Scheme Primer

by Christine Lemmer-Webber of the Spritely Institute

Jiri Vlasak

2025-10-30

Úvod k Scheme Primer

Programovací jazyk Scheme

- LISP
  - LISt Processing
  - Lots of Irritating Superfluous Parentheses
- Prefix notace

Prefix	Infix
`(+ 2 3)`	`2 + 3`
`(fn arg1 arg2)`	`fn(arg1, arg2)`

- Mezi implementace Scheme patří:
  - Racket – programovací jazyk pro výzkum programovacích jazyků
  - Guile – programovací jazyk pro jednoduché rozšíření C/C++ projektů

Filosofie Scheme

Originál z R7RS

Programming languages should be designed not by piling feature on top of feature, but by removing the weaknesses and restrictions that make additional features appear necessary.

Neuměle přeloženo

Programovací jazyky by neměly být navrhovány přidáváním funkcí, ale odebíráním slabostí a omezení, která přidávání funkcí vynucují.

Scheme Primer

- https://files.spritely.institute/papers/scheme-primer.html
- Scheme Primer je dokument sloužící jako úvod do rodiny programovacích jazyků (implementací) Scheme.
- Firefox –> Print –> 34 stránek.
- Začíná programem *Hello World!* a končí implementací *Scheme v Scheme*.

Obsah prezentace

Zbytek prezentace kopíruje kapitoly Scheme Primer:

- Hello Scheme!
- Základní typy, příklad funkcí
- Proměnné a procedury
- Predikáty a podmínkové výrazy
- Seznamy (`list` a `cons`)
- Closures (uzávěry)
- Iterace a rekurze
- Změna hodnoty a přiřazení – vedlejší efekty
- O rozšiřitelnosti Scheme (a Lisp obecně)
- Scheme in Scheme

Scheme Primer

Hello Scheme!

```
(display "Hello World!")
```

Základní typy, příklad funkcí

Čísla

```
(+ 1 2 3) (+ 1 2 3.0)
(* 1 2 3) (* 1 2 3.0)
(/ 1 2 3) (/ 1 2 3.0)
```

Pravdivostní hodnoty

```
#t
#f
```

Text, symboly, procedury a seznamy

```
"Nějaký text"
'nejaky-symbol
(lambda (a) (+ a a)) ; anonymní funkce
'(lambda (a) (+ a a)) ; seznam (list)
```

Proměnné a procedury

Proměnné

```
(define slovo "slovo")
(string-append "Cituji: '" slovo "'.")
```

Procedury

```
(define (cituj co)
  (string-append "Cituji: '" co "'."))
```

Obsah obdélníku jako anonymní funkce

```
((lambda (sirka vyska)
   (* sirka vyska))
 10 3)
```

Predikáty a podmínkové výrazy

Predikáty

```
(string? "prší") (string? 123) (string? 'prsi)
(number? "prší") (number? 123) (number? 'prsi)
(symbol? "prší") (symbol? 123) (symbol? 'prsi)
```

Když if

```
(define A 3)
(define B 4)
(if (< A B)
  "B je větší."
  "A je větší.")
```

Predikáty a podmínkové výrazy

Když cond

```
(define co #t)
(cond
  ((string? co) "text")
  ((number? co) "číslo")
  ((symbol? co) "symbol")
  (else "něco jiného"))
```

Když cond v proceduře

```
(define (jaky-typ-ma co)
  (cond
    ((string? co) "text")
    ((number? co) "číslo")
    ((symbol? co) "symbol")
    (else "něco jiného")))
```

Seznamy (`list` a `cons`)

Seznam ...

```
(list 1 2 3 'co "prší")
```

... nebo ...

```
'(1 2 3 co "prší")
```

... je rekurzivní datová struktura ...

```
(cons 1 (cons 2 (cons 3 (cons 'co (cons "prší" '()))))))
```

... mající první prvek seznamu ...

```
(car (list 1 2 3 'co "prší"))
```

... a zbytek seznamu.

```
(cdr (list 1 2 3 'co "prší"))
```

Closures (uzávěry)

Přičítač je funkce

```
(define (pricti-1 k-cemu)
  (+ 1 k-cemu))
```

Vytváření přičítačů-uzávěrů

```
(define (vytvor-pricitac z-cisla)
  (define (pricitac k-cemu)
    (+ z-cisla k-cemu))
  pricitac)
```

Přičítač a anonymní funkce

```
(define (vytvor-pricitac z-cisla)
  (lambda (k-cemu)
    (+ z-cisla k-cemu)))
```

Iterace a rekurze

- Většinou pro zpracování seznamu, třeba:

```
(list "ředkvičky" "rýže" "minerálka")
```

- Počet písmen jednotlivých slov s map:

```
(map string-length (list "ředkvičky" "rýže" "minerálka"))
```

- Součet všech čísel seznamu s rekurzivní funkcí:

```
(define (secti-cisla seznam)
  (if (null? seznam)
      0
      (+ (car seznam) (secti-cisla (cdr seznam)))))

(secti-cisla ; tedy delky vsech slov seznamu
  (map string-length
       (list "ředkvičky" "rýže" "minerálka")))
```

Změna hodnoty a přiřazení – vedlejší efekty

```
(define dva 2)
(+ dva dva)
(set! dva 4)
(+ dva dva)
```

O rozšiřitelnosti Scheme (a Lisp obecně)

Uhm postfix notace: `(uhm 2 3 4 +)`

```
(define-syntax uhm
  (syntax-rules ()
    ((uhm v ... op)
     (op v ...))))
```

`nif` – nepravdivý `if`

```
(define-syntax nif
  (syntax-rules ()
    ((nif ne-podminka co-kdyz-ano co-kdyz-ne)
     (if (not ne-podminka)
         co-kdyz-ano
         co-kdyz-ne))))
```

Scheme in Scheme

Implementovaný v Guile, viz

https://files.spritely.institute/papers/scheme-primer.html#scheme-in-scheme

Shrnutí Scheme Primer

Shrnutí

```
(display "Hello World!")
(+ 1 2) (/ 1 2) (* 3.0 4.0) #t #f "Nějaký text"
(define p "proměnná") (define (fn p1 p2) (+ p1 p2))
(if (< A B) "B je větší." "A je větší.")
(cons (car (list 1 2 3)) (cdr (list 1 2 3)))
(define (mk-pricitac c) (lambda (k-cemu) (+ c k-cemu)))
(map string-length (list "ředkvičky" "rýže" "minerálka"))
(define dva 2) (+ dva dva) (set! dva 4) (+ dva dva)
(nif (> A B) "B je větší." "A je větší.")
```