

Базові примітиви строого функціональної мови програмування

Лекція 4

7 функцій

1. Селектори
2. Конструктор
3. Предикати
4. Умовна функція
5. Функція блокування

Виклик функції

здійснюється у формі списку.

Головою такого списку є ідентифікатор відомої функції.


Аргументи задаються через пробіл

(name arg1 arg2 ...),

де name - ім'я функції, arg1, arg2, ... - її аргументи.



Що може бути на місці `arg1,arg2,...`?



Аргументами функцій можуть бути S-вирази, які мають значення.

Тобто ті вирази, які можуть бути обчислені інтерпретатором функційної мови програмування.

Які саме?



Проте не завжди потрібно здійснювати обчислення на місці аргументів.

Тоді використовують функцію блокування обчислень QUOTE, яка позначається як одинарна відкриваюча лапка ' .

Аргумент цієї функції – будь-який S-вираз.

Значення цієї функції – це й же S-вираз.

Наприклад

- 'f
- f
- '(d f)
- (d f)
- '(d f (g . h))
- (d f (g . h))

Селектори

- це функції, що здійснюють вибір певних складових з складених S-виразів (list – класичний аргумент списку).

CAR вибрати голову списку.

CDR вибрати хвіст списку.

Стосовно атому ці функції невизначені.

Результатом функції (CAR list) завжди є перший елемент списку list.

Результатом функції (CDR list) є список list без першого елемента.

Наприклад

Welcome to DrScheme, version 301.

Language: Standard (R5RS).

```
>(car (d f g h)) //1
```

. reference to undefined identifier: d

```
>(car ' (d f g h)) //2
```

d

```
> (cdr ()) //3
```

. cdr: expects argument of type <pair>; given ()

> (cdr '(d . f)) **//4**

f

> (car '(c . v)) **//5**

c

> (car '()) **//6**

. car: expects argument of type <pair>; given ()

> (cdr '()) **//7**

. cdr: expects argument of type <pair>; given ()

> (cdr '(f)) **//8**

()

Наприклад

```
> (cdr '( d g 1 2))
```


```
(g 1 2)
```

```
> (cdr '((1 . 2) j k))
```

```
(j k)
```

```
> (car '((1 . 2) j k))
```

```
(1 . 2)
```



Послідовне використання функцій-селекторів дає можливість вибрати будь-який елемент списку.

Правило композиції для селекторів як формування нової функції

Дозволяється використовувати композицію функцій CAR та CDR.

Тобто на місці аргумента одного селектора є виклик іншого селектора.

Імена таких функцій починаються на С і закінчуються на R, а між ними знаходиться послідовність літер А та D (але не більше 4 літер, як правило в більшості реалізаціях Lisp). Ця послідовність вказує шлях обчислення.


Наприклад

```
> (car (cdr '(1 2 3 4)))
```

```
2
```

```
> (cadr '(1 2 3 4))
```

```
2
```




```
>(cдар '(1 2 3 4))
```

➤ ???

Обчислимо

```
(cadar '((1 2 3) a b c)
```

```
(car (cdr (car '((1 2 3) a b c)))))
```

Як доступитися до i -ого елемента списку
(верхній рівень) ?

Конструктор

CONS використовується для додання об'єкту, який задається першим аргументом, до списку, заданого другим аргументом. Тобто об'єкт, який додається, стає головою списку
(cons arg1 arg2)

Наприклад

```
> (cons 'a '(s))
```

```
(a s)
```

```
> (cons '(s) '(a))
```

```
((s) a)
```


У Scheme конструктор формує пари

Для цього другий аргумент потрібно задати атомом


```
>(cons '(a) 'a)  
((a) . a)
```

Предикати


Функції, які призначені для визначення чи їх аргументи володіють певними властивостями і як результат видають логічні константи, називаються предикатами




У строго функційному програмуванні один предикат має виявляти тотожність 2 своїх аргументів, інший – виявляти “природу” аргумента (простий, складений).



У Scheme є домовленість у назві
предикатів останнім символом давати ?



Функцією порівняння двох об'єктів є функція EQ (у Scheme - EQ?). Вона порівнює на тотожність значення першого та другого аргументів, які обов'язково повинні бути атомами, та повертає значення істини (T, #t) або хибності (NIL, #f).




У випадку аргументів списків, навіть у разі їх тотожності, функція повертає значення хибності.

Наприклад,

```
>(eq? '(g) '(g))
```


```
#f
```



При написанні програм мовою Lisp часто виникає запитання: чи є даний об'єкт простим/складеним?

У більшості функційних мов це питання вирішує предикат АТОМ, який має один аргумент. Ця функція повертає Т, якщо об'єкт є атомом і NIL в іншому випадку.

Порожній список NIL є атомом.



У мові Scheme предикат PAIR? перевіряє
навпаки – чи його аргумент не є атомом,
а предикат LIST? - чи його аргумент є
списком.

Наприклад

```
>list? '(d . f))
```

```
#f
```

```
> (pair? '(d f))
```

```
#t
```

```
>(pair? 'h)
```

```
#f
```

```
> (list? 'h)
```

```
#f
```

```
> (pair? h)
```

```
. reference to undefined identifier: h
```

```
>(pair? '(x . y))
```

```
#t >
```

Умовна функція

COND має змінну кількість параметрів.

Кожен параметр представляється списком типу $(e\ f)$, де e – предикатний вираз, а f – значення S-виразу.

$COND\ (e1\ f1)\ (e2\ f2)\\ (en\ fn).$

Схема обчислень умовної функції складається з таких дій:

- визначення значення e_1 , у випадку істинності вибирається значення як результат функції (обчислення завершено);
- у випадку хибності e_1 відбувається перехід до наступного аргументу і т.д;
- процес завершується обчисленням значенням вибраного виразу.

Якщо ж жоден з логічних виразів $e_1, e_2 \dots e_n$ не набув значення істина, результуюче значення функції COND не визначено.

У цьому випадку бажано останніми параметрами задавати вираз $(\#t f^*)$, де f^* - деяке визначене значення, якого набуває функція COND, якщо попередні усі вирази $e_1, e_2 \dots e_n$ були значення "неістина".

Наприклад

```
> (cond (#t 2) (#t 3))
```

```
2
```

```
> (cond (#f 2) (#f 3))
```

```
>
```