関数を引数で渡す (Applicative Programming)

- 関数を引数で渡す
  (funcall #'cons 'a 'b) ⇒ (a . b)
- 関数を引用するには #'
- > (setf fn #'cons) ⇒
  > fn ⇒
  > (type-of fn) ⇒
  > (funcall fn 'c 'd) ⇒

62

mapcar

- (mapcar #'square '(2 3 4)) ⇒ (4 9 16)
  ただし (defun square (n) (* n n)) と定義しておく
- mapcarはリストの要素数を保つ
  (mapcar #'square '( )) ⇒ nil
- 関数定義を埋め込む
  (mapcar #'(lambda (n) (* n n)) '(2 3 4))

63

- (mapcar #'+ '(1 2 3) '(10 20 30 40))
  ⇒ (11 22 33)
- (mapcar #'cons '(1 2) '(3 4))
  ⇒ ((1 . 3) (2 . 4))
- (mapcar #'(lambda (x y) (list x 'gets y))
  '(Ken Kaoru Yumi)
  '(job1 job2 job3))
  ⇒ ((KEN GETS JOB1)
  (KAORU GETS JOB2)
  (YUMI GETS JOB3))

64

remove-if と remove-if-not

- remove-ifはリストの各要素に述語を適用し，真となった要素を除いたリストを返す.
- (remove-if #'numberp '(2 for 1)) ⇒ (for)
  (remove-if #'(lambda (x) (not (plusp x)))
  '(2 0 -4 6 -8 10)) ⇒ (2 6 10)
- (remove-if-not #'plusp '(2 0 -4 6 -8 10))
  ⇒ (2 6 10)
  (remove-if-not #'(lambda (x) (> x 3))
  '(2 4 6 8 4 2 1)) ⇒ (4 6 8 4)

65

find-if

- リストに述語を適用していき，最初に真となった要素を返す.
- (find-if #'oddp '(2 4 6 7 8)) ⇒ 7
  (find-if #'(lambda (x) (> x 3))) '(2 4 6))
  ⇒ 4

66

クロージァ

○ (defun my-assoc (key table)
  (find-if #'(lambda (entry)
  (equal key (car entry))) table))
ラムダ式の中のkeyはラムダ式の引数リストに入っていない。これを自由変数という。自由変数を含むラムダ式をクロージャという。上の関数を2つに分けて下のようにするとkeyが何を指すか分からず、エラーとなる。

×(defun helper (entry)
  (equal key (car entry)))
(defun faulty-assoc (key table)
  (find-if #'helper table))

67

関数を作る関数

- 入力がある数Nより大きい時は真を返す関数

```
(defun  mk-greater-n-p  (n)
        #'(lambda  (x)  (>  x  n)))
>  (setf  pred3  (mk-greater-n-p  3))
  ⇒  #<lexical-closure  7314225>
> (funcall  pred3  2)  ⇒
> (funcall  pred3  5)  ⇒
```

68

プリント5