of evaluating R.
  3 Specify a value to set R to.
  4 (abort) Return to top loop level 0.

Type :b for backtrace or :c <option number> to proceed.
Type :bug-form "<subject>" for a bug report template or :? for
other options.

CL-USER 16 : 1 >
```

Chybová hláška `Error: The variable R is unbound` na prvním řádku znamená, že Lisp nezná hodnotu `R`. Další řádky můžeme zatím ignorovat. (Jak jste si teď mohli všimnout, Lisp nerozlišuje, zda je symbol psán malými, nebo velkými písmeny, a sám je píše velkými.)

Z uvození příkazového řádku `CL-USER 16 : 1 >` poznáme, že Listener je v *chybovém stavu* a je připraven pomoci nám řešit nastalý problém. To se v budoucnu bude hodit, nyní ale z chybového stavu odejdeme tak, že napíšeme `:a`

```
CL-USER 16 : 1 > :a

CL-USER 17 >
```

Teď si aspoň vypočítáme obvod kružnice o poloměru 10:

```
CL-USER 17 > (* 2 pi 10)
62.83185307179586D0
```

Když už jsme u chyb, ukážeme si další pokus o výpočet, který vyvolá chybu:

```
CL-USER 23 > (/ 1 0)

Error: Division-by-zero caused by / of (1 0).
  1 (continue) Return a value to use.
  2 Supply new arguments to use.
  3 (abort) Return to top loop level 0.

Type :b for backtrace or :c <option number> to proceed.
Type :bug-form "<subject>" for a bug report template or :? for
other options.

CL-USER 24 : 1 >
```

Jsme v chybovém stavu, ze kterého se opět dostaneme pomocí `:a`

```
CL-USER 24 : 1 > :a

CL-USER 25 >
```

A můžeme pokračovat. Kromě operací s čísly můžeme používat i další funkce, například odmocninu (*square root*, náš název je `sqrt`) a goniometrické funkce:

```
CL-USER 25 > (sqrt 16)
4.0

CL-USER 26 > (sqrt 2)
1.4142135

CL-USER 27 > (sqrt -1)
#C(0.0 1.0)

CL-USER 28 > (sin (/ pi 2))
1.0D0
```

Jak vidíte, Lisp umí vypočítat i odmocninu z -1. Komplexní číslo i zapsal jako `#C(0.0 1.0)`. Některé výsledky ale samozřejmě nevypočítal přesně, jak se můžeme snadno přesvědčit:

```
CL-USER 29 > (* (sqrt 2) (sqrt 2))
1.9999999
```

Číslo $\sin \pi$ (které je, jak víme, rovno nule) Lisp také nevypočte přesně:

```
CL-USER 30 > (sin pi)
1.2246063538223773D-16
```

Znaky `D-16` na konci ukazují, že jde o zápis čísla v *exponenciálním tvaru*. Výsledek je zhruba $1{,}22 \cdot 10^{-16}$, tedy 0,0000000000000000122.

Ve škole jsme se učili, že $\sin \frac{\pi}{6} = \frac{1}{2}$. Jestlipak to v našem Listeneru vyjde stejně?

```
CL-USER 31 > (sin (/ pi 6))
0.49999999999999995D0
```

Vidíme, že (téměř) ano. A poslední příklad: Víme, že $\sin \frac{\pi}{4} = \frac{\sqrt{2}}{2}$, takže když ho umocníme na 2 (neboli vynásobíme sebou samým), měli bychom dostat $\frac{1}{2}$:

```
CL-USER 32 > (* (sin (/ pi 4)) (sin (/ pi 4)))
0.4999999999999999D0
```

Obecně platí, že pokud je číslo zapsáno s desetinnou tečkou (ať už s koncovým „`D`“, nebo ne), nesmíme se spoléhat, že jde o přesný výsledek. Celá čísla zapsaná bez tečky a zlomky jsou přesné. [2]

Pokus o použití funkce, kterou Listener nezná, vede k chybě:

```
CL-USER 33 > (funkce 10)

Error: Undefined operator FUNKCE in form (FUNKCE 10).
  1 (continue) Try invoking FUNKCE again.
  2 Return some values from the form (FUNKCE 10).
  3 Try invoking something other than FUNKCE with the same
arguments.
  4 Set the symbol-function of FUNKCE to another function.
  5 Set the macro-function of FUNKCE to another function.
  6 (abort) Return to top loop level 0.

Type :b for backtrace or :c <option number> to proceed.
Type :bug-form "<subject>" for a bug report template or :? for
other options.

CL-USER 34 : 1 >
```

[2] Různé zápisy čísel v Lispu (a v ostatních programovacích jazycích) souvisí s tím, že počítač si čísla v paměti ukládá různým způsobem a různě s nimi pracuje. Podrobnosti o tom se dozvíte v jiném předmětu.

3 Symbolické výrazy

Do okna Listeneru píšeme tzv. *symbolické výrazy*. Symbolický výraz je, jednoduše řečeno, text, kterému Listener rozumí a který pochopí jako zadání výpočtu. Nyní si řekneme, jak správně utvořený symbolický výraz vypadá.

Symbolický výraz (expression, form) je

- jednoduchý výraz, neboli *atom*, nebo
- složený výraz, neboli *seznam*.

Jednoduchý výraz (atom) je

- číslo nebo
- symbol nebo
- ... (o dalších možnostech si řekneme později)

Číslo je zapsané jedním ze způsobů, které jsme již uvedli, např. `10`, `-5`, `2/3`, `0.0`, `0.4999999999999999D0`, `1.2246063538223773D-16`, `#C(0.0 1.0)` a podobně.

Symbol je posloupnost písmen a případně čísel a dalších znaků, která neoznačuje číslo. Symboly neobsahují některé nepovolené znaky, například `#`, závorky, mezery a další prázdné znaky. Symboly jsou například `pi`, `r`, `a1`, `sqrt`, `+`, `/`, `1+1`. Například `-5` symbol není, protože to je číslo.

Složený výraz (seznam) je posloupnost jednoho nebo více výrazů. Prvky složeného výrazu jsou odděleny mezerami nebo jinými prázdnými znaky. Složený výraz začíná levou a končí pravou kulatou závorkou.

Příklady složených výrazů:

- `(+ 1 2)`
- `(+)`
- `(1 2 3)`
- `(1 + 2 + 3)`
- ```
  (* (sin (/ pi 4))
     (sin (/ pi 4)))
  ```

Poslední složený výraz má tři prvky, některé z nich jsou složené výrazy.

Toto *nejsou* symbolické výrazy:

- `)`
  ```

- `(+ (- 10 11)`
- `ah oj`

Abychom mohli v Lispu pracovat, musíme umět složené symbolické výrazy vytvářet. Vysvětlím to na příkladu výrazů vyjadřujících matematický výpočet.

Vytváření symbolických výrazů

První symbol v seznamu označuje **název** výrazu (např. matematického: součet, rozdíl, součin, podíl). Označuje tedy **poslední** operaci prováděnou ve výpočtu.

Například výraz

$$\frac{5-3}{5+3}$$

je podíl rozdílu a součtu:

```
(/ (- 5 3) (+ 5 3))
```

Při výpočtu se tedy podíl provede jako poslední operace.

Pro přehlednost můžeme výraz rozdělit na více řádků:

```
(/ (- 5 3)
   (+ 5 3))
```

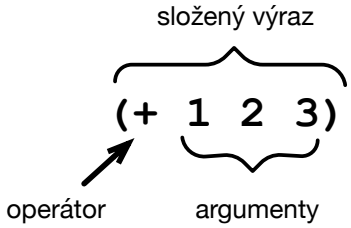
Výraz

$$\sin\left(\frac{5-3}{5+3}\cdot\pi\right)$$

je sinus součinu dříve uvedeného podílu a čísla π:

```
(sin (* (/ (- 5 3) (+ 5 3))
        pi))
```

Terminologie pro složené výrazy (seznamy)



Operátor: *co* se má udělat

Argumenty: *s čím* se to má udělat

Terminologii byste měli dobře ovládat, abyste chápali, o čem se hovoří nebo píše, a abyste se sami o programování v Lispu mohli vyjadřovat.

Prefixová notace

Lisp důsledně používá tzv. **prefixovou notaci**: operátor je vždy uveden *před* argumenty. To je velké zjednodušení proti ostatním programovacím jazykům i matematice:

prefix: $\sin x$, -5

infix: $x + y \cdot z - 3$ (komplikace: přednost operací)

postfix: $10!$

a další: x^y, $\sqrt[3]{10}$, $\frac{5}{6}$, $\bar{z}$, $|z|$, $\int_a^b 2x \, \mathrm{d}x$

Na důsledně prefixovou notaci je třeba si ale zvyknout.

4 Vyhodnocování symbolických výrazů

Když napíšeme do Listeneru symbolický výraz (a stiskneme Enter), dostaneme výsledek (samozřejmě pokud nedojde k chybě). Tomuto výsledku říkáme *hodnota symbolického výrazu*. Například

Číslo $\frac{1}{4}$ je **hodnotou symbolického výrazu** `(/ (- 5 3) (+ 5 3))`.

Počítač k hodnotě výrazu dojde přesně popsaným procesem, kterému se říká **vyhodnocovací proces**. Následuje jeho (zatím zjednodušený) popis:

Vyhodnocení výrazu E

Je-li E symbol, výsledkem je **hodnota** symbolu E.

Je-li E jiný atom než symbol, výsledkem je E.

Je-li E seznam s operátorem o a argumenty $a_1 \dots a_n$, pak

1. se zjistí **funkce** f, kterou operátor o označuje,
2. zjistí se hodnoty $v_1 \dots v_n$ argumentů $a_1 \dots a_n$ (opět vyhodnocovacím procesem).
3. Výsledkem je výsledek **aplikace** funkce f na hodnoty $v_1 \dots v_n$.

V popisu zjednodušeného vyhodnocovacího procesu jsme narazili na tři pojmy, kterým zatím nerozumíme: *hodnota symbolu*, *funkce* a *aplikace funkce*. Pojďme si je vysvětlit.

Hodnota symbolu

Symbol může sloužit jako **jméno** jiné hodnoty. S takovým symbolem jsme se už i setkali:

```
CL-USER 14 > pi
3.141592653589793D0
```

Hodnotou symbolu `pi` je číslo `3.141592653589793D0`. Pokud symbol slouží jako jméno hodnoty, říkáme mu **proměnná (variable)**. Také říkáme, že je symbol na svou hodnotu *navázán*. Symbol `r` nemá hodnotu:

```
CL-USER 35 > r

Error: The variable R is unbound.
  1 (continue) Try evaluating R again.
  2 Specify a value to use this time instead of evaluating R.
  3 Specify a value to set R to.
  4 (abort) Return to top loop level 0.

Type :b for backtrace or :c <option number> to proceed.
Type :bug-form "<subject>" for a bug report template or :? for
other options.

CL-USER 36 : 1 >
```

Malá odbočka: symboly `*`, `+` a další (především `**` a `***`), mají v Listeneru hodnotu, kterou můžeme využívat. Můžete si sami zjistit, o jakou hodnotu jde.

Symbol `cos` nebo třeba `sqrt` také nemá hodnotu. Slouží ale k označení funkce. **Funkci** můžeme chápat jako nějaký výpočet, který počítač umí provádět. Takovým výpočtem může být třeba provedení aritmetické operace (jako je sčítání nebo rozdíl

čísel) nebo výpočet hodnoty matematické funkce (jako je kosinus nebo odmocnina) a mnoho dalších věcí.

Funkce v Lispu jsou označovány symboly. Například symbol `+` označuje funkci, která umí sčítat čísla (provádět aritmetickou operaci sčítání), symbol `sin` označuje funkci na výpočet sinu (tj. funkci, která umí počítat hodnoty matematické funkce sinus). Symbol označující funkci se také nazývá její *název*.

Jak je napsáno výše, výpočet výsledku funkce se spustí tak, že se funkce *aplikuje na vstupní hodnoty*. Aplikace funkce se v rámci vyhodnocovacího procesu provede sama, jak je popsáno výše.

Funkce

- funkce
- název funkce
- aplikace funkce

Vyhodnocovací proces: příklad

Ukážeme si krok po kroku proces vyhodnocování výrazu `(/ (- 5 3) (+ 5 3))`.

Vyhodnocení výrazu `(/ (- 5 3) (+ 5 3))`
Je to složený výraz, jeho operátor `/` označuje funkci dělení.
Vyhodnotí se argumenty:

> **Vyhodnocení výrazu `(- 5 3)`**
> Je to složený výraz, operátor `-` označuje funkci rozdílu:
> Vyhodnotí se argumenty:
>
> > **Vyhodnocení výrazu `5`**
> > Je to číslo, výsledkem je číslo `5`
> >
> > **Vyhodnocení výrazu `3`**
> > Je to číslo, výsledkem je číslo `3`
>
> Výsledkem je aplikace funkce `-` na `5` a `3`, tedy číslo `2`
>
> **Vyhodnocení výrazu `(+ 5 3)`**
> Je to složený výraz, jeho operátor `+` označuje funkci sčítání
> pak se vyhodnotí argumenty:
>
> > **Vyhodnocení výrazu `5`**
> > Je to číslo, výsledkem je číslo `5`
> >
> > **Vyhodnocení výrazu `3`**
> > Je to číslo, výsledkem je číslo `3`
>
> Výsledkem je aplikace funkce `+` na hodnoty `5` a `3`, tedy `8`

Výsledkem je aplikace funkce `/` na hodnoty `2` a `8`, tedy `1/4`

5 Logické hodnoty a predikáty

Ukázali jsme si několik funkcí, které realizují výpočet matematických operací a matematických funkcí. Například funkce `+ -`, `*`, `/` počítají hodnoty operací, procedury `sin`, `cos`, `sqrt` hodnoty funkcí. Je jistě výhoda, že funkce lze použít jak na označení operací, tak matematických funkcí (a pomáhá nám v tom prefixová notace).

Existuje ještě jeden základní matematický pojem, a to pojem *relace*. Rozdíl mezi operacemi a funkcemi na jedné straně a relacemi na druhé je, že zatímco operace a funkce mají jako výsledek hodnotu (v našem případě číslo), relace popisují *vztah* mezi hodnotami. Například základní relace mezi čísly: $=$, $<$, $>$, $\leq$, $\geq$. Relace nám říkají, *jestli* jsou dvě čísla v určitém vztahu[3]. Výraz, ve kterém vystupuje relace, se dá chápat, jako *tvrzení*, které platí nebo neplatí. Například $1 < 2$ je pravda, zatímco $1 \geq 2$ není pravda.

Z tohoto přístupu k relacím vychází možnost realizovat relace v Lispu pomocí funkcí. Jazyk nám poskytuje dvě tzv. *pravdivostní* (*logické*) *hodnoty* `t` a `nil`, které symbolizují pravdu a nepravdu. Matematické relace jsou v jazyce realizovány pomocí funkcí, které logické hodnoty vracejí jako výsledek:

```
CL-USER 37 > (= 1 1)
T

CL-USER 38 > (= 1 2)
NIL

CL-USER 39 > (< (/ 2 2) 2)
T

CL-USER 40 > (>= 1 (+ 1 1))
NIL
```

Funkce, které vracejí jako výsledek logické hodnoty (a svou hodnotou tedy odpovídají na otázku, zda je něco pravda, nebo ne), se nazývají *predikáty*.

Hodnoty `t` a `nil` jsou symboly, které mají tu zvláštnost, že se stejně jako čísla vyhodnocují na sebe:

```
CL-USER 41 > t
T

CL-USER 42 > nil
NIL
```

Pomocí logických hodnot můžeme pracovat s tzv. *podmíněnými výrazy*:

[3] Přesnou a matematicky správnou definici relace se dozvíte v jiném předmětu.

```
CL-USER 43 > (if (<= 2 1) 10 11)
11

CL-USER 44 > (if (< 1 2) 1 (/ 1 0))
1
```

Druhý příklad nás vede k pojmu *speciálního operátoru* a *makra*.

6 Speciální operátory a makra

Zopakujme si poslední příklad:

```
CL-USER 44 > (if (< 1 2) 1 (/ 1 0))
1
```

Je vidět, že vyhodnocení tohoto výrazu se neřídí vyhodnocovacím procesem popsaným dříve. Jinak by totiž došlo k chybě (jaké a proč?).

Operátor `if` ve vyhodnocovaném výrazu je totiž vyjímečný (*speciální*) a klasický vyhodnocovací proces pro něj neplatí.

Operátory `if` a `setf`

Podmíněný výraz je složený výraz s operátorem `if`. Takový výraz musí mít tři argumenty. (Lisp umožňuje i verzi s e dvěma argumenty, ale tu nebudeme používat.) Vyhodnocuje se takto:

Vyhodnocení výrazu `(if` a b c`)`

1. Vyhodnotí se a na hodnotu u.
2. Pokud je u rovno `NIL`, vyhodnotí se c a vrátí jeho hodnota.
3. Pokud není, vyhodnotí se b a vrátí jeho hodnota.

Uvědomte si přesný význam bodu 3. Plyne z něj, že hodnotou prvního argumentu operátoru `if` může být i jiná hodnota než `t` nebo `nil`.

Operátor `if` je prvním příkladem operátoru, který neoznačuje funkci. Takovým operátorům říkáme **speciální operátory** a **makra** (čím se makra a speciální operátory liší, zatím nebudeme zkoumat). Výrazy se speciálními operátory a makry mají svá vlastní pravidla vyhodnocování.

Dalším příkladem operátoru, který neoznačuje funkci, je symbol `setf`. Tento operátor označuje makro.

Výraz s operátorem `setf` musí mít dva argumenty, z nichž první musí být symbol. (Lisp opět připouští i jiné možnosti, ty ale nebudeme zatím využívat.) Operátor nastaví jeho hodnotu na hodnotu druhého argumentu.

Vyhodnocení výrazu `(setf` a b`)`

1. Vyhodnotí se b.
2. hodnota symbolu a se nastaví na tuto hodnotu.

Všimněte si, že výraz a se **nevyhodnocuje** (co by se stalo, kdyby se vyhodnocoval?).

Příklad:

```
CL-USER 45 > (setf r 10)
10

CL-USER 46 > (* 2 pi r)
62.83185307179586D0

CL-USER 47 > (setf r 11)
11

CL-USER 48 > (* 2 pi r)
69.11503837897544D0
```

Takto vypadá obecnější popis vyhodnocovacího procesu v jazyce Common Lisp:

Vyhodnocení výrazu E

Je-li E **symbol,** výsledkem je hodnota symbolu E.

Je-li E **jiný atom než symbol,** výsledkem je E.

Je-li E **seznam s operátorem** o **a argumenty** $a_1 \ldots a_n$**,** pak

> **Jestliže** o **je speciální operátor nebo makro,** seznam se vyhodnotí podle pravidel tohoto operátoru.
>
> **Jinak** operátor o musí být symbol. Pak
>
> 1. se zjistí funkce f, kterou symbol o označuje,
> 2. zjistí se hodnoty $v_1 \ldots v_n$ argumentů $a_1 \ldots a_n$ (opět vyhodnocovacím procesem).
> 3. Výsledkem je výsledek aplikace funkce f na hodnoty $v_1 \ldots v_n$.

7 Závěr

Probrané pojmy

Symbolický výraz, jednoduchý výraz (atom), složený výraz (seznam), číslo, symbol, logická (pravdivostní) hodnota. Symbol jako proměnná. Funkce. Operátor, argumenty. Vyhodnocovací proces. Speciální operátor a makro, speciální operátor `if`, makro `setf`.

Otázky a úkoly na cvičení

1. Kolik prvků mají tyto složené výrazy?

```
((((1)))), (1 1), ((1 1)), (((1 1))), ((3 3 3) (1))
```

2. Pokuste se odhadnout, co bude výsledkem vyhodnocení následujících výrazů. Pokud by mělo dojít k chybě, řekněte i, k jaké. Pak ověřte na počítači:

```
(* 2 2 2), (* 2 2), (* 2), (*), (/ 0 1), (/ 1 0),
(/ (/ (/ (/ 16 2) 2) 2) 2), (1 2 3 4), 1, 2,
(if (= 0 (+ (- (+ 10 20) 30))) 7 8),
(+ 1 2), (+1 2), (1 + 2 + 3),
((1 + 2) (1 + 3)), ((+ 1 2) (+ 1 3))
```

3. Přepište do prefixové notace jazyka Lisp a pak vyhodnoťte:

$$\frac{7(1+5.8)}{4(2.51-2.34)}$$

4. Přepište do prefixové notace jazyka Lisp a pak vyhodnoťte:

$$\frac{5+\frac{14}{3}+(2-(3-(6-\frac{4}{3})))}{(1-\frac{2}{3})(2-6)}$$

5. Jaký je rozdíl mezi funkcí a speciálním operátorem nebo makrem?

6. Proč musí být `setf` speciální operátor?

7. Napište výraz, jehož vyhodnocením ověříme, že `if` není funkce.

8. Co je hodnotou následujících výrazů?

```
    t, nil, (if 1 2 3)
```