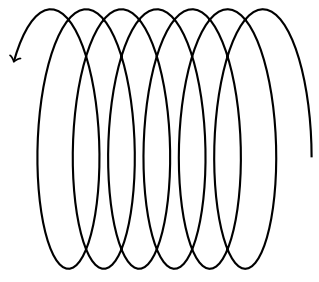


Neprogramátor

CC BY-SA 4.0 neprogramator.cz



(Aaron, The Internet's Own Boy)

... že programování je čarování, že programátoři zvládnou udělat věci, které normální lidé nezvládnou...

Český jazyk

Normálním lidem jde komunikovat. Mají na to jazyk.

- podstatná jména: "kniha"
- přídavná jména: "kouzelná"
- číslovky: 0 3.14 -9
- slovesa: sečti spoj
- ...

velké písmeno → slovosled → . ! ?

↑ syntax ↑ ↓ sémantika ↓

Význam slov a vět se učíme od dětství a chápeme intuitivně.

Sečti 0 a 3.14 a -9!
Spoj "kouzelná" a "kniha".



Sečti 0 a "kniha".

Výsledky výkonání

```

1 -5.859999999999999999
2 "kouzelnáknihá"
3 . . +: contract violation
  expected: number?
  given: "kniha"
13 #f
15 #t
16 #t
17 #t
18 #t
24 18
25 "1.|2."
32 "1.|2."
    
```

Kouzelný jazyk Racket

"kniha" } podstatná jména
"kouzelná" } ukryvají *informaci*
0 3.14 -9

+ sloveso říká *co*
string-append } s podstatnými jmény
(sloveso podstatná jména)

↑ syntax ↑ ↓ sémantika ↓

Význam slov a vět je přesně daný specifikací, ale odhadnutelný!

1 (+ 0 3.14 -9)
2 (string-append "kouzelná" "kniha")

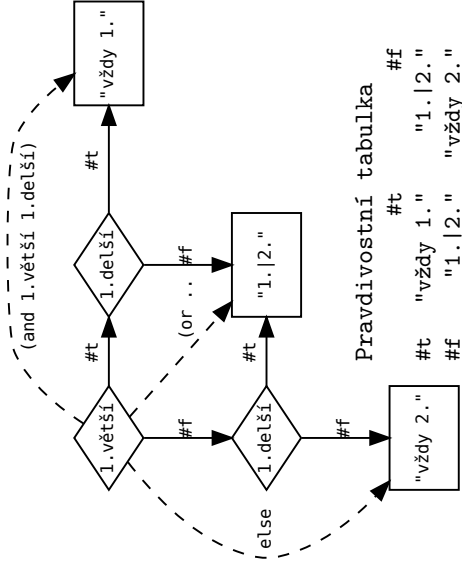
3 (+ 0 "kniha")



Podmínkové výrazy

```

25 (if 1.větší
26 (if 1.delší
27 "vždy 1."
28 "1.|2.")
29 (if 1.delší
30 "1.|2."
31 "vždy 2.")
32 (cond
33 ((and 1.větší
34 "vždy 1."
35 "1.|2.")
36 (if 1.delší
37 "1.|2."
38 "vždy 2.")
39 (else "vždy 2."))
    
```



Kouzelná hůlka

... vykoná kouzelnou větu tak, že z kouzelné věty udělá kouzelné slovo.

- 1 #lang racket
- 2 ; Za středníkem je komentář,
- 3 ; který kouzelná hůlka ignoruje,
- 4 ; takže se hodí pro dokumentaci.
- 5 ;
- 6 ; Kód se spustí z menu "Racket" -> "Run"

Language: racket, with debugging, memory limit: 128 MB.

```

> (+ 0 3.14 -9)
-5.859999999999999999
> (string-append "kouzelná" "kniha")
"kouzelnáknihá"
> (+ 0 "kniha")
expected: number?
given: "kniha"
    
```

Vývojové prostředí DrRacket

číslo	text	logika
0 3.14 -9	"kniha" "3.14"	#t #f

Slovesa

+ - * / sqrt expt	substring	and
modulo round abs	string-number	or
sin max log atan	number->string	not

predikáty

výsledek výkonání je #t nebo #f

= < > <= >= odd? | string=? | equal?

pojmenování informací

definice proměnných

```

4 (define 1.číslo 3.14)
5 (define 2.číslo -9)
6 (define 1.větší (> 1.číslo 2.číslo))
7 (define 1.delší
8 (> (string-length "kniha")
9 (string-length "kouzelná")))
    
```

rekurze

použití funkce při její definici

```

(define (sečti od až) ; včetně
  (if (= od až)
      až ; základní krok / indukční krok
      (+ od (sečti (+ 1 od) až))))
    
```

podstatná jména spojená do jednoho seznamu

```

(define čísla (list 0 3.14 -9))
(equal? čísla (0) , (3.14 -9))
(equal? (car čísla) (cdr čísla))
(equal? čísla ; (cons 0 '(3.14 -9)) (cdr čísla))
(equal? čísla ; (cons 0 '(3.14 -9)) (cdr čísla))
    
```

nové sloveso

definice funkce

```

(define (1.delší? 1.slovo 2.slovo)
  (> (string-length 1.slovo)
      (string-length 2.slovo)))
    
```

podstatná jména

první tvůrčí datové typy

```

("kniha" "kouzelná")
(1.delší? "kniha" "kouzelná")
    
```

Kouzelné čtení

Jazyk je jeden ze způsobů, jak se dají sdílet myšlenky. Použije se při mluvení a poslouchání a psaní a čtení. To, že kouzelná hůlka dokáže vykonat věty zapsané v některém z jazyků, neovlivňuje možnost sdílet v jazyce myšlenky. Vykonání kouzelné věty kouzelnou hůlkou ale souvisí s tím, jak se dá s myšlenkami vyjádřenými v kouzelném jazyce pracovat, protože dost práce udělá právě kouzelná hůlka.

Krátká matematická myšlenka

```
(define (Pythagorova-věta a b c)
  ; a, b jsou odvěsny, c je přepona
  (= (* c c) (+ (* a a) (* b b))))
(define (vypočítej-přeponu-pro-odvěsny a b)
  (sqrt (+ (* a a) (* b b))))
(define (vypočítej-odvěsnu-pro-přeponu-a-odvěsnu c a)
  (sqrt (- (* c c) (* a a))))
```

Krátká chemická myšlenka

```
(define (oxidační-číslo->koncovka oxidační-číslo)
  (cond ((= 1 oxidační-číslo) "-ný")
        ((= 2 oxidační-číslo) "-natý")
        ((= 3 oxidační-číslo) "-itý")
        ((= 4 oxidační-číslo) "-ičitý")
        ((= 5 oxidační-číslo) "-ičný, -ečný")
        ((= 6 oxidační-číslo) "-ový")
        ((= 7 oxidační-číslo) "-istý")
        ((= 8 oxidační-číslo) "-ičelý")
        ((= 9 oxidační-číslo) "-utý"))
```

Několik myšlenek o logice

; základní definice a opak (negace) jedné věty

```
(define pravda #t)
(define (je-pravda? věta)
  (if (equal? pravda věta)
      pravda
      lež))
(define lež #f)
; Z hlediska logiky je nekorektní
; používat lež jako opak pravdy.
(define (je-lež? věta)
  (if (equal? lež věta)
      pravda
      lež))
(define (neplatí-že věta)
  (cond ((je-pravda? věta) lež)
        ((je-lež? věta) pravda)))
```

; možné kombinace dvou vět

```
(define (platí-obě? první-věta druhá-věta)
  (if (je-pravda? první-věta)
      (if (je-pravda? druhá-věta)
          pravda
          lež)
      lež))
(define (platí-první-a-druhá-ne? první-věta druhá-věta)
  (if (je-pravda? první-věta)
      (if (je-lež? druhá-věta)
          pravda
          lež)
      lež))
(define (platí-druhá-a-první-ne? první-věta druhá-věta)
  (if (je-pravda? druhá-věta)
      (if (je-lež? první-věta)
          pravda
          lež)
      lež))
```

```
(define (neplatí-ani-jedna? první-věta druhá-věta)
  (if (je-lež? první-věta)
      (if (je-lež? druhá-věta)
          pravda
          lež)
      lež))
(define (řekni-jestli-platí-že má-být věta)
  (cond ((platí-obě? má-být věta) "platí")
        ((neplatí-ani-jedna? má-být věta) "platí")
        (else "neplatí")))
```

; souvětí dvou vět

```
(define (souvětí-s-a první-věta druhá-věta)
  (cond ((platí-obě? první-věta druhá-věta) pravda)
        ((platí-první-a-druhá-ne? první-věta druhá-věta) lež)
        ((platí-druhá-a-první-ne? první-věta druhá-věta) lež)
        ((neplatí-ani-jedna? první-věta druhá-věta) lež)))
(define (souvětí-s-nebo první-věta druhá-věta)
  (cond ((platí-obě? první-věta druhá-věta) pravda)
        ((platí-první-a-druhá-ne? první-věta druhá-věta) pravda)
        ((platí-druhá-a-první-ne? první-věta druhá-věta) pravda)
        ((neplatí-ani-jedna? první-věta druhá-věta) lež)))
(define (souvětí-s-vylučovacím-nebo první-věta druhá-věta)
  (cond ((platí-obě? první-věta druhá-věta) lež)
        ((platí-první-a-druhá-ne? první-věta druhá-věta) pravda)
        ((platí-druhá-a-první-ne? první-věta druhá-věta) pravda)
        ((neplatí-ani-jedna? první-věta druhá-věta) lež)))
(define (když první-věta druhá-věta)
  (cond ((platí-obě? první-věta druhá-věta) pravda)
        ((platí-první-a-druhá-ne? první-věta druhá-věta) lež)
        ((platí-druhá-a-první-ne? první-věta druhá-věta) pravda)
        ((neplatí-ani-jedna? první-věta druhá-věta) pravda)))
(define (právě-tehdy-když první-věta druhá-věta)
  (cond ((platí-obě? první-věta druhá-věta) pravda)
        ((platí-první-a-druhá-ne? první-věta druhá-věta) lež)
        ((platí-druhá-a-první-ne? první-věta druhá-věta) lež)
        ((neplatí-ani-jedna? první-věta druhá-věta) pravda)))
```

; odborné názvy

```
(define konjunkce souvětí-s-a)
(define disjunkce souvětí-s-nebo)
(define negace neplatí-že)
(define implikace když)
(define ekvivalence právě-tehdy-když)
```

; Jaká je negace věty 'Když prší je mokro'?

```
(define prší pravda)
(define je-mokro pravda)
(řekni-jestli-platí-že
  (neplatí-že (když prší je-mokro))
  (když (neplatí-že prší) je-mokro))
; Když neprší je mokro.
(řekni-jestli-platí-že
  (neplatí-že (když prší je-mokro))
  (když prší (neplatí-že je-mokro)))
; Když prší není mokro.
(řekni-jestli-platí-že
  (neplatí-že (když prší je-mokro))
  (když (neplatí-že prší) (neplatí-že je-mokro)))
; Když neprší není mokro.
```