第8回
設計の抽象化

情報処理演習II

Literacy

例題

Literacy

例題1. map 関数

- map 関数を Scheme インタプリタ (DrScheme) で実行し，実行結果を確認する
  - map 関数は，DrScheme に組み込み済みの関数である

入力は関数とリスト(1つあるいは複数)

出力は1つのリスト

Literacy

map関数の実行例1

Lit

map関数の使用例(1)

```
(map sqr (list 1 2 3 4))
```

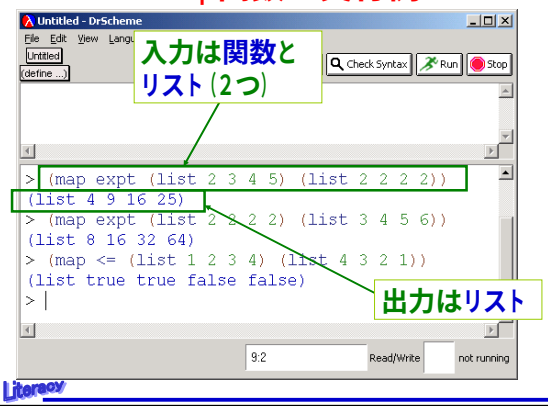
```
(list 1 2 3 4)
```

sqr 関数を適用

```
(list 1 4 9 16)
```

Literacy

map関数の実行例2



Literacy

map関数の使用例(2)

```
(map expt
     (list 2 3 4 5) (list 2 2 2 2))
```

(list 2 3 4 5) (list 2 2 2 2)

expt 関数を適用

(list 4 9 16 25)

型変数

map 関数の Contract(規約)

```
;; map: (X ... -> Y)  (listof X)  ... -> (listof Y)
```

関数
入力:データタイプ X から始まる並び
出力:データタイプ Y

Xのリスト

続けて リストが来る こともありえる

Yのリスト

出力

入力

X, Y は型変数

データ型 (number, string, boolean, struct, list などなど) があてはまる

map関数の実行時に起こるエラーの例

入力として，関数とリスト(2つ)を与えなければならない

入力として与えるべきリストの数が正しくない

```
> (map expt (list 1 2 3 4) 2)
map: expects type <list> as 3rd argument, given: 2
other arguments were: expt (list 1 2 3 4)
> (map < (list 1 2))
map: arity mismatch for <: expects at least 2
arguments, given 1
> (map < (list 1 2) (list 4 5 3))
map: all lists must have same size; arguments were: <
(list 1 2) (list 4 5 3)
>
```

入力として与えたリストの要素数は一致していなければならない

例題2．series 関数

- 数列の和を求める関数 `series` を定義し，実行する
  - 自然数 `n` と関数 `f` から，`(f 0)`, `(f 1)`, … `(f n)` の和を求める

つまり $\sum_{k=0}^{n} f(k)$

3 f → series → (f 0) (f 1) (f 2) (f 3) の和 (数値)

入力は自然数と関数

出力は数値

数列の和

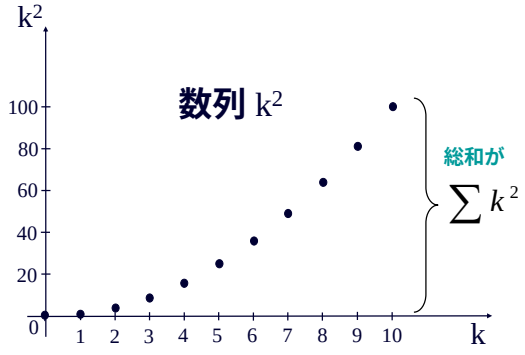
- n = 0 のとき

$$\sum_{k=0}^{n} f(k) = f(0)$$

- n > 0 のとき

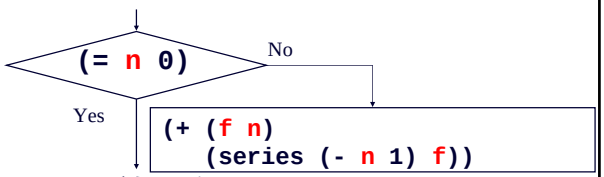
$$\sum_{k=0}^{n} f(k) = f(n) + \sum_{k=0}^{n-1} f(k)$$

数列の和



例題2．series 関数

1. 入力 n = 0 ならば：　　(終了条件)
   解は (f n)　　(自明な解)
2. そうでなければ：
   – (n - 1)項までの数列の和を求め，それと(f n)を足したものが求める解



(f 0)が自明な解

Literacy

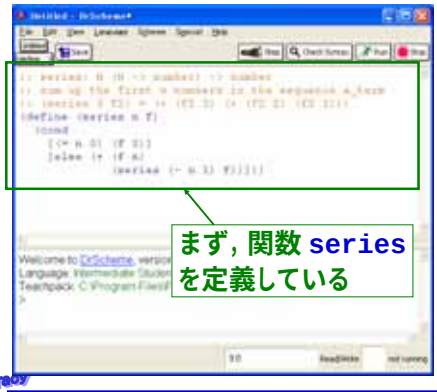
例題2．series 関数

```
;; series: N (N -> number) -> number
;; sum up the first n numbers in the sequence a_term
;; (series 3 f2) = (+ (f2 3) (+ (f2 2) (f2 1)))
(define (series n f)
  (cond 終了条件　自明な解
    [(= n 0) (f 0)]
    [else (+ (f n)
             (series (- n 1) f))]))
```

series の内部に series (再帰による繰り返し)
例：(series 3 f)
= (+ (f 3) (series 2 f))

Literacy

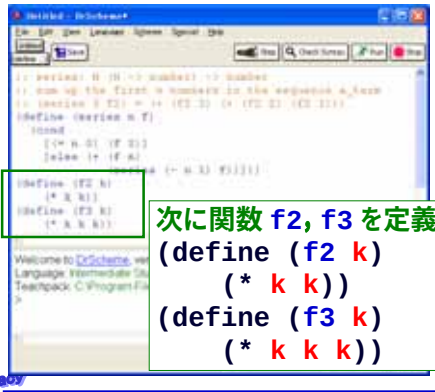
series関数の実行例(1/3)



まず，関数 series を定義している

Literacy

series関数の実行例(2/3)

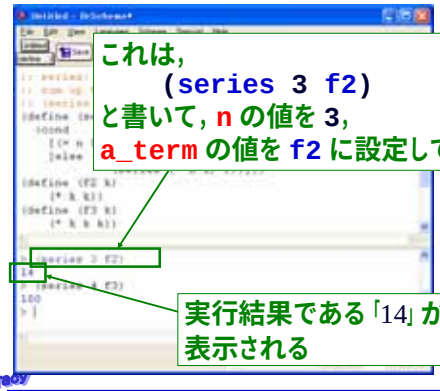


次に関数 f2，f3 を定義している

```
(define (f2 k)
    (* k k))
(define (f3 k)
    (* k k k))
```

Literacy

series関数の実行例(3/3)



これは，
(series 3 f2)
と書いて，n の値を 3，
a_term の値を f2 に設定して実行

実行結果である「14」が表示される

Literacy

例題3．project 関数

- リストのリスト data と，リストを入入力とする関数 f から，リストを求める関数 project を定義し，実行する

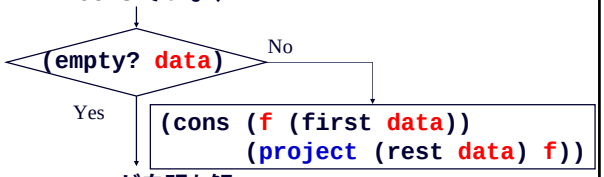
入力はリストのリストと関数

出力はリスト

Literacy

例題3．project 関数

1. 入力 data が空ならば： (終了条件)
   解は empty (自明な解)
2. そうでなければ：
   – data の first 部に f を適用したものと，data の rest 部と f に project を適用したものとを cons でつなぐ



empty が自明な解

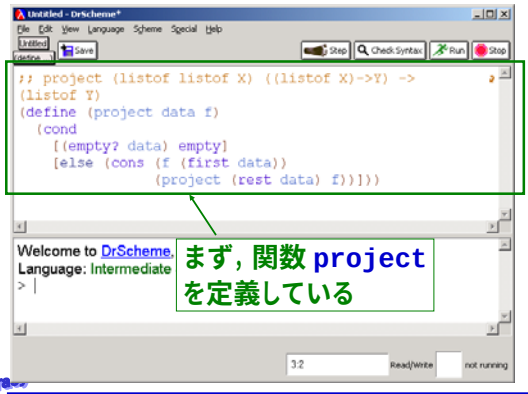
例題3．project 関数

```
;; project (listof listof X) ((listof X)->Y)
;;         -> (listof Y)
(define (project data f)
  (cond     終了条件    自明な解
    [(empty? data) empty]
    [else (cons (f (first data))
                (project (rest data) f))]))
```

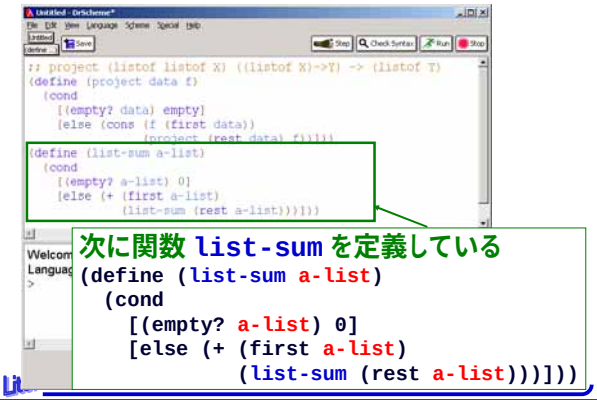
project の内部に project (再帰による繰り返し)

例：
```
(project (list (list 1 2) (list 3 4 5)) f)
  = (cons (f (list 1 2))
          (project (list (list 3 4 5)) f))
```

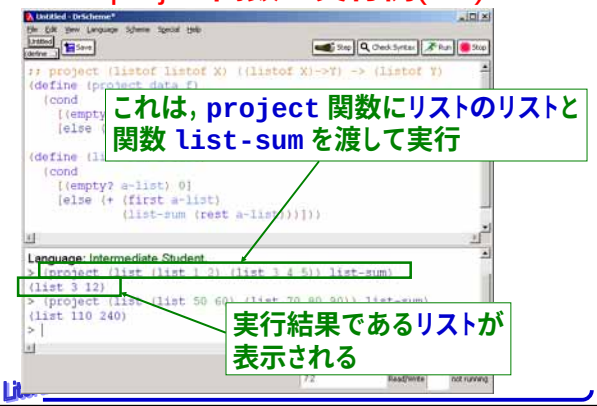
project関数の実行例(1/3)



project関数の実行例(2/3)



project関数の実行例(3/3)



型変数

project 関数の Contract(規約)

```
;; project (listof listof X) ((listof X)->Y) -> (listof Y)
```

Xのリストのリスト　関数 入力：Xのリスト 出力：Y　Yのリスト

入力　出力

X，Y は型変数

実習

Literacy

実習の進め方

- 資料を見ながら，「実習」を行ってみる
- その後，各自「課題」に挑戦する
  - 各自で自習 ＋ 巡回指導
  - 遠慮なく質問してください
- 自分のペースで先に進んで構いません

Literacy

DrScheme の使用

- DrScheme の起動
  プログラム → PLT Scheme → DrScheme
- 今日の実習では「Intermediate Student」に設定
  Language
  → Choose Language
  → Intermediate Student
  → OK ボタン

Literacy

DrScheme の使用 / ステップ実行

- プログラム実行の振る舞いを観察するツール
- 「定義用ウインドウ」のみを使用
  （通常の実行と異なる）
- 「Intermediate Student」に設定する必要あり
  Language → Choose Language
  → Intermediate Student
  → Run ボタン

Literacy

実習1．map 関数

- map 関数を Scheme インタプリタ（DrScheme）で実行し，実行結果を確認する
  - map 関数は，DrScheme に組み込み済みの関数である

入力は関数とリスト（1つあるいは複数）

出力は1つのリスト

Literacy

実習1．map 関数

1. 次を「定義用ウインドウ」で，実行しなさい
   ・ 入力した後に，Run ボタンを押す

```
(define (C2F c)
    (+ 32 (* 9/5 c)))
(define-struct IR (name price))
```

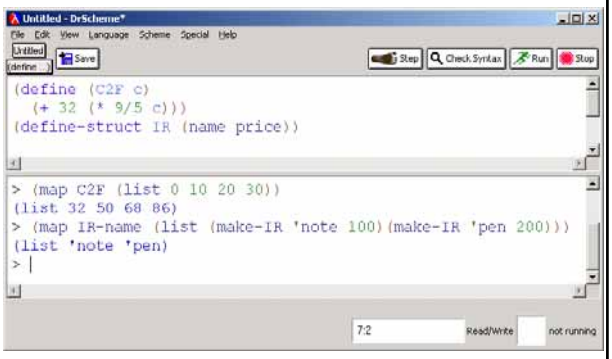
2. その後，次を「実行用ウインドウ」で実行しなさい

```
(map C2F (list 0 10 20 30))
```

```
(map IR-name (list (make-IR 'note
100)(make-IR 'pen 200)))
```

Literacy

map関数の実行例



Literacy

実習2．series 関数

- 数列の和を求める関数 `series` を定義し，実行する
  - 自然数 `n` と関数 `f` から，`(f 0)`，`(f 1)`，… `(f n)` の和を求める

つまり $\sum_{k=0}^{n} f(k)$

`3 f` ⇒ `series` ⇒ `(f 0) (f 1) (f 2) (f 3)` の和（数値）

入力は自然数と関数

出力は数値

Literacy

「実習2．series 関数」の手順

1. 次を「定義用ウインドウ」で，実行しなさい
   - 入力した後に，Run ボタンを押す

```
;; series : N (N -> number) -> number
;; sum up the first n numbers in the sequence a_term
;; (series 3 f2) = (+ (f2 3) (+ (f2 2) (f2 1)))
(define (series n f)
    (cond
        [(= n 1) (f 1)]
        [else (+ (f n) (series (- n 1) f))]))
(define (f3 n)
    (* n n n))
```

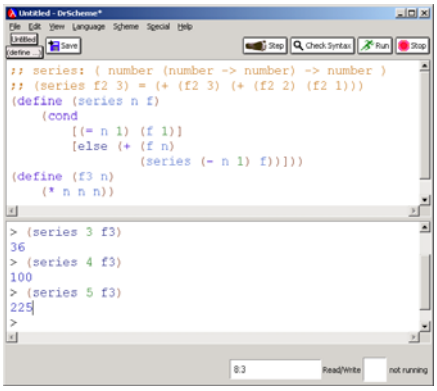
Contract 中で「N」とあるのは自然数の意味

2. その後，次を「実行用ウインドウ」で実行しなさい

```
(series 3 f3)
(series 4 f3)
(series 5 f3)
```

Literacy

series関数の実行例



Literacy

実習2b．ステップ実行

- 関数 `series` (実習2)について，実行結果に至る過程を見る
  - `(series 3 f3)` から 実行結果 36 に至る過程を見る
  - DrScheme のステップ実行機能を使用する

Literacy

「実習2b．ステップ実行」の手順

1. 次を「定義用ウインドウ」で，実行しなさい
   - Intermediate Student で実行すること
   - 入力した後に，Run ボタンを押す

実習2と同じ

```
;; series : N (N -> number) -> number
;; sum up the first n numbers in the sequence a_term
;; (series 3 f2) = (+ (f2 3) (+ (f2 2) (f2 1)))
(define (series n f)
    (cond
        [(= n 1) (f 1)]
        [else (+ (f n) (series (- n 1) f))]))
(define (f3 n)
    (* n n n))
(series 3 f3)
```

2. DrScheme を使って，ステップ実行の様子を確認しなさい　(Step ボタン，Next ボタンを使用)
   - 理解しながら進むこと

Literacy

実習3．filter1 関数

- 関数 **filter1** を定義し，実行する
  - リストの要素がある数より小さいか(大きいか)でふるいにかける
  - 小さいか(大きいか)を表す関数を入力する
  - 数値のリストを入力する
  - 基準となる数値(ある数)を入力する

入力は
関数，数値のリスト，数値

出力は
数値のリスト

Literacy

「実習3．filter1 関数」の手順

1. 次を「定義用ウインドウ」で，実行しなさい
   - 入力した後に，Run ボタンを押す

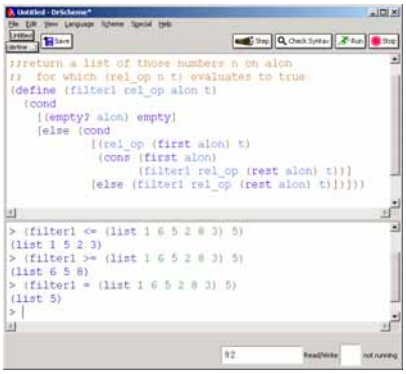
```
;;filter1: (number number -> boolean) lon number -> lon
;;return a list of those numbers n on alon
;;   for which (rel_op n t) evaluates to true
(define (filter1 rel_op alon t)
  (cond
    [(empty? alon) empty]
    [else (cond
            [(rel_op (first alon) t)
             (cons (first alon)
                   (filter1 rel_op (rest alon) t))]
            [else (filter1 rel_op (rest alon) t)])]))
```

2. その後，次を「実行用ウインドウ」で実行しなさい

```
(filter1 <= (list 1 6 5 2 8 3) 5)
(filter1 >= (list 1 6 5 2 8 3) 5)
(filter1 = (list 1 6 5 2 8 3) 5)
```

Literacy

filter1関数の実行例



Literacy

実習3b．ステップ実行

- 関数 **filter1** (実習3)について，実行結果に至る過程を見る
  - (filter1 <= (list 1 6 5 2 8 3) 5) から 実行結果 (list 1 5 2 3) に至る過程を見る
  - DrScheme のステップ実行機能を使用する

Literacy

「実習3b．ステップ実行」の手順

1. 次を「定義用ウインドウ」で，実行しなさい
   - Intermediate Student で実行すること
   - 入力した後に，Run ボタンを押す

実習3と同じ

```
;;filter1: (number number -> boolean) lon number -> lon
;;return a list of those numbers n on alon
;;   for which (rel_op n t) evaluates to true
(define (filter1 rel_op alon t)
  (cond
    [(empty? alon) empty]
    [else (cond
            [(rel_op (first alon) t)
             (cons (first alon)
                   (filter1 rel_op (rest alon) t))]
            [else (filter1 rel_op (rest alon) t)])]))
(filter1 <= (list 1 6 5 2 8 3) 5)
```

2. DrScheme を使って，ステップ実行の様子を確認しなさい (Step ボタン，Next ボタンを使用)
   - 理解しながら進むこと

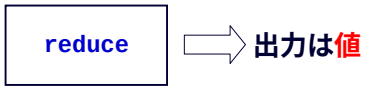
Literacy

実習4．reduce 関数

- 高階関数 **reduce** を定義する
  - さらに，数のリスト a-list から総和を求める関数 list-sum を定義し，実行する

```
;; reduce (X Y->Y) Y (listof X) -> Y
;; (reduce g 0 (list 1 2 3)) = (g 1 (g 2 (g 3 0)))
(define (reduce op base L)
  (cond
    [(empty? L) base]
    [else (op (first L)
              (reduce op base (rest L)))]))
```

入力は
関数，値，リスト



出力は値

Literacy

「実習4. reduce 関数」の手順

1. 次を「定義用ウインドウ」で，実行しなさい
   - 入力した後に，Run ボタンを押す

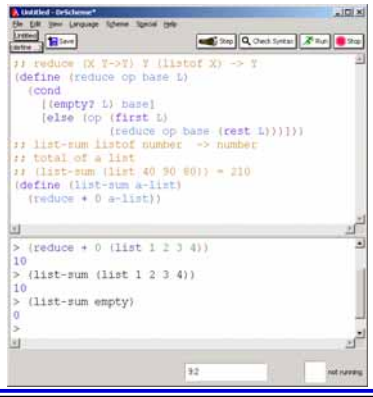
```
;; reduce (X Y->Y) Y (listof X) -> Y
;; (reduce g 0 (list 1 2 3)) = (g 1 (g 2 (g 3 0)))
(define (reduce op base L)
  (cond
    [(empty? L) base]
    [else (op (first L)
              (reduce op base (rest L)))]))
;; list-sum listof number  -> number
;; total of a list
;; (list-sum (list 40 90 80)) = 210
(define (list-sum a-list)
    (reduce + 0 a-list))
```

2. その後，次を「実行用ウインドウ」で実行しなさい

```
(reduce + 0 (list 1 2 3 4))
(list-sum (list 1 2 3 4))
(list-sum empty)
```

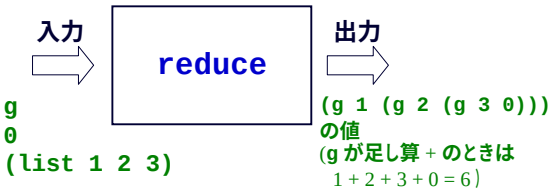
Literacy

reduce関数の実行例



Literacy

入力と出力



g
0
(list 1 2 3)

(g 1 (g 2 (g 3 0)))
の値
(g が足し算 + のときは
1 + 2 + 3 + 0 = 6)

入力はリスト，数値と関数

出力は数値

reduce は，関数を入力とするような関数
(つまり高階関数)

Literacy

実習4b. ステップ実行

- 関数 reduce (実習4)について，実行結果に至る過程を見る
  - (list-sum (list 1 2 3)) から 実行結果 6 に至る過程を見る
  - DrScheme のステップ実行機能を使用する

Literacy

「実習4b. ステップ実行」の手順

1. 次を「定義用ウインドウ」で，実行しなさい
   - Intermediate Student で実行すること
   - 入力した後に，Run ボタンを押す

実習4と同じ

```
;; reduce (X Y->Y) Y (listof X) -> Y
(define (reduce op base L)
  (cond
    [(empty? L) base]
    [else (op (first L)
              (reduce op base (rest L)))]))
;; list-sum listof number  -> number
;; total of a list
;; (list-sum (list 40 90 80)) = 210
(define (list-sum a-list)
    (reduce + 0 a-list))
(list-sum (list 1 2 3))
```

2. DrScheme を使って，ステップ実行の様子を確認しなさい (Step ボタン，Next ボタンを使用)
   - 理解しながら進むこと

Literacy

(list-sum (list 1 2 3)) から 6 に至る過程の概略

```
(list-sum (list 1 2 3))   最初の式
= (reduce + 0 (list 1 2 3))
= ...
= (+ 1 (reduce + 0 (list 2 3)))
= ...
= (+ 1 (+ 2 (reduce + 0 (list 3))))
= ...
= (+ 1 (+ 2 (+ 3 (reduce + 0 empty))))
= ...
= (+ 1 (+ 2 (+ 3 0)))
= ...                          コンピュータ内部での計算
= 6   実行結果
```

Literacy

(reduce + 0 (list 1 2 3) から
(+ 1 (reduce + 0 (list 2 3) に至る過程

```
(list-sum (list 1 2 3))
= (reduce + 0 (list 1 2 3))
= ...
= (+ 1 (reduce + 0 (list 2 3)))
= ...
= (+ 1 (+ 2 (reduce + 0 (list 3))))
= ...
= (+ 1 (+ 2 (+ 3 (reduce + 0 nil))))
= ...
= (+ 1 (+ 2 (+ 3 0)))
= ...
= 6
```

この部分は

```
(reduce + 0 (list 1 2 3))
= (cond
    [(empty? (list 1 2 3)) 0]
    [else (+ (first (list 1 2 3))
             (reduce + 0 (rest (list 1 2 3))))])
= (cond
    [false 0]
    [else (+ (first (list 1 2 3))
             (reduce + 0 (rest (list 1 2 3))))])
= (+ (first (list 1 2 3))
     (reduce + 0 (rest (list 1 2 3))))
= (+ 1 (reduce + 0 (rest (list 1 2 3))))
= (+ 1 (reduce + 0 (list 1 2)))
```

Literacy

(reduce + 0 (list 1 2 3) から
(+ 1 (reduce + 0 (list 2 3) に至る過程

```
(list-sum (list 1 2 3))
= (reduce + 0 (list 1 2 3))
= ...
= (+ 1 (reduce + 0 (list 2 3)))
= ...
= (+ 1 (+ 2 (reduc
```

この部分は

```
(reduce + 0 (list 1 2 3))
= (cond
    [(empty? (list 1 2 3)) 0]
    [else (+ (first (list 1 2 3))
             (reduce + 0 (rest (list 1 2 3))))])
= (cond
    [false 0]
```

これは，

```
(define (reduce op base L)
  (cond
    [(empty? L) base]
    [else (op (first L)
              (reduce op base (rest L)))]))
```

の op を + で，base を 0 で，L を (list 1 2 3) で置き換えたもの

Literacy

課題

Literacy

課題1．filter1 関数に関する問題

x > c が成立するかをテストする関数 is_large? を定義して，

```
;; is_large?: number number -> boolean
(define (is_large? x c)  (> x c))
```

下のように filter1 関数の入力として渡すと，「リスト中で，3 より大きい数を抽出する」ことができる

```
(filter1 is_large? (list 1 2 3 4 5) 3)
```

上と同様の手順で「リスト中で，値が3よりも大きい偶数を抽出する」ためには，どのような関数を定義して is_large? 関数の代わりにすればよいか．また，実行結果を報告しなさい．

Literacy

課題2．map 関数に関する問題

数のリストから数のリストを生成

リストの要素を全て10倍する関数 f を作りなさい

- 例えば，
  (f (list 1 2 3)) = (list 10 20 30)
- map 関数を使いなさい

入力は数値のリスト　　出力は数値のリスト

Literacy

課題3．抽象関数としての reduce (1)

reduce 関数を使い，数値のリストから，その要素の全ての積を求めるにはどうしたらよいか？

(list 2 5 7) から 70 を求めたい

Literacy

課題4．抽象関数としての reduce (2)

reduce 関数を使い，正の数値のリストから，その最大値を求める関数 f を定義し，実行結果を報告しなさい

- 例えば，(f (list 1 2 4)) = 4
- reduce 関数を使っていない場合は，正解とは認めない
- ここでは，問題を簡単にするため，次のように定める
  - 入力は，正の数のリストである
  - (f empty) = 0 とする

Literacy

課題4のヒント

```
(f (list 1 2 4))
= (reduce max 0 (list 1 2 4))
= ...
= (max 1 (reduce max 0 (list 2 4)))
= ...
= (max 1 (max 2 (reduce max 0 (list 4))))
= ...
= (max 1 (max 2 (max 4 (reduce max 0 empty))))
= ...
= (max 1 (max 2 (max 4 0)))
= ...
= 4
```

Literacy

課題5．抽象関数としての reduce (3)

reduce 関数を使い，数値のリストから，その最大値を求める関数 f を定義し，実行結果を報告しなさい

- 例えば，(f (list 1 2 4)) = 4
  (f (list -1 -2 -3)) = -1
- reduce 関数を使っていない場合は，正解とは認めない
- 下記に示す max2 関数を使用しなさい

```
;; max2: Anything Anything -> number or 'None
(define (max2 a b)
  (cond
    [(and (not (number? a)) (not (number? b))) 'None]
    [(and (not (number? a)) (number? b)) b]
    [(and (number? a) (not (number? b))) a]
    [else (max a b)]))
```

Literacy