




# Функції вищого порядку

## Лекція 8

# Поняття функції

- 1) Функція є зображенням обчислень
- 2) Виклик функції є застосуванням цього зображення
- 3) Значення функції є результатом такого застосування



Функція – це правило перетворення даних з області визначення у дані з області значень.

Область визначення – це множина даних, при яких функція має значення.

Область значень – це множина значень, які може набувати функція.

# Терміни стосовно поняття функції


- аргумент = параметр
- значення = результат



У програмуванні розрізняють 2 сутності:  
дані та операції.

**Дані** – це пасивні складові  
обчислювального процесу, це матеріал,  
яким управляють.

**Операції** – це активні складові, які  
описують правила виконання дій.



У функціональному програмуванні  
закладено концепцію про непринципову  
відмінність між даними і операціями.  
Функції - елементи з найменшими  
обмеженнями.



# Важливі властивості функцій

- 1) На функції можна посилатися з допомогою символічного позначення
- 2) Функції можна включати в структури даних
- 3) Функції можна передавати як параметри
- 4) Функції можуть повертатися в якості результату інших функцій



# Класифікація функцій

стосовно природи аргументу та  
результату



# Функції

```
graph TD; A[Функції] --> B[Функції першого порядку]; A --> C[Функції вищих порядків]; B --> D[Аргумент та результат – звичайні дані]; C --> E[Аргумент і/або результат – функції];
```

**Функції  
першого  
порядку**

Аргумент та  
результат –  
звичайні  
дані

**Функції  
вищих  
порядків**

Аргумент  
і/або  
результат –  
функції


# Функції вищих порядків

```
graph TD; A[Функції вищих порядків] --> B[Звичайні функціонали]; A --> C[Функції з функціональним значенням]; A --> D[Функціонали з функціональним значенням];
```

Звичайні  
функціонали

Функції з  
функціональним  
значенням

Функціонали з  
функціональним  
значенням




Функціонал – це функція, яка має аргумент іншу функцію.

Цей аргумент наз. функціональним.

Звичайний функціонал – це функціонал, значення якого звичайне пасивне дане.

Функціонал з функціональним значенням – це функціонал, значенням якого є функція.

Тобто у функціонала з функціональним значенням і аргумент, і значення її функціями.



Функція з функціональним значенням – це функція, у якої аргумент – звичайне дане, а результат – функція.




# Пишемо функціонали

# Приклад 1


Нехай над списком числових даних потрібно проводити різні операції (збільшувати кожен елемент на 2, ділити на 3 і т.п.)

X – список чисел ( $x_1$   $x_2$  \_\_\_\_\_  $x_k$ )



```
(define f1 (lambda (x) (cond ((eq? x '()) ())  
  (#t (cons (+ (car x) 2) (f1(cdr x)))))))
```


```
(define f2 (lambda (x) (cond ((eq? x '()) ())  
  (#t (cons (/ (car x) 3) (f2(cdr x)))))))
```




Подібність визначень  $f1$  і  $f2$  та можливість побудови багатьох інших аналогічних функцій дозволяє припустити наявність деякої узагальненої функції.

У такій функції варто ввести параметр, що відповідатиме за конкретну операцію, що виконуватиметься над кожним елементом списку.





```
(define F (lambda (x op) (cond ((eq? x '()) ())  
  (#t (cons (op (car x)) (F (cdr x) op))))))
```



- $f1 = F(x, op1), op1(z) = z + 2$

- $f2 = F(x, op2), Op2(z) = z / 3$

`(define op1 (lambda (x) (+ x 2)))`

`(define op2 (lambda (x) (/ x 3)))`

# Можна задавати виклики

- $(F \text{ '(2 4 6 7) op1})$
- $(F \text{ '(2 4 6 7) op2})$

## Приклад 2

Нехай над списком числових даних потрібно проводити перетворення, яке зводить всі числа до одного.

$X$  – список чисел  $(x_1 \ x_2 \ \underline{\hspace{2cm}} \ x_k)$

# Редукція – спеціальна функція

має 3 аргументи:

$X$  – список чисел,

$G$  – бінарна функція (в сенсі двох аргументів),

$A$  – константа, яка приводить (редукує) список  $X$  до одного значення.

$G(x_1, G(x_2, G(x_3, \dots G(x_k, A) \dots)))$

# Функційне означення редукції

- Якщо  $X$  – порожній список, то результат  $A$ .
- Інакше проводити обчислення  
( $G$  ( $\text{car } X$ ) ( $\text{reduct } (\text{cdr } X) G A$ ))



# Аплікативні функціонали

APPLY застосовує функцію до списку параметрів.

Це функція 2 аргументів, з яких перший аргумент – функція, а другий – список аргументів.

# Приклад побудови функцій вищих порядків

$\lambda$  – нотація зручна форма запису правила перетворення, яка може використовуватися як на місці аргументу, так і на місці результату.



# $\lambda$ –вираз як аргумент

(suma '(2 4 5 8)) ->19

suma (x) = reduct (x,  $\lambda$  (z,y). z+y,0)

# $\lambda$ – вираз як результат

$\lambda(z) . z + y$

$\lambda(3) . 3 + y$  – функція додавання 3

$\lambda(2) . 2 + y$  – функція додавання 2

Тобто  $f_1(y) = \lambda(3)$

$f_2(y) = \lambda(2)$

А  $f_1(2)$  - ?

$f_2(7)$  - ?

# Функціонал з функціональним значенням


Визначаємо функцію композицію двох функцій

$$\text{Comp } (F,G) = \lambda(x). F(G(X)) ,$$

F,G – функціональні аргументи, а результат “добуток” цих функцій, тобто нова функція.

# Наприклад

$F(X) = \text{Comp}(f1, \text{reverse})$  – функція, яка обертає список  $X$ , а потім збільшує кожен елемент на 1.



```
(define Comp (lambda (F G) (lambda arg  
(f (apply g arg))))))
```

```
( (Comp sqrt +) 5 2 4 2 3) -> 4
```