sp-7. リストの生成

（Scheme プログラミング）

URL: https://www.kkaneko.jp/pro/scheme/index.html

金子邦彦

アウトライン

7-1 リストの生成

　リストを「出力」とするような関数

　cons を使用

　リストのリスト(リストの入れ子）の生成

7-2 パソコン演習

7-3 課題

7-1 リストの生成

リストの表記

- cons による表記

  例） (cons 'x (cons 'y (cons 'z

  empty)))

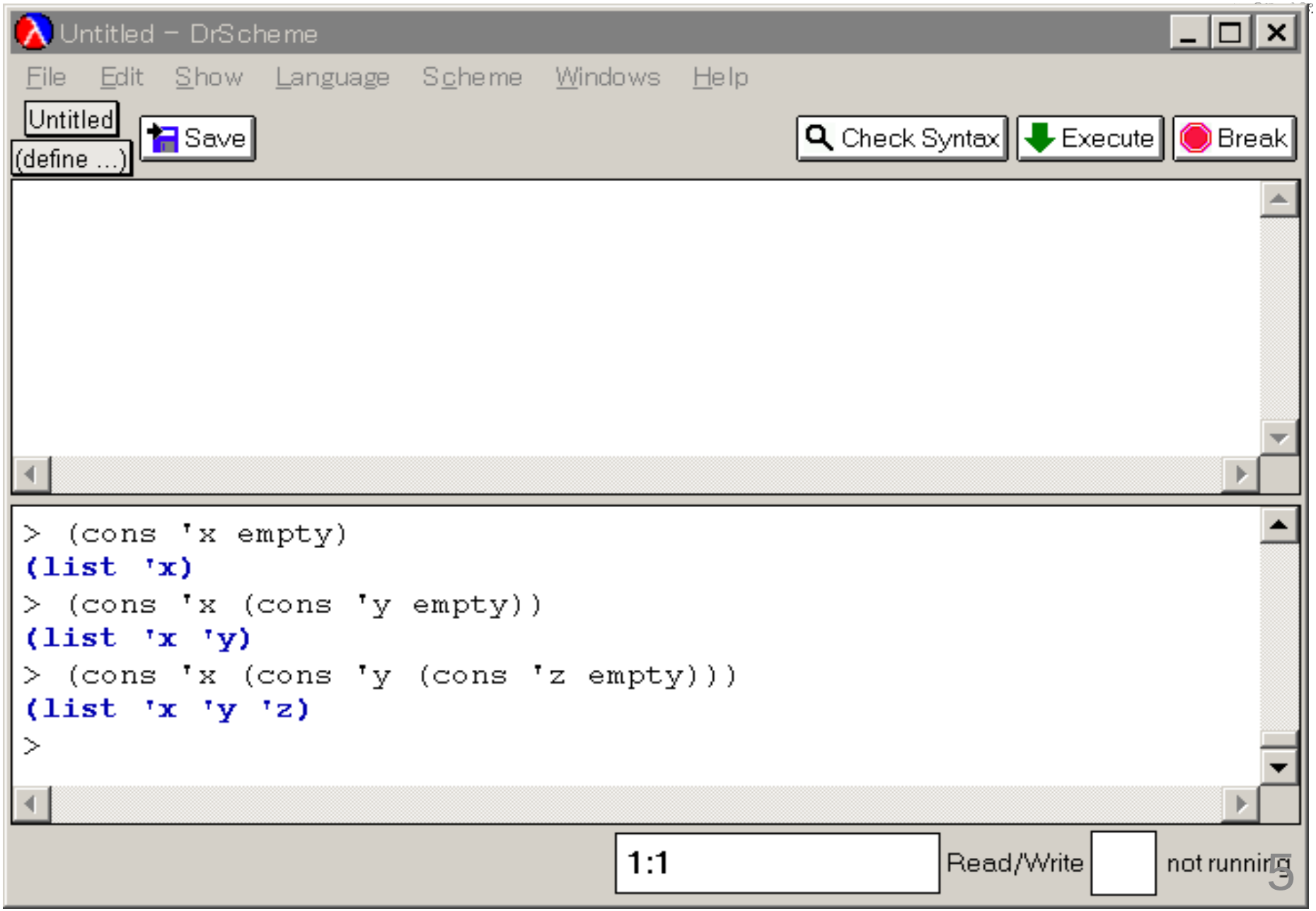
- list による表記

  例） (list 'x 'y 'y)

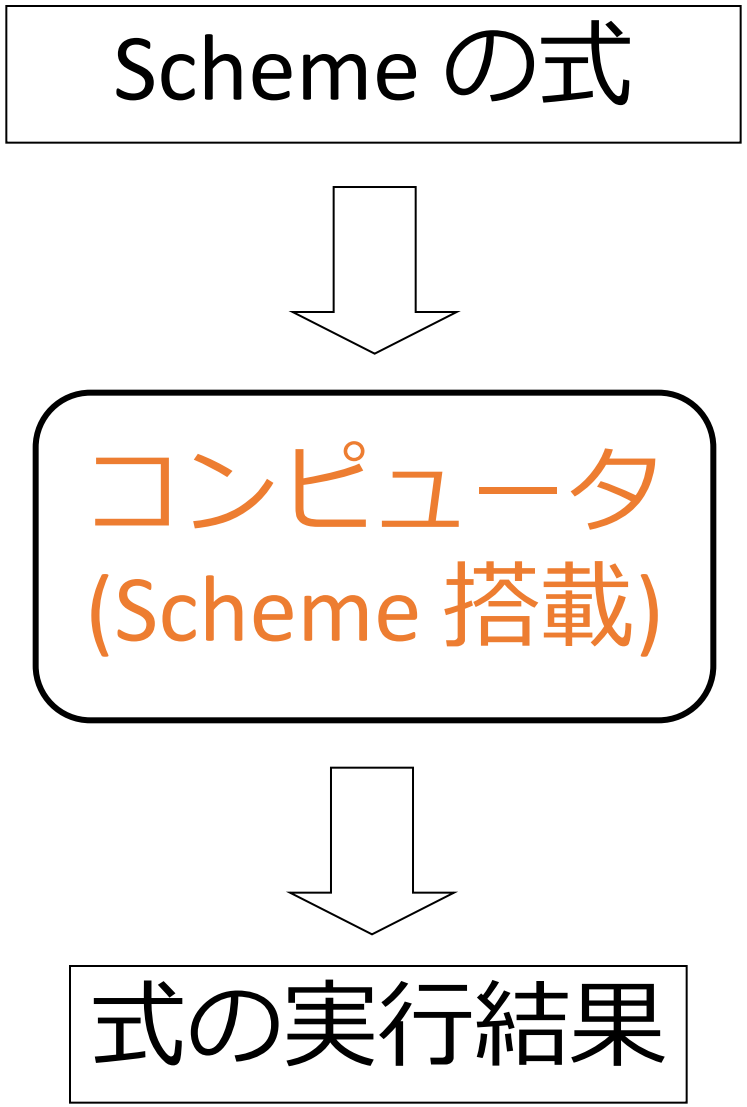
「empty」は空リストの意味であったが，ここでは，末尾の意味になる

２種類の表記がある

実行結果の例


Untitled – DrScheme
File Edit Show Language Scheme Windows Help
Untitled
(define ...)
Save
Check Syntax
Execute
Break
> (cons 'x empty)
(list 'x)
> (cons 'x (cons 'y empty))
(list 'x 'y)
> (cons 'x (cons 'y (cons 'z empty)))
(list 'x 'y 'z)
>
1:1
Read/Write
not running

コンピュータが行っていること



例えば：

(cons 'x (cons 'y empty))

を入力すると・・・

(list 'x 'y)

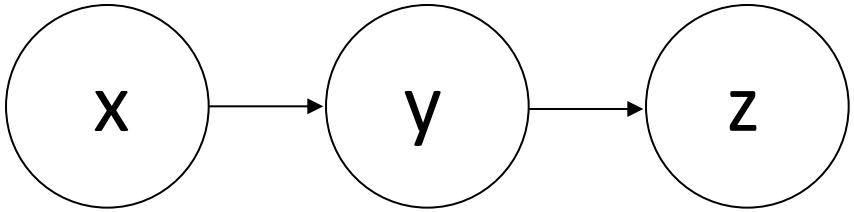
が表示される

cons の実行例

```
(cons 'x empty)
  →   (list 'x)
(cons 'x (cons 'y empty))
  →   (list 'x 'y)
(cons 'x (cons 'y (cons 'z empty)))
  →   (list 'x 'y 'z)
```

cons の意味

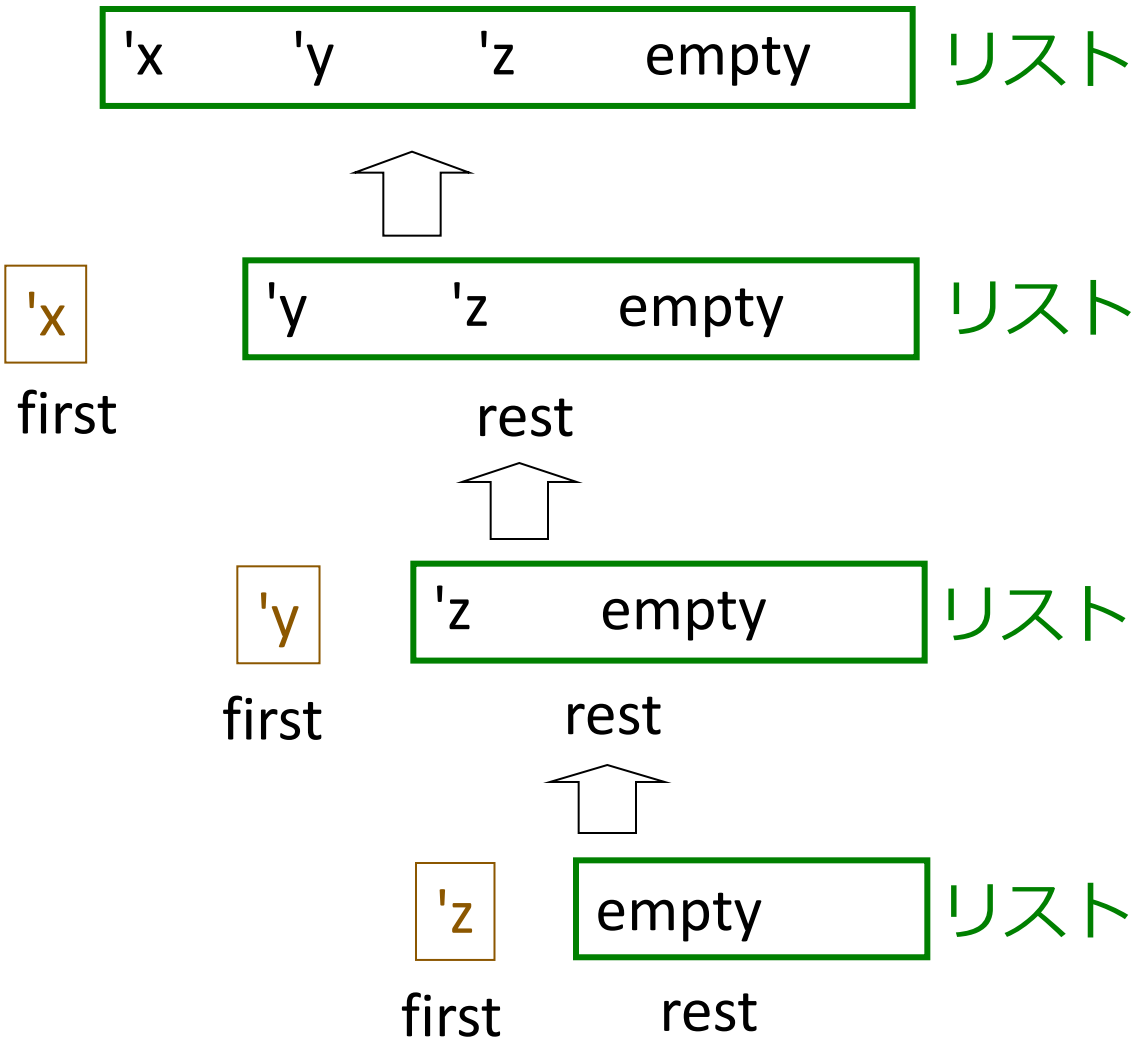
```
(cons 'x (cons 'y (cons 'z empty)))
```



x, y, z のリスト

cons の意味

cons は，リストの first と rest をつなげて，リストを作る



リストと cons の関係

1. 空リスト
   - empty
2. 長さ１以上のリスト
   - list 形式:　　例 (list 12 24 26)
   - cons 形式：　例 (cons 12 (cons 24 (cons 36 empty)))

この２つは同じ意味

Scheme の式の構成物

- 数値：
  5, -5, 0.5 など
- true, false 値
  true, false
- シンボル，文字列
- 変数名
- empty
- 四則演算子：
  +, -, *, /
- 比較演算子
  <, <=, >, >=, =
- 奇数か偶数かの判定
  odd?, even?
- 論理演算子
  and, or, not
- リストに関する演算子
  first, rest, empty?, length, list-ref,
  append, cons
- その他の演算子：
  remainder, quotient, max, min,
  abs, sqrt, expt, log, sin, cos, tan
  asin, acos, atan など

- 括弧
  (,), [,]
- 関数名
- define
- cond
- list

7-2 パソコン演習

パソコン演習の進め方

- 資料を見ながら，「例題」を行ってみる
- 各自，「課題」に挑戦する
  - 各自で自習　＋　巡回指導
- 自分のペースで先に進んで構いません

DrScheme の使用

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

例題１．２次方程式

- ２次方程式 $ax^2 + bx + c = 0$ の解を求める関数 quadratic-roots を作り，実行する
  - 解を「リスト」として出力する
  - 重解を求める
  - 但し，虚数解は考えない
  - a=0 の場合も考えない

参考 Webページ：
http://www.htdp.org/2001-11-21/Book/node57.htm: Exercise 10.1.8

- 一変数 x の二次方程式の一般形:

  $ax^2 + bx + c = 0$

- 二次方程式の解の数:

  a, b, cの値に依存

  (1) a = 0 ⇒ 方程式は degenerate

  (2) a ≠ 0 ⇒ proper な二次方程式

  1. もし $b^2 > 4ac$ なら　　二つの解
  2. もし $b^2 = 4ac$ なら　一つの解
  3. もし $b^2 < 4ac$ なら　解無し

判別式 $D = b^2 - 4ac$ とする

1) $D > 0$ のとき

$$x = \frac{-b + \sqrt{D}}{2a}, \frac{-b - \sqrt{D}}{2a}$$

異なる２実数解

2) $D = 0$ のとき

$$x = -\frac{b}{2a},$$

重解（解の個数は１）

3) $D < 0$ のとき

解なし

「例題１．２次方程式」の手順

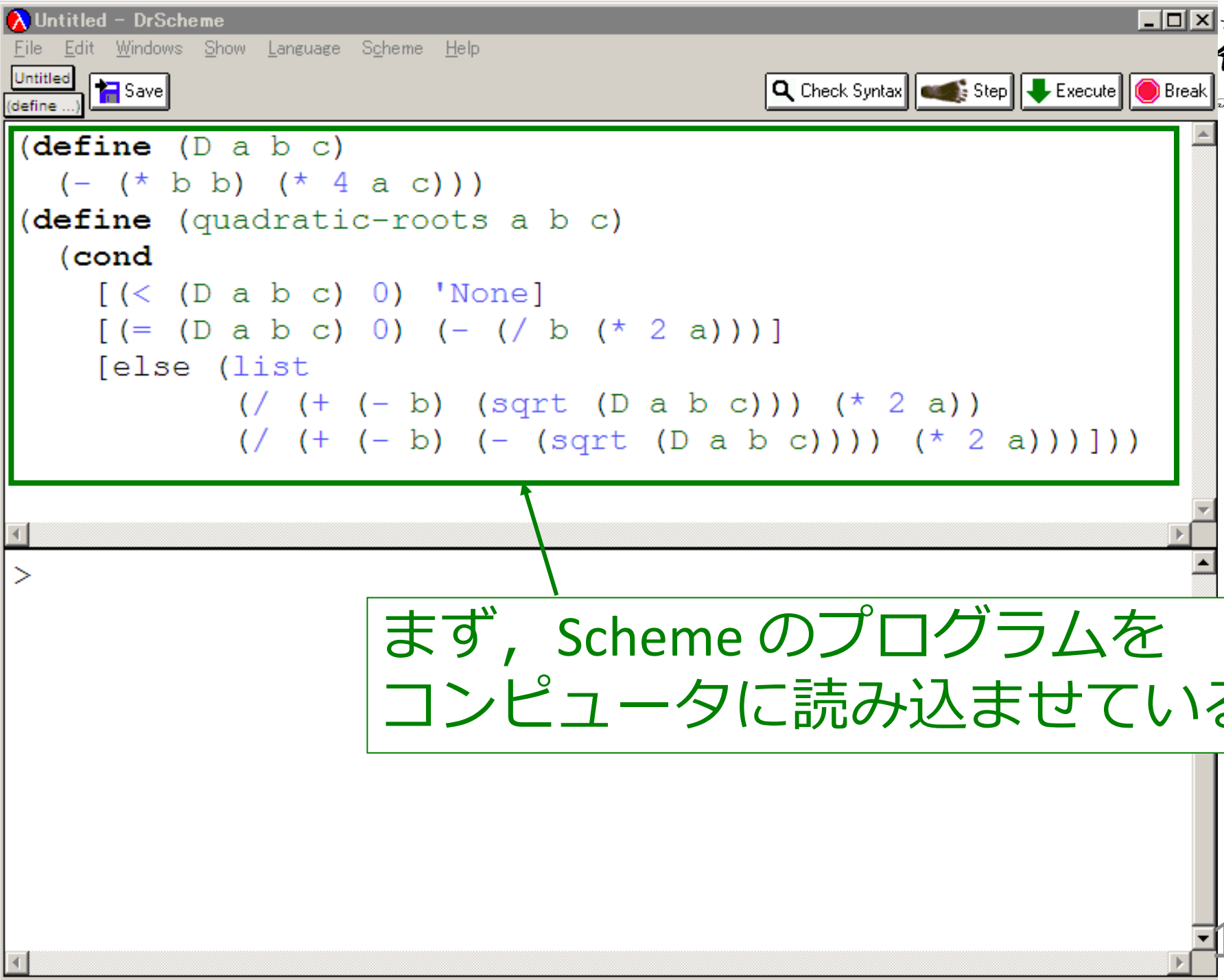
1. 次を「定義用ウインドウ」で，実行しなさい
   - 入力した後に，Execute ボタンを押す

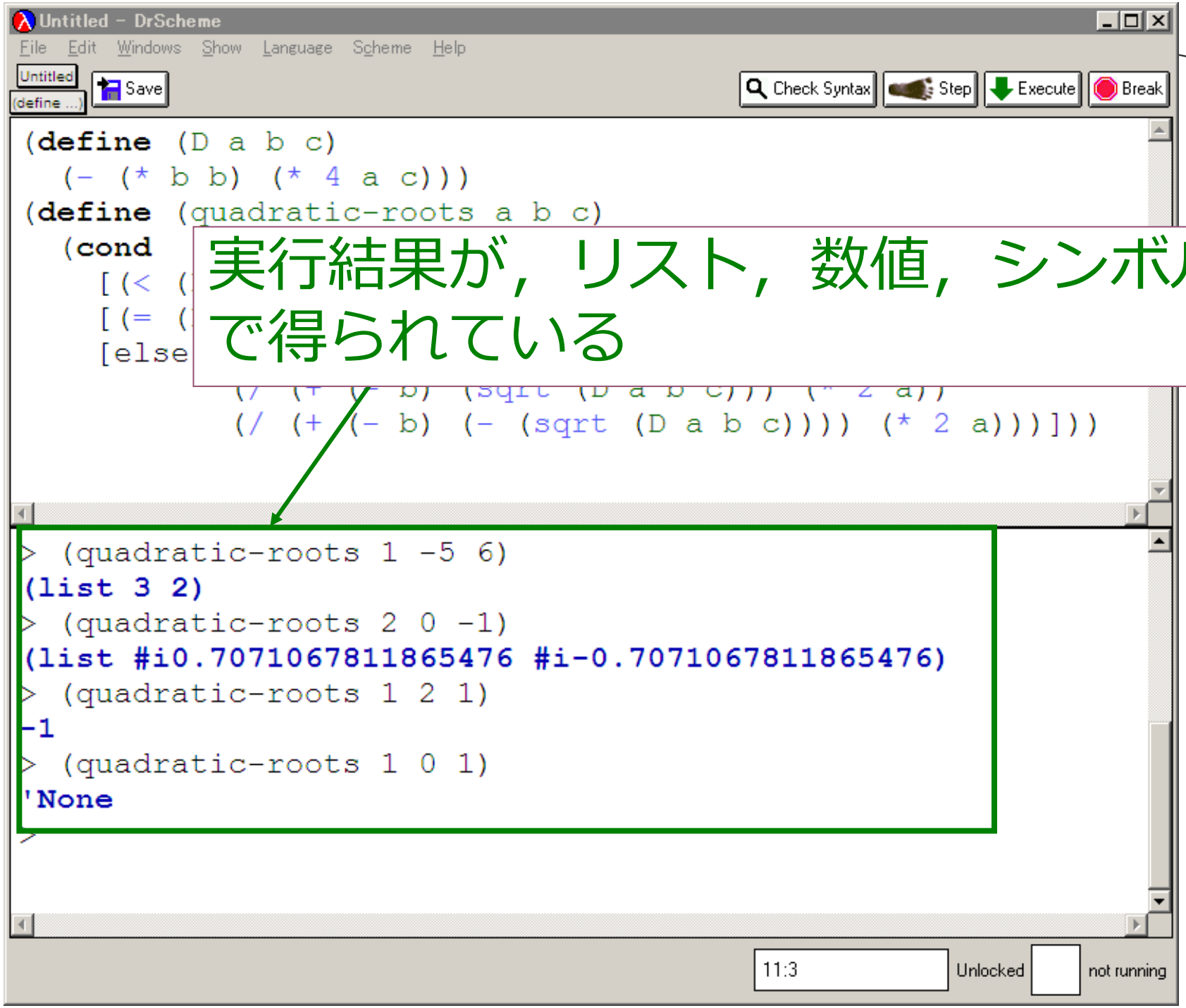
```
(define (D a b c)
  (- (* b b) (* 4 a c)))
(define (quadratic-roots a b c)
  (cond
    [(< (D a b c) 0) 'None]
    [(= (D a b c) 0) (- (/ b (* 2 a)))]
    [else (list
           (/ (+ (- b) (sqrt (D a b c))) (* 2 a))
           (/ (+ (- b) (- (sqrt (D a b c)))) (* 2 a)))]))
```

2. その後，次を「実行用ウインドウ」で実行しなさい

```
(quadratic-roots 1 -5 6)
(quadratic-roots 2 0 -1)
(quadratic-roots 1 2 1)
(quadratic-roots 1 0 1)
```

☆　次は，例題２に進んでください


Untitled – DrScheme
File Edit Windows Show Language Scheme Help
Untitled
(define ...)
Save
Check Syntax
Step
Execute
Break
(define (D a b c)
(- (* b b) (* 4 a c)))
(define (quadratic-roots a b c)
(cond
[(< (D a b c) 0) 'None]
[(= (D a b c) 0) (- (/ b (* 2 a)))]
[else (list
(/ (+ (- b) (sqrt (D a b c))) (* 2 a))
(/ (+ (- b) (- (sqrt (D a b c)))) (* 2 a)))]))
>
まず，Scheme のプログラムを
コンピュータに読み込ませている


Untitled - DrScheme
File Edit Windows Show Language Scheme Help
Untitled
(define ...)
Save
Check Syntax
Step
Execute
Break
(define (D a b c)
(- (* b b) (* 4 a c)))
(define (quadratic-roots a b c)
(cond
[else
(/ (+ (- b) (sqrt (D a b c))) (* 2 a)))]))
実行結果が，リスト，数値，シンボルで得られている
> (quadratic-roots 1 -5 6)
(list 3 2)
> (quadratic-roots 2 0 -1)
(list #i0.7071067811865476 #i-0.7071067811865476)
> (quadratic-roots 1 2 1)
-1
> (quadratic-roots 1 0 1)
'None
>
11:3
Unlocked
not running

入力と出力

入力は
３つの数値

出力は
・１つのリスト
・１つの数値
・シンボル 'None
のどれか

quadraric-roots 関数

```
(define (D a b c)
  (- (* b b) (* 4 a c)))
(define (quadratic-roots a b c)
  (cond
    [(< (D a b c) 0) 'None]
    [(= (D a b c) 0) (- (/ b (* 2 a)))]
    [else (list
           (/ (+ (- b) (sqrt (D a b c))) (* 2 a))
           (/ (+ (- b) (- (sqrt (D a b c)))) (* 2 a)))]))
```

例題２．リストの生成

- n 個の数字 「1」 を要素とするリストを生成する関数 1list を作り，実行する
  - cons を使用する

「例題２．リストの生成」の手順

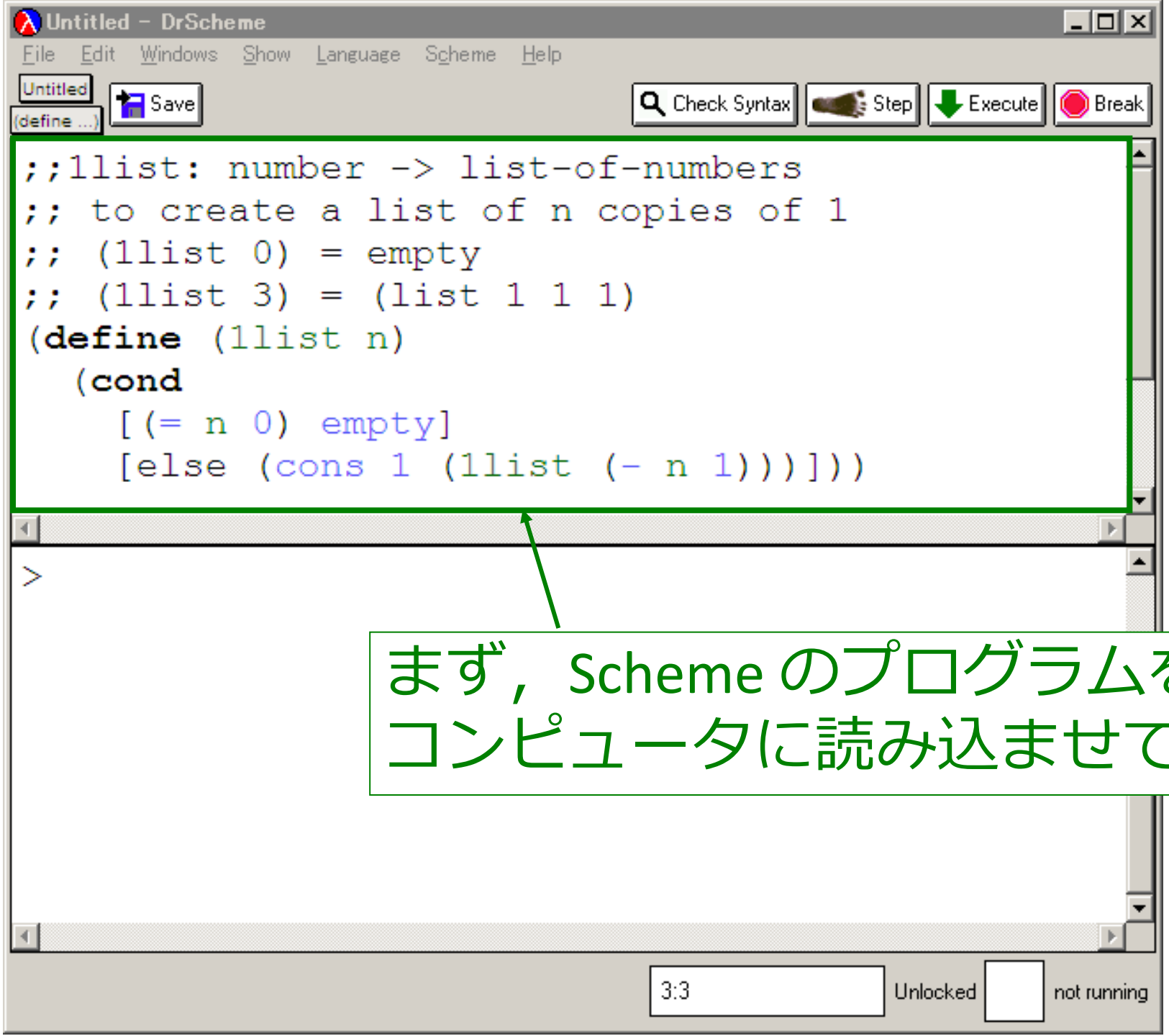
1. 次を「定義用ウインドウ」で，実行しなさい
   - 入力した後に，Execute ボタンを押す

```
(define (1list n)
  (cond
    [(= n 0) empty]
    [else (cons 1 (1list (- n 1)))]))
```

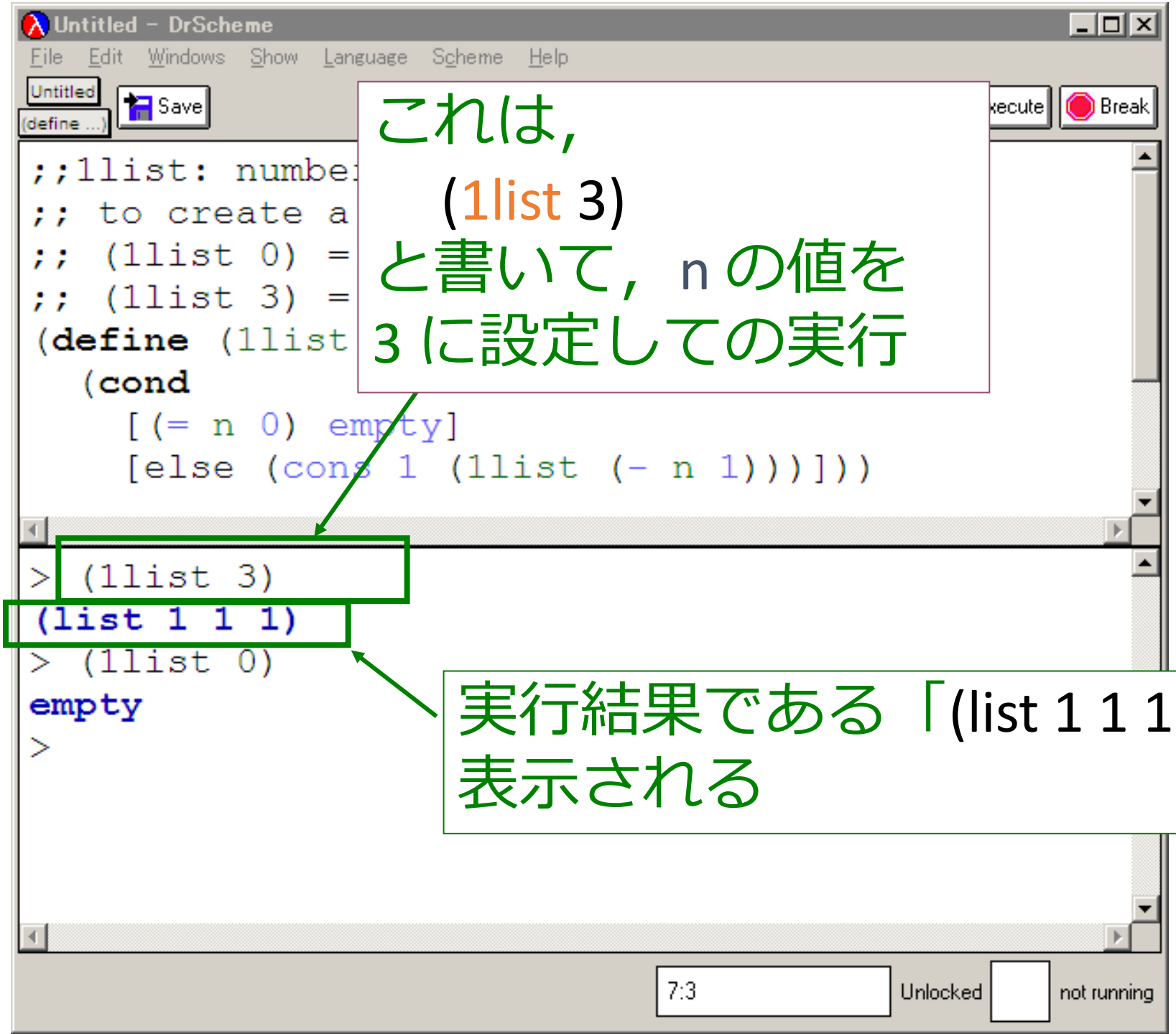
2. その後，次を「実行用ウインドウ」で実行しなさい

```
(1list 0)
(1list 3)
```

☆　次は，例題３に進んでください



まず，Scheme のプログラムを
コンピュータに読み込ませている



これは，

(1list 3)

と書いて，n の値を3 に設定しての実行

実行結果である「(list 1 1 1)」が表示される

入力と出力

入力は
１つの数値

出力は
１つのリスト

1list 関数

```
;;1list: number -> list-of-numbers
;; to create a list of n copies of 1
;; (1list 0) = empty
;; (1list 3) = (list 1 1 1)
(define (1list n)
   (cond
      [(= n 0) empty]
      [else (cons 1 (1list (- n 1)))]))
```

リストの生成

1. n = 0 ならば：　　→　終了条件

   empty　　→　自明な解

2. そうで無ければ：
   - 長さ n-1 のリストを作り，その先頭に「１」をつなげる
   - リストの先頭に「１」をつなげるために，cons を使う

リストの生成 1list

- 1list の内部に 1list が登場

```
(define (1list n)
  (cond
    [(= n 0) empty]
    [else (cons 1 (1list (- n 1)))]))
```

- 1list の実行が繰り返される

例：(1list 3)

= (cons 1 (1list 2))

例題３．ステップ実行

- 関数 1list （例題２）について，実行結果に至る過程を見る
  - (1list 3) から (list 1 1 1) に至る過程を見る
  - DrScheme の stepper を使用する

```
(define (1list n)
  (cond
    [(= n 0) empty]
    [else (cons 1 (1list (- n 1)))]))
(1list 3)
```

```
(1list 3)
= ...
= (cons 1 (1list 2))
= ...
= (cons 1 (cons 1 (1list 1)))
= ...
= (cons 1 (cons 1 (cons 1 (1list 0))))
= ...
= (cons 1 (cons 1 (cons 1 empty)))
= (list 1 1 1)
```

「例題３．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，Execute ボタンを押す

```
(define (1list n)
  (cond
    [(= n 0) empty]
    [else (cons 1 (1list (- n 1)))]))
(1list 3)
```

← 例題２と同じ

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

☆　次は，例題４に進んでください

(1list 3) から
(list 1 1 1) が得られる過程の概略

(1list 3) 最初の式

= …

= (cons 1 (1list 2))

= …

= (cons 1 (cons 1 (1list 1)))

= …

= (cons 1 (cons 1 (cons 1 (1list 0))))

= …

= (cons 1 (cons 1 (cons 1 empty))) コンピュータ内部での計算

= (list 1 1 1) 実行結果

(1list 3) から
(cons 1 (1list 2)) が得られる過程

(1list 3)
= …
= (cons 1 (1list 2))

この部分は

(1list 3)
= (cond
  [(= 3 0) empty]
  [else (cons 1 (1list (- 3 1)))])
= (cond
  [false empty]
  [else (cons 1 (1list (- 3 1)))])
= (cons 1 (1list (- 3 1)))
= (cons 1 (1list 2))

= (cons 1 (cons 1 (1list 1))
= …
= (cons 1 (cons 1 (cons 1
= …
= (cons 1 (cons 1 (cons 1 empty)))
= (list 1 1 1)

(1list 3) から
(cons 1 (1list 2)) が得られる過程

(1list 3)
= …
= (cons 1 (1list 2))
= …

この
部分は

(1list 3)
= (cond
  [(= 3 0) empty]
  [else (cons 1 (1list (- 3 1)))])
= (cond

これは,

```
(define (1list n)
  (cond
    [(= n 0) empty]
    [else (cons 1 (1list (- n 1)))]))
```

の n を 3 で置き換えたもの

例題４．リストの生成

- n 個のシンボル「'*」 を要素とするリストを生成する関数 astlist を作り，実行する
  - cons を使用する

「例題４．リストの生成」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，Execute ボタンを押す

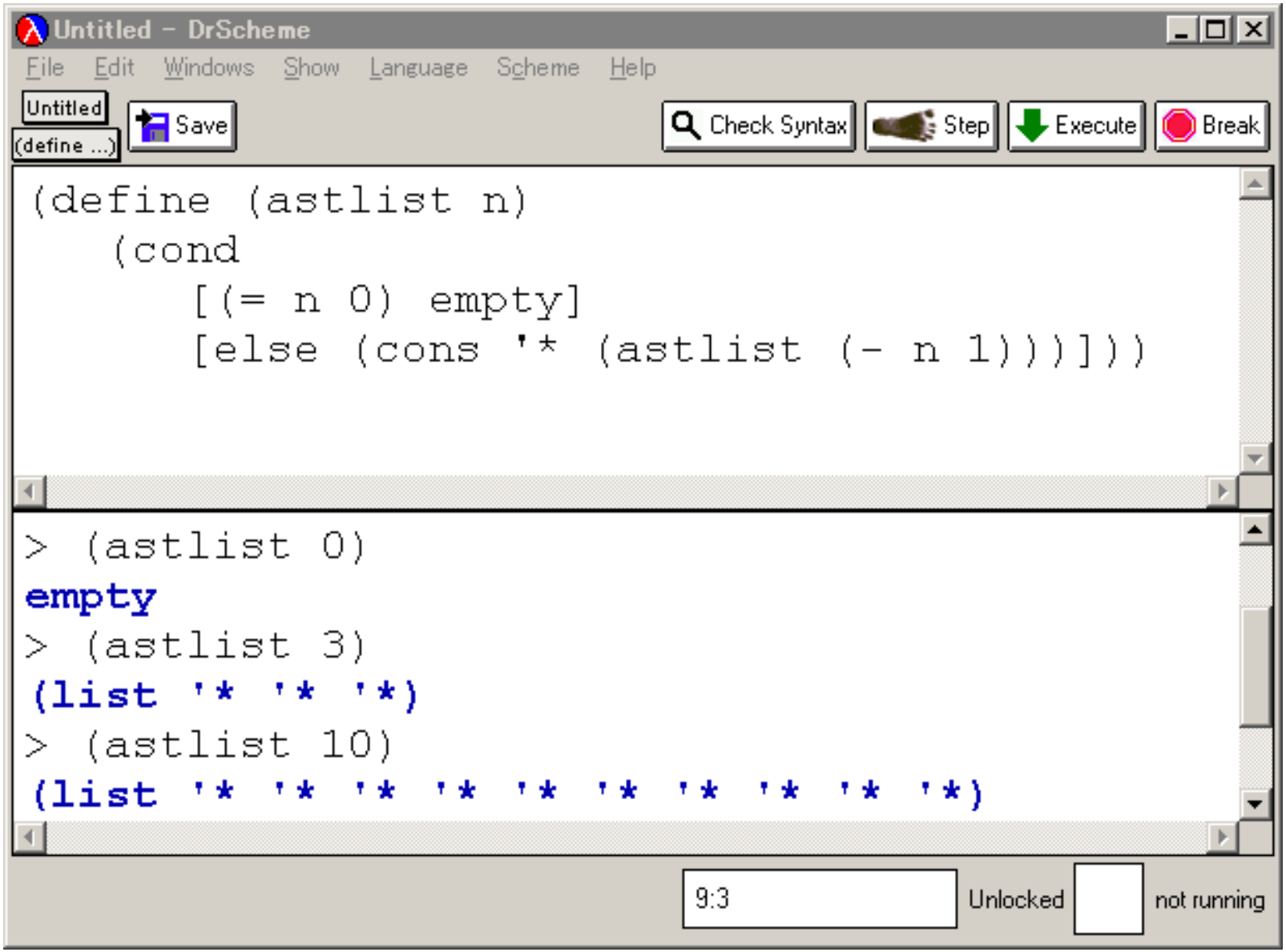
```
(define (astlist n)
  (cond
    [(= n 0) empty]
    [else (cons '* (astlist (- n 1)))]))
```

2. その後，次を「実行用ウインドウ」で実行しなさい

```
(astlist 0)
(astlist 3)
(astlist 10)
```

☆　次は，例題５に進んでください

実行結果の例


Untitled - DrScheme
File Edit Windows Show Language Scheme Help
Untitled
(define ...)
Save
Check Syntax
Step
Execute
Break
(define (astlist n)
(cond
[(= n 0) empty]
[else (cons '* (astlist (- n 1)))]))
> (astlist 0)
empty
> (astlist 3)
(list '* '* '*)
> (astlist 10)
(list '* '* '* '* '* '* '* '* '* '*)
9:3
Unlocked
not running

入力と出力

入力は数値

出力はリスト

リストの生成

1. n = 0 ならば：　→　終了条件

   empty　→　自明な解

2. そうで無ければ：
   - 長さ n-1 のリストを作り，その先頭に「'*」をつなげる
   - リストの先頭に「'*」をつなげるために，cons を使う

astlist 関数

```
;; astlist: number -> list of symbols
;; to create a list of n copies of '*
;; (astlist 0) = empty
;; (astlist 3) = (list '* '* '*)
(define (astlist n)
  (cond
    [(= n 0) empty]
    [else (cons '* (astlist (- n 1)))]))
```

(astlist 3) から
(list '* '* '*) が得られる過程の概略

= (astlist 3)
= …
= (cons '* (astlist 2))
= …
= (cons '* (cons '* (astlist 1)))
= …
= (cons '* (cons '* (cons '* (astlist 0))))
= …
= (cons '* (cons '* (cons '* empty)))
= (list '* '* '*)

例題５．賃金リストの生成

- 賃金を求める関数 wage を作り実行する
  - 従業員について

    賃金　＝　１２×勤務時間
- 全従業員の勤務時間のリスト alon から，賃金のリストを生成する関数 hours->wages を作り，実行する
  - 関数 wage を使う

「例題５．賃金リストの生成」の手順

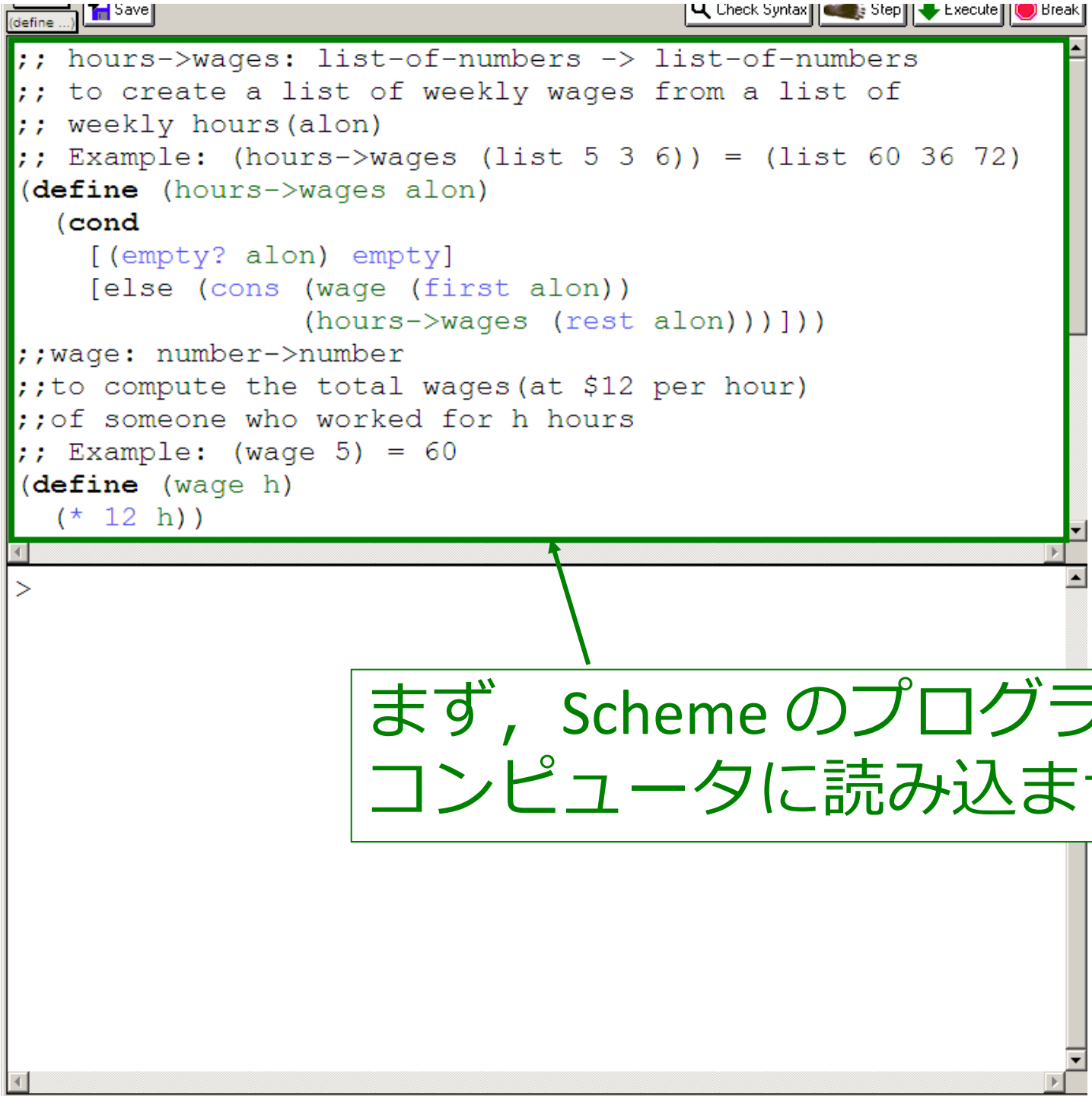
1. 次を「定義用ウインドウ」で，実行しなさい
   - 入力した後に，Execute ボタンを押す

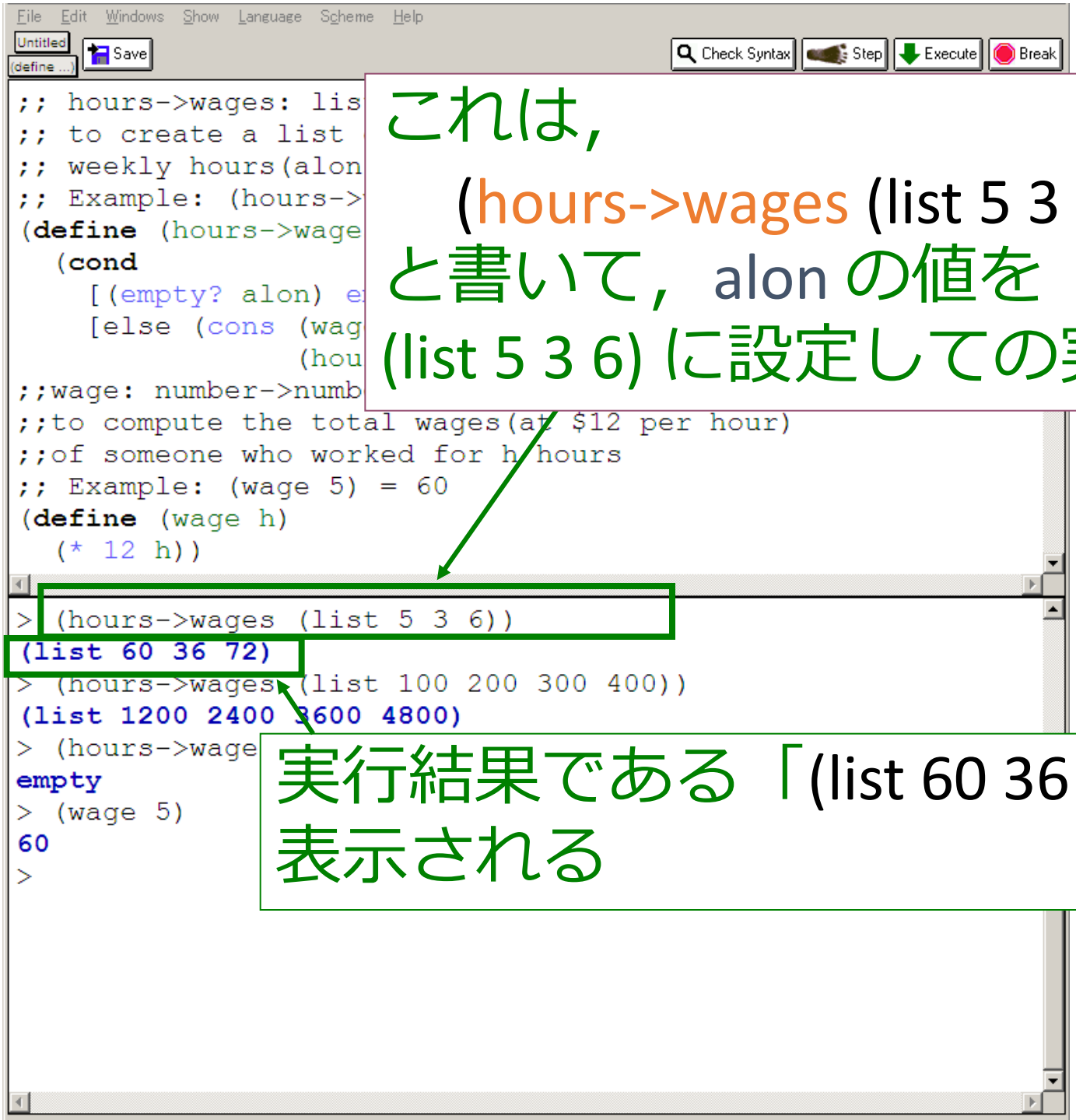
```
(define (hours->wages alon)
  (cond
    [(empty? alon) empty]
    [else (cons (wage (first alon))
                (hours->wages (rest alon)))]))
(define (wage h)
  (* 12 h))
```

2. その後，次を「実行用ウインドウ」で実行しなさい

```
(hours->wages (list 5 3 6))
(hours->wages (list 100 200 300 400))
(hours->wages empty)
```

☆　次は，例題６に進んでください


(define ...)
Save
Check Syntax
Step
Execute
Break
;; hours->wages: list-of-numbers -> list-of-numbers
;; to create a list of weekly wages from a list of
;; weekly hours(alon)
;; Example: (hours->wages (list 5 3 6)) = (list 60 36 72)
(define (hours->wages alon)
(cond
[(empty? alon) empty]
[else (cons (wage (first alon))
(hours->wages (rest alon)))]))
;;wage: number->number
;;to compute the total wages(at $12 per hour)
;;of someone who worked for h hours
;; Example: (wage 5) = 60
(define (wage h)
(* 12 h))
>
まず，Scheme のプログラムを
コンピュータに読み込ませている


File Edit Windows Show Language Scheme Help
Untitled
(define ...)
Save
Check Syntax
Step
Execute
Break
;; hours->wages: lis
;; to create a list
;; weekly hours(alon
;; Example: (hours->
(define (hours->wage
(cond
[(empty? alon) e
[else (cons (wag
(hou
;;wage: number->numb
;;to compute the total wages(at $12 per hour)
;;of someone who worked for h hours
;; Example: (wage 5) = 60
(define (wage h)
(* 12 h))
> (hours->wages (list 5 3 6))
(list 60 36 72)
> (hours->wages (list 100 200 300 400))
(list 1200 2400 3600 4800)
> (hours->wage
empty
> (wage 5)
60
>
これは,
(hours->wages (list 5 3 6))
と書いて，alon の値を
(list 5 3 6) に設定しての実行
実行結果である「(list 60 36 72)」が
表示される

入力と出力

入力は
１つのリスト

出力は
１つのリスト

```
;; hours->wages: list-of-numbers -> list-of-numbers
;; to create a list of weekly wages from a list of
;; weekly hours(alon)
;; Example: (hours->wages (list 5 3 6)) = (list 60 36 72)
(define (hours->wages alon)
   (cond
      [(empty? alon) empty]
      [else (cons (wage (first alon))
                  (hours->wages (rest alon)))]))
;;wage: number->number
;;to compute the total wages(at $12 per hour)
;;of someone who worked for h hours
;; Example: (wage 5) = 60
(define (wage h)
    (* 12 h))
```

賃金リストの生成

1. リストが空ならば： → 終了条件
   empty → 自明な解
2. そうで無ければ：
   - 「勤務時間のリストの rest に対する hours->wages の結果（リスト）の先頭に，勤務時間の first に対する wage の結果（数値）をつなげたもの」 が求める解
   - リストの先頭に数値をつなげるために，cons を使う

賃金リストの生成

- hours->wages の内部に hours->wages が登場

```
(define (hours->wages alon)
  (cond
    [(empty? alon) empty]
    [else (cons (wage (first alon))
                (hours->wages (rest alon)))]))
```

- hours->wages の実行が繰り返される

  例： (cons (wage 1) (hours->wages (list 2 3)))

例題６．ステップ実行

- 関数 hours-wage （例題５）について，実行結果に至る過程を見る
  - (hours->wage (list 1 2 3)) から (list 12 24 36) に至る過程を見る
  - DrScheme の stepper を使用する

```
(hours->wages (list 1 2 3))
= …
= (cons (wage 1) (hours->wages (rest (list 1 2 3))))
= …
= (cons 12 (cons (wage 2) (hours->wages (rest (list 2 3)))))
= …
= (cons 12 (cons 24 (cons (wage 3) (hours->wages (rest (list 3))))))
= …
= (cons 12 (cons 24 (cons 36 (hours->wages empty))))
= …
= (cons 12 (cons 24 (cons 36 empty)))
= (list 12 24 36)
```

「例題６．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，Execute ボタンを押す

```
(define (hours->wages alon)
  (cond
    [(empty? alon) empty]
    [else (cons (wage (first alon))
                (hours->wages (rest alon)))]))
(define (wage h)
  (* 12 h))
(hours->wages (list 1 2 3))
```

← 例題５と同じ

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

☆　次は，例題７に進んでください

(hours->wages (list 5 3 6)) から (list 60 36 72) が得られる過程の概略

(hours->wages (list 5 3 6))　最初の式

= ...

= (cons 60 (hours->wages (list 3 6)))

= …

= (cons 60 (cons 36 (hours->wages (list 6))))

= …

= (cons 60 (cons 36 (cons 72 (hours->wages empty)))))

= ...

= (cons 60 (cons 36 (cons 72 empty)))　コンピュータ内部での計算

= (list 60 36 72)　実行結果

(hours->wages (list 5 3 6)) から (cons 60 (hours->wages (list 3 6)))が得られる過程


```
(hours->wages (list 5 3 6))
= ...
= (cons 60 (hours->wages (
= …
= (cons 60 (cons 36 (hours-
= …
= (cons 60 (cons 36 (cons 7
= ...
= (cons 60 (cons 36 (cons 7
= (list 60 36 72)
```

この部分は

```
(hours->wages (list 5 3 6))
= (cond
    [(empty? (list 5 3 6)) empty]
    [else (cons (wage (first (list 5 3 6)))
                (hours->wages (rest (list 5 3 6))))])
= (cond
    [false empty]
    [else (cons (wage (first (list 5 3 6)))
                (hours->wages (rest (list 5 3 6))))])
= (cons (wage (first (list 5 3 6)))
        (hours->wages (rest (list 5 3 6))))
= (cons (wage 5)
        (hours->wages (rest (list 5 3 6))))
= (cons (* 12 5)
        (hours->wages (rest (list 5 3 6))))
= (cons 60 (hours->wages (rest (list 5 3 6))))
= (cons 60 (hours->wages (list 3 6)))
```

(hours->wages (list 5 3 6)) から (cons 60 (hours->wages (list 3 6)))が得られる過程

(hours->wages (list 5 3 6))

= ...

= (cons 60 (hours->wages (

=

=

=

=

=

=

= (list 60 36 72)

この部分は

(hours->wages (list 5 3 6))

= (cond
  [(empty? (list 5 3 6)) empty]
  [else (cons (wage (first (list 5 3 6)))
              (hours->wages (rest (list 5 3 6))))])

= (cond
  [false empty]

= (cons 60 (hours->wages (list 3 6)))

これは,

```
(define (hours->wages alon)
  (cond
    [(empty? alon) empty]
    [else (cons (wage (first alon))
                (hours->wages (rest alon)))]))
```

の alon を (list 5 3 6) で置き換えたもの

例題７．かけ算の表

- ２つの数 m, n から，m 行 n 列のかけ算の表を出力するプログラム hyou を作り，実行する
  - かけ算の表の「１行分」を出力する gyou も作る
  - かけ算の表は，リストのリストの形で得る

「例題７．かけ算の表」の手順

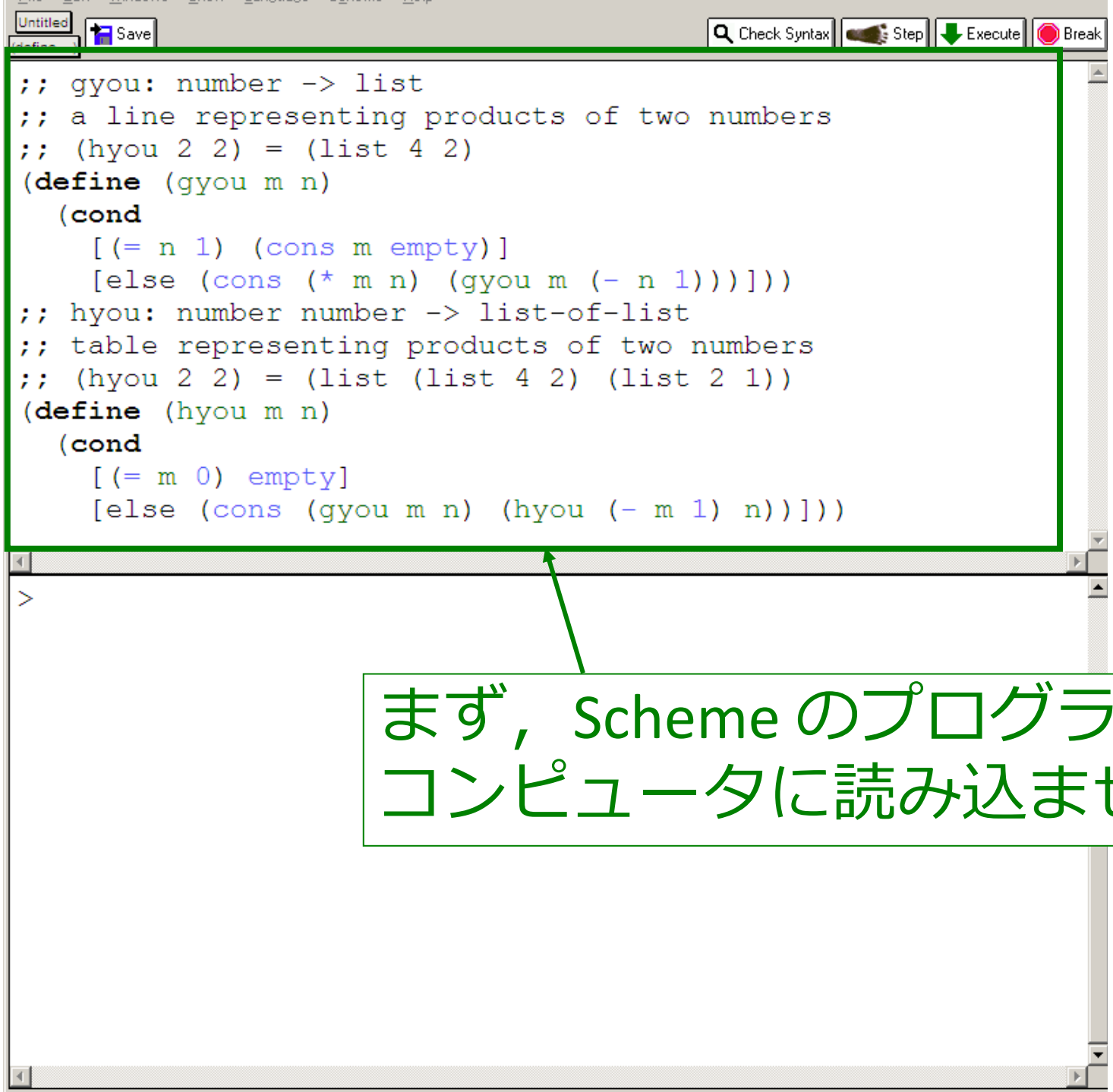
1. 次を「定義用ウインドウ」で，実行しなさい
   - 入力した後に，Execute ボタンを押す

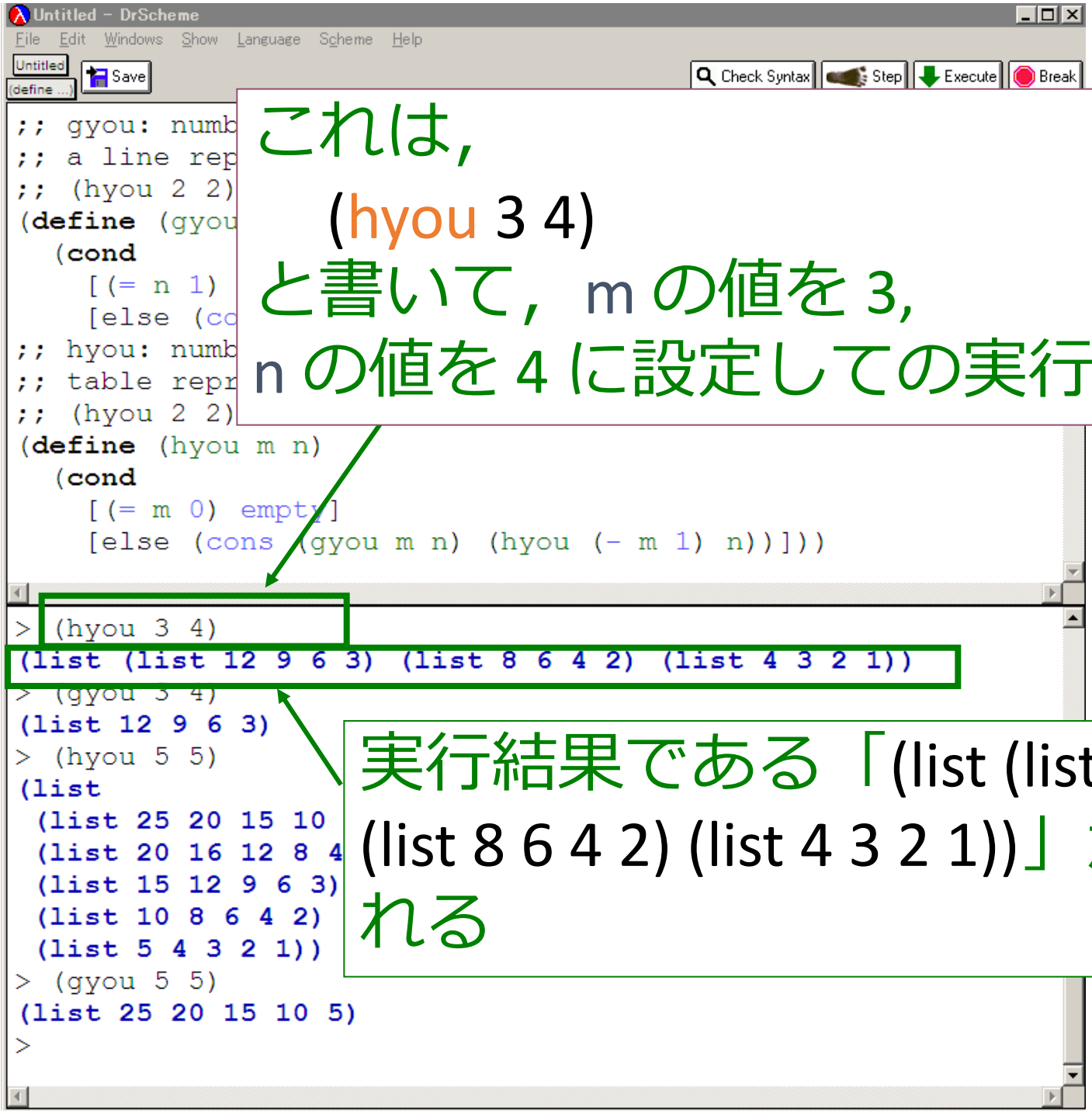
```
;; gyou: number -> list
;; a line representing products of two numbers
;; (hyou 3 4) = (list 12 9 6 3)
(define (gyou m n)
   (cond
      [(= n 1) (cons m empty)]
      [else (cons (* m n) (gyou m (- n 1)))]))
;; hyou: number number -> list-of-list
;; table representing products of two numbers
;; (hyou 2 2) = (list (list 4 2) (list 2 1))
(define (hyou m n)
   (cond
      [(= m 0) empty]
      [else (cons (gyou m n) (hyou (- m 1) n))]))
```

2. その後，次を「実行用ウインドウ」で実行しなさい

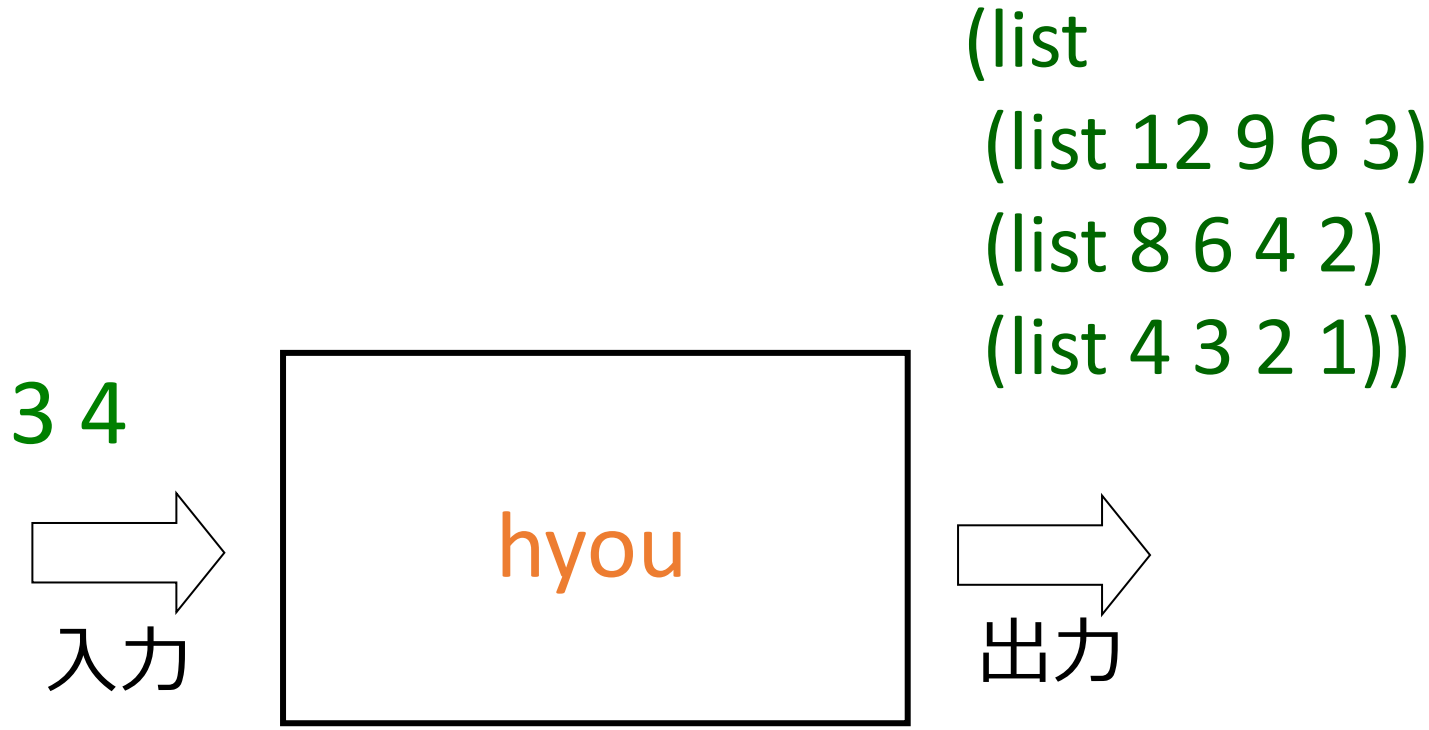
```
(hyou 3 4)
```

☆　次は，例題８に進んでください


Untitled
Save
Check Syntax
Step
Execute
Break
;; gyou: number -> list
;; a line representing products of two numbers
;; (hyou 2 2) = (list 4 2)
(define (gyou m n)
(cond
[(= n 1) (cons m empty)]
[else (cons (* m n) (gyou m (- n 1)))]))
;; hyou: number number -> list-of-list
;; table representing products of two numbers
;; (hyou 2 2) = (list (list 4 2) (list 2 1))
(define (hyou m n)
(cond
[(= m 0) empty]
[else (cons (gyou m n) (hyou (- m 1) n))]))
>
まず，Scheme のプログラムを
コンピュータに読み込ませている


Untitled – DrScheme
File Edit Windows Show Language Scheme Help
Untitled
(define ...)
Save
Check Syntax
Step
Execute
Break
;; gyou: numb
;; a line rep
;; (hyou 2 2)
(define (gyou
(cond
[(= n 1)
[else (co
;; hyou: numb
;; table repr
;; (hyou 2 2)
(define (hyou m n)
(cond
[(= m 0) empty]
[else (cons (gyou m n) (hyou (- m 1) n))]))
> (hyou 3 4)
(list (list 12 9 6 3) (list 8 6 4 2) (list 4 3 2 1))
> (gyou 3 4)
(list 12 9 6 3)
> (hyou 5 5)
(list
(list 25 20 15 10
(list 20 16 12 8 4
(list 15 12 9 6 3)
(list 10 8 6 4 2)
(list 5 4 3 2 1))
> (gyou 5 5)
(list 25 20 15 10 5)
>
これは,
(hyou 3 4)
と書いて，m の値を 3,
n の値を 4 に設定しての実行
実行結果である「(list (list 12 9 6 3)
(list 8 6 4 2) (list 4 3 2 1))」が表示さ
れる

入力と出力



入力は,
2つの数値

出力はリストのリスト

```
;; gyou: number -> list
;; a line representing products of two numbers
;; (hyou 3 4) = (list 12 9 6 3)
(define (gyou m n)
    (cond
        [(= n 1) (cons m empty)]
        [else (cons (* m n) (gyou m (- n 1)))]))
;; hyou: number number -> list-of-list
;; table representing products of two numbers
;; (hyou 2 2) = (list (list 4 2) (list 2 1))
(define (hyou m n)
    (cond
        [(= m 0) empty]
        [else (cons (gyou m n) (hyou (- m 1) n))]))
```

hyou, gyou の関係

- hyou
  - 1 つの表を作る

    例）　(list (list 12 9 6 3) (list 8 6 4 2) (list 4 3 2 1))
  - gyou を使用

- gyou
  - 1 行分を作る

    例）　(list 12 9 6 3)

かけ算の表　hyou

1. 縦の数 m = 0 ならば：　→　終了条件

   empty　→　自明な解

2. そうで無ければ：
   - １行分を作ることを m 回繰り返す

かけ算の表　hyou

- hyou の内部に hyou が登場

```
(define (hyou m n)
   (cond
      [(= m 0) empty]
      [else (cons (gyou m n) (hyou (- m 1) n))]))
```

- hyou の実行が繰り返される

例：(hyou 5 5) = (cons (list 25 20 15 10 5) (hyou 4 5))

(hyou 4 5) = (cons (list 20 16 12 8 4) (hyou 3 5))

⇒ まさに「再帰」である

例題８．ステップ実行

- 関数 hyou （例題７）について，実行結果に至る過程を見る
  - (gyou 3 4) から (list 12 6 9 3) に至る過程と，(hyou 5 5) から実行結果に至る過程を見る
  - DrScheme の stepper を使用する

「例題８．ステップ実行」の手順　(1/2)

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，Execute ボタンを押す

```
;; gyou: number -> list
;; a line representing products of two numbers
;; (hyou 3 4) = (list 12 9 6 3)
(define (gyou m n)
   (cond
      [(= n 1) (cons m empty)]
      [else (cons (* m n) (gyou m (- n 1)))]))
;; hyou: number number -> list-of-list
;; table representing products of two numbers
;; (hyou 2 2) = (list (list 4 2) (list 2 1))
(define (hyou m n)
   (cond
      [(= m 0) empty]
      [else (cons (gyou m n) (hyou (- m 1) n))]))
(gyou 3 4)
```

← 例題７と同じ

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

☆　次は，次ページに進んでください

「例題８．ステップ実行」の手順　(2/2)

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，Execute ボタンを押す

```
;; gyou: number -> list
;; a line representing products of two numbers
;; (hyou 3 4) = (list 12 9 6 3)
(define (gyou m n)
   (cond
      [(= n 1) (cons m empty)]
      [else (cons (* m n) (gyou m (- n 1)))]))
;; hyou: number number -> list-of-list
;; table representing products of two numbers
;; (hyou 2 2) = (list (list 4 2) (list 2 1))
(define (hyou m n)
   (cond
      [(= m 0) empty]
      [else (cons (gyou m n) (hyou (- m 1) n))]))
(hyou 5 5)
```

← 例題７と同じ

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

☆　次は，課題に進んでください

(gyou 3 4)から (list 12 9 6 3) が得られる過程の概略

(gyou 3 4) 最初の式

= …
= (cons 12 (gyou 3 3))
= …
= (cons 12 (cons 9 (gyou 3 2)))
= …
= (cons 12 (cons 9 (cons 6 (gyou 3 1))))
= … コンピュータ内部での計算
= (cons 12 (cons 9 (cons 6 (cons 3 empty))))
= (list 12 9 6 3) 実行結果

(gyou 3 4)から (cons (gyou 3 3)) が得られる過程

(gyou 3 4)
= …
= (cons 12 (gyou 3 3))
= …
= (cons 12 (cons 9 (gyou
= …
= (cons 12 (cons 9 (cons
= …
= (cons 12 (cons 9 (cons 6 (cons 3 empty))))
= (list 12 9 6 3)

この部分は

(gyou 3 4)
= (cond
[(= 4 1) (cons 3 empty)]
[else (cons (* 3 4) (gyou 3 (- 4 1)))])
= (cond
[false (cons 3 empty)]
[else (cons (* 3 4) (gyou 3 (- 4 1)))])
= (cons (* 3 4) (gyou 3 (- 4 1)))
= (cons 12 (gyou 3 (- 4 1)))
= (cons 12 (gyou 3 3))

(gyou 3 4)から (cons (gyou 3 3)) が得られる過程

(gyou 3 4)
= …
= (cons 12 (gyou 3 3))
= …

(gyou 3 4)
= (cond
  [(= 4 1) (cons 3 empty)]
  [else (cons (* 3 4) (gyou 3 (- 4 1)))])

この部分は

= (cond
  [false (cons 3 empty)]

= これは,
  (define (gyou m n)
    (cond
      [(= n 1) (cons m empty)]
      [else (cons (* m n) (gyou m (- n 1)))]))
= の m を 3 で，n を 4 で置き換えたもの

(hyou 5 5)から 結果が得られる過程の概略

(hyou 5 5) 最初の式

= …
= (cons (list 25 20 15 10 5) (hyou 4 5))
= …
= (cons (list 25 20 15 10 5) (cons (list 20 16 12 8 4) (hyou 3 5)))
= …
= (cons (list 25 20 15 10 5) (cons (list 20 16 12 8 4) (cons (list 15 12 9 6 3) (hyou 2 5))))
= …
= (cons (list 25 20 15 10 5) (cons (list 20 16 12 8 4) (cons (list 15 12 9 6 3) (cons (list 10 8 6 4 2) (hyou 1 5)))))
= …
= (cons (list 25 20 15 10 5) (cons (list 20 16 12 8 4) (cons (list 15 12 9 6 3) (cons (list 10 8 6 4 2) (cons (list 5 4 3 2 1) (cons (hyou 0 5)))))))
= …
= (cons (list 25 20 15 10 5) (cons (list 20 16 12 8 4) (cons (list 15 12 9 6 3) (cons (list 10 8 6 4 2) (cons (list 5 4 3 2 1) (cons empty)))))))
= (list (list 25 20 15 10 5) (list 20 16 12 8 4) (list 15 12 9 6 3) (list 10 8 6 4 2) (list 5 4 3 2 1))

コンピュータ内部での計算

実行結果

7-3 課題

課題 1

- 実行結果を報告しなさい
  - 「DrScheme の実行用ウインドウ」で実行して，実行結果を報告しなさい

```
(cons 1 (cons 2 (cons 3 empty)))
```

課題２

- 次の関数 insert について，「(insert 4 (list 5 1)) 」から「(list 5 4 1)」が得られる過程の概略を数行程度で説明しなさい
  - DrScheme の stepper を使うと，すぐに分かる

```
(define (insert n alon)
  (cond
    [(empty? alon) (cons n empty)]
    [else (cond
        [(<= (first alon) n) (cons n alon)]
        [(> (first alon) n)
            (cons (first alon) (insert n (rest alon)))])]))
```

課題２．リストと再帰の組み合わせ

- 次の関数 insert について，，「(relative-to-absolute (list 1 2 3))」 から 「(list 1 3 6)」が得られる過程の概略を数行程度で説明しなさい
  - DrScheme の stepper を使うと，すぐに分かる

```
;; relative-2-absolute : list-of-numbers -> list-of-numbers
(define (relative-2-absolute alon)
  (cond
    [(empty? alon) empty]
    [else (cons (first alon)
                (add-to-each (first alon) (relative-2-absolute (rest alon))))]))
;
;; add-to-each : number (listof number) -> (listof number)
(define (add-to-each n alon)
  (cond
    [(empty? alon) empty]
    [else (cons (+ (first alon) n) (add-to-each n (rest alon)))]))
```

課題３

- 数値 n から，「1番目からn番目の奇数のリスト」を作る関数 series-odd-list を作成し，実行結果を報告しなさい
  - 例えば，(series-odd-list 4) = (7 5 3 1)
  - ヒント：　次の空欄を埋めなさい

```
(define (series-odd-list n)
  (cond
    [                                    ]
    [                                    ]))
```

課題４

- 関数 quadratic-roots （例題１）に関係するの問題
  - 二次方程式の係数 a, b, c から解の数を求めるプログラム how-many を作成し，実行結果を報告しなさい
  - 但し，a ≠0 とする

例えば

(how-many 1 0 -1) = 2

(how-many 1 2 1) = 1

課題５．２次方程式の解

- 関数 quadratic-roots （例題１）についての問題
  - a =0 のときにも正しく計算ができるように書き直しを行い，書き直した結果と，実行結果を報告せよ

a = 0 かつ b = 0 かつ c = 0 のとき
　すべての x が解である
a = 0 かつ b = 0 かつ c ≠0 のとき
　解なし
a = 0 かつ b ≠ 0 のとき
　x = - c / b

課題６

- ある年 y のある月 m のある日 d の曜日 youbi から，その「１週間分のリスト」を作る関数を作成し，実行結果を報告しなさい
  - youbi は 0 から 6 までの数値で， 0 なら日曜，1なら月曜・・・
  - 「１週間分のリスト」とは，日曜日から土曜日までのリストで，数値あるいは「'*」である

    例えば
    - y = 2004, m = 10, d = 2, youbi = 6 のとき

    ⇒ 「(list '* '* '* '* '* 1 2)」 が出力される
    - y = 2004, m = 10, d = 31, youbi = 0 のとき

    ⇒ 「(list 31 '* '* '* '* '* '*)」 が出力される

課題 7

- ある年 y のある月 m から，その「月のカレンダー」を作る関数を作成し，実行結果を報告しなさい
  - 「月のカレンダー」は，リストのリスト
  - １週間で１つのリストとなり，５週間あれば，５つのリストからなる大きな１つのリスト

  例えば

```
(list
  (list '* '* '* '* 1 2 3)
  (list 4 5 6 7 8 9 10)
  (list 11 12 13 14 15 16 17)
  (list 18 19 20 21 22 23 24)
  (list 25 26 27 28 29 30 31))
```

課題 8

- ある年 y から，その「年のカレンダー」を作る関数を作成し，実行結果を報告しなさい
  - 「年のカレンダー」は，リストのリストのリスト

さらに勉強したい人への
補足説明事項

リストに関係する関数

- (append list ...)

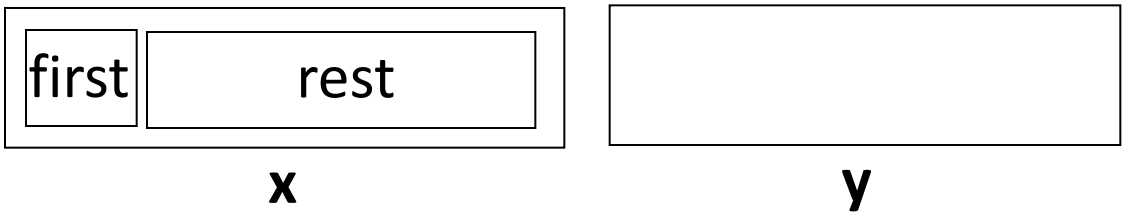
  リストの連結

敢えて自分で書いてみた例を以下に紹介する

例題９．リストの連結

- ２つのリスト（x と y）を連結する関数 my-append を作り，実行する
  1. x が空ならば：　y
  2. そうで無ければ：

     x の rest と y とを連結し，x の first と cons でつなげる



- リストが空であるかどうかを調べるために empty? を使う

my-append 関数

```
(define (my-append x y)
    (cond
        [(empty? x) y]
        [else (cons (first x) (my-append (rest x) y))]))
```