sp-9. 高階関数

（Scheme プログラミング）

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

金子邦彦

アウトライン

9-1 高階関数

9-2 パソコン演習

9-3　課題

今日の到達目標

- 高階関数のプログラムを理解できるようになる
  1. リスト処理の関数 reduce
  2. 数列の和　series
  3. べき級数　taylor-series
     - 指数関数のテーラー展開 my-exp
     - cos 関数のテーラー展開 my-cos, my-log
     - 対数関数のテーラー展開 my-cos, my-log

高階関数とは

- 関数の入力が，関数である

置き換えモデル

置き換えモデル(substitution model)のプロセス

- 関数のパラメータが，実際の値で置き換わる
- 変数が実際の値で置き換わる
- 関数が，その「中身」で置き換わる

#i の意味

- (expt 11 200)
  11^{200} を，有理数で計算
  ⇒　結果は有理数で得られる

- (expt #i11 200)
  11^{200} を，十数桁の小数で計算（近似値）
  ⇒　結果は小数で得られる

(expt #i11 200) = #i1.8990527646046183e+208

この「#i」は「結果が近似値」という意味

e+208 の意味

10 の 208 乗の意味

- 1.8990527646046183e+208
  $= 1.8990527646046183 \times 10^{208}$

[参考]

1.23e-005 $= 1.23 \times 10^{-5}$

$= 0.0000123$

近似計算について

- #iを付けると近似計算
  - 計算結果として，近似値を得たいとき，#i を付ける
    例）(expt #i11 200)

- sin, cos, log 等の値は，近似値でしか得られない
    例：「(sin 0.1)」を実行すると →
    #i0.09983341664682816

- 近似値を使った計算を繰り返すと，誤差が積み重なる

9-2 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
- 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「Advanced Student」に設定
  Language
  → Choose Language
  → Advanced Student
  → OK
  →「Execute ボタン」

Advanced Student

- 高階関数を実行するとき
  - DrScheme の Language で Advanced Student を選択すること

	Intermediate Student	Advanced Student
高階関数の実行	不可	可能
ステップ実行の機能	あり	なし

例題１．リストの総和

- 数のリスト a-list から総和を求める関数 list-sum を作る
- 但し，次の高階関数 reduce を使う

```
;; reduce: (number, data)->data, data, list -> data
;; (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)))]))
```

「例題１．リストの総和」の手順

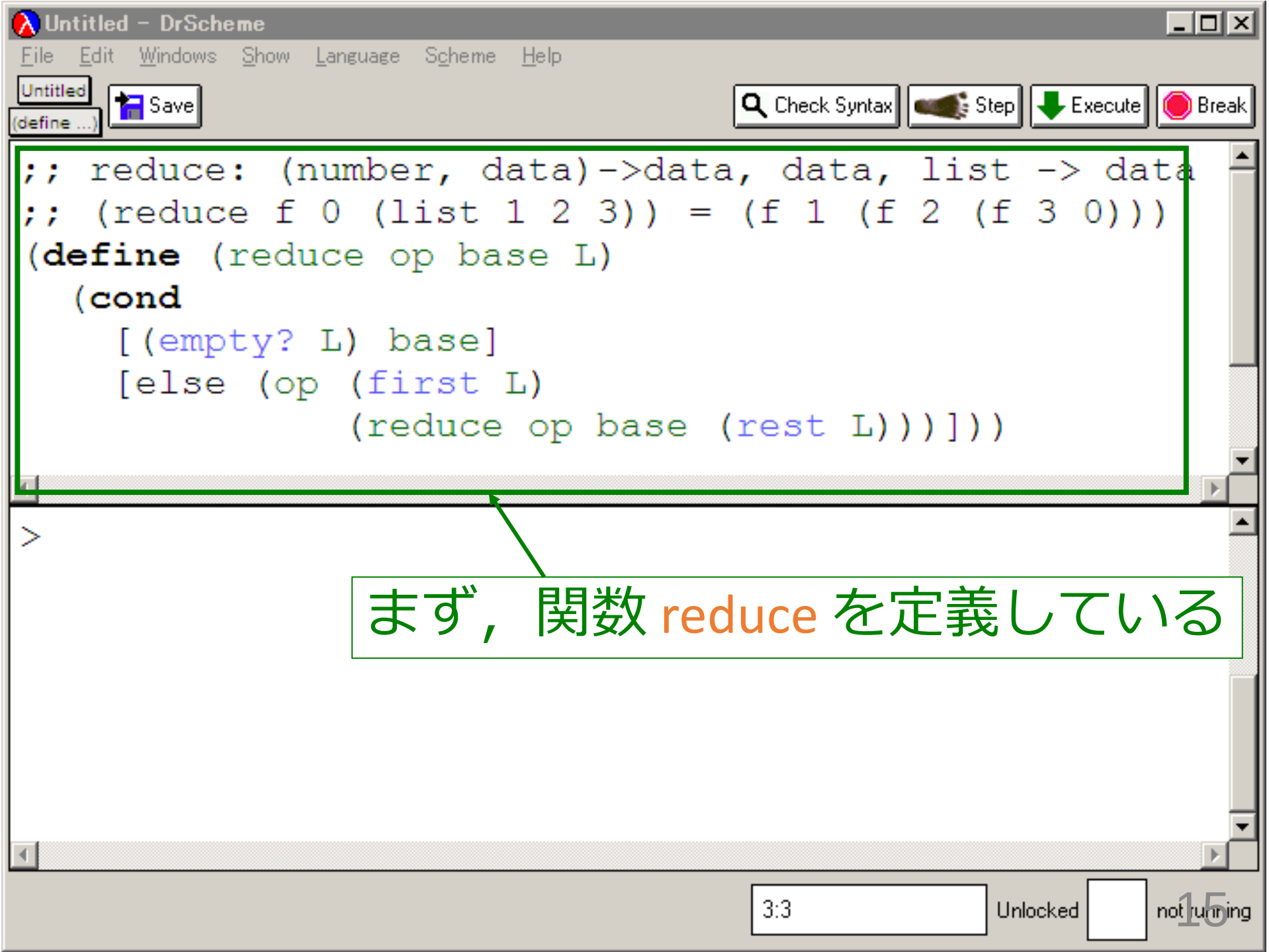
1. 次を「定義用ウインドウ」で，実行しなさい
   - 入力した後に，Execute ボタンを押す

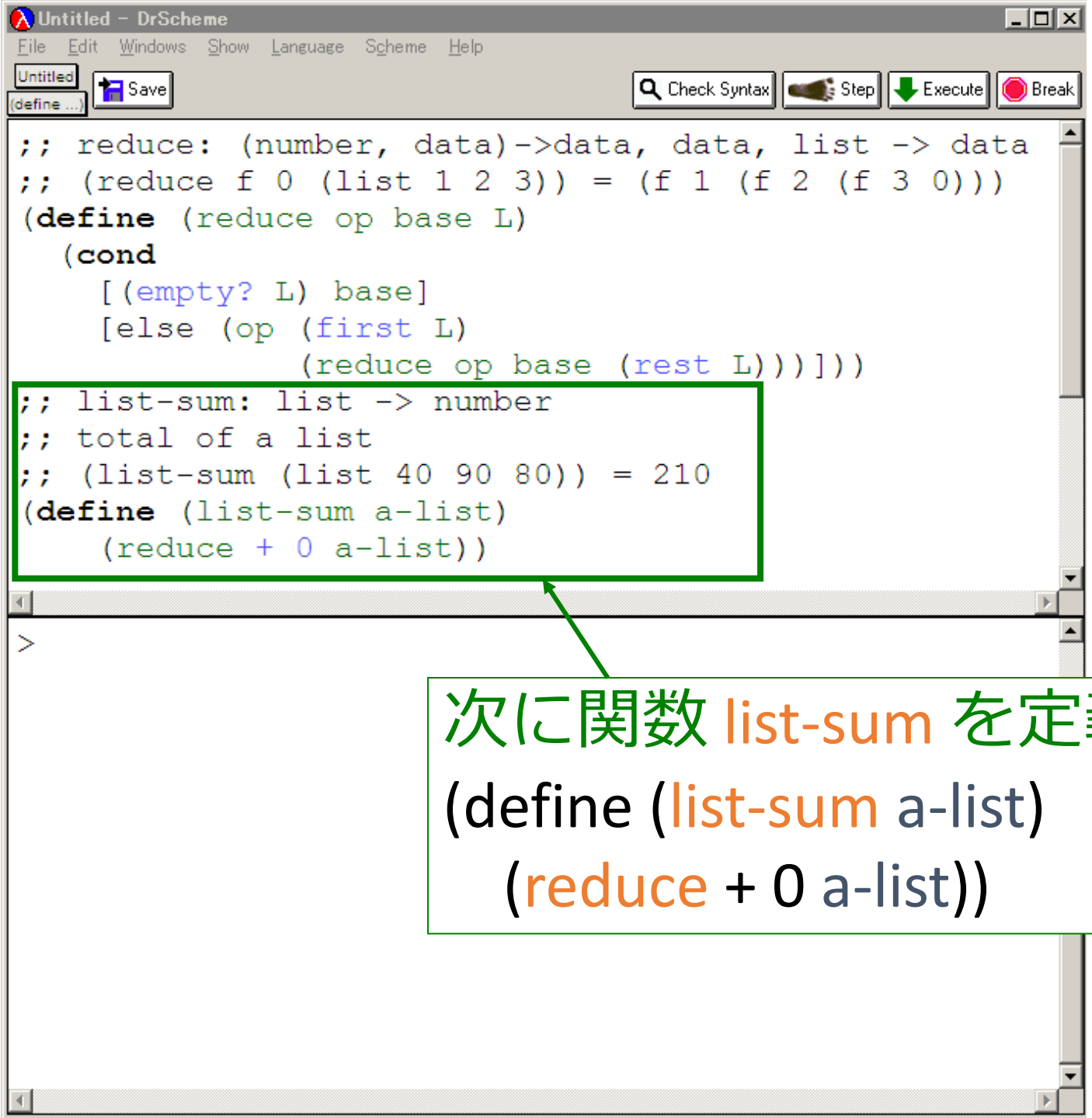
```
;; reduce: (number data -> data) data list -> data
;; (reduce g 0 (list 1 2 3)) = (g 1 (g 2 (g 3 0)))
(define (reduce op base L)
  (cond
    [(empty? L) base]
    [else (op (first L)
              (reduce op base (rest L)))]))
;; list-sum: list -> number
;; total of a list
;; (list-sum (list 40 90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

```
(list-sum (list 1 2 3))
(list-sum empty)
```

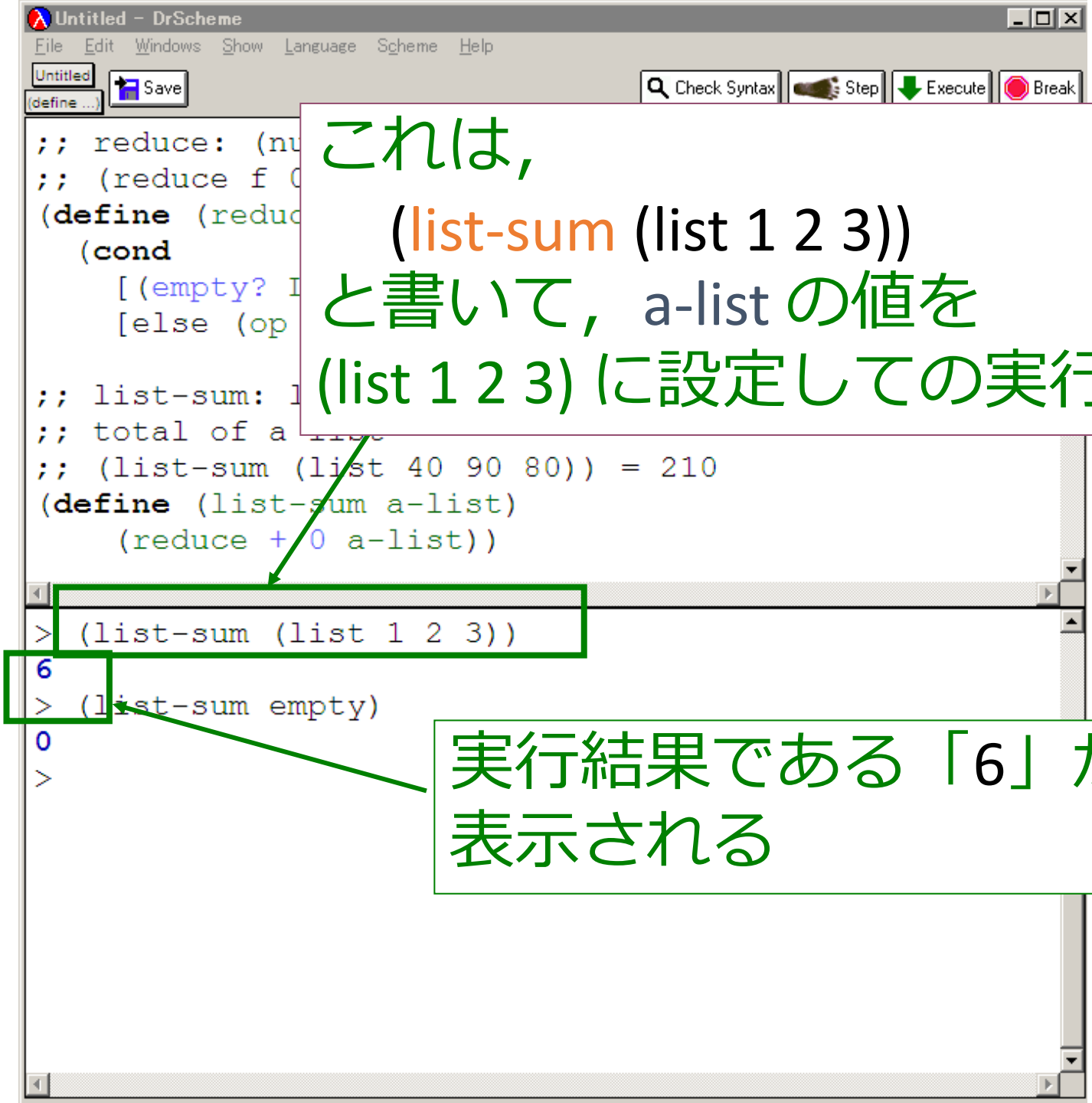
☆　次は，例題２に進んでください


Untitled – DrScheme
File Edit Windows Show Language Scheme Help
Untitled
(define ...)
Save
Check Syntax
Step
Execute
Break
;; reduce: (number, data)->data, data, list -> data
;; (reduce f 0 (list 1 2 3)) = (f 1 (f 2 (f 3 0)))
(define (reduce op base L)
(cond
[(empty? L) base]
[else (op (first L)
(reduce op base (rest L)))]))
>
まず，関数 reduce を定義している
3:3
Unlocked
not running



次に関数 list-sum を定義している

```
(define (list-sum a-list)
   (reduce + 0 a-list))
```



これは,

(list-sum (list 1 2 3))

と書いて，a-list の値を

(list 1 2 3) に設定しての実行

実行結果である「6」が表示される

入力と出力

(list 1 2 3)
0
g

入力

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

出力

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）

```
;; reduce: (number data -> data) data list -> data
;; (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: list -> number
;; total of a list
;; (list-sum (list 40 90 80)) = 210
(define (list-sum a-list)
    (reduce + 0 a-list))
```

関数 op が，「reduce」の入力になっている

(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

(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
  empty))))
= ...
= (+ 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)))
```

(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)
= ..
= (-
  er
= ..
= (-
= ..
= 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]
```

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

例題２．ｘのｎ乗の近似値

- #iを付けた近似計算を試してみるために，expt を使って，11 の 200 乗を計算してみる

```
(expt  11 200)

(expt #i11 200)
```

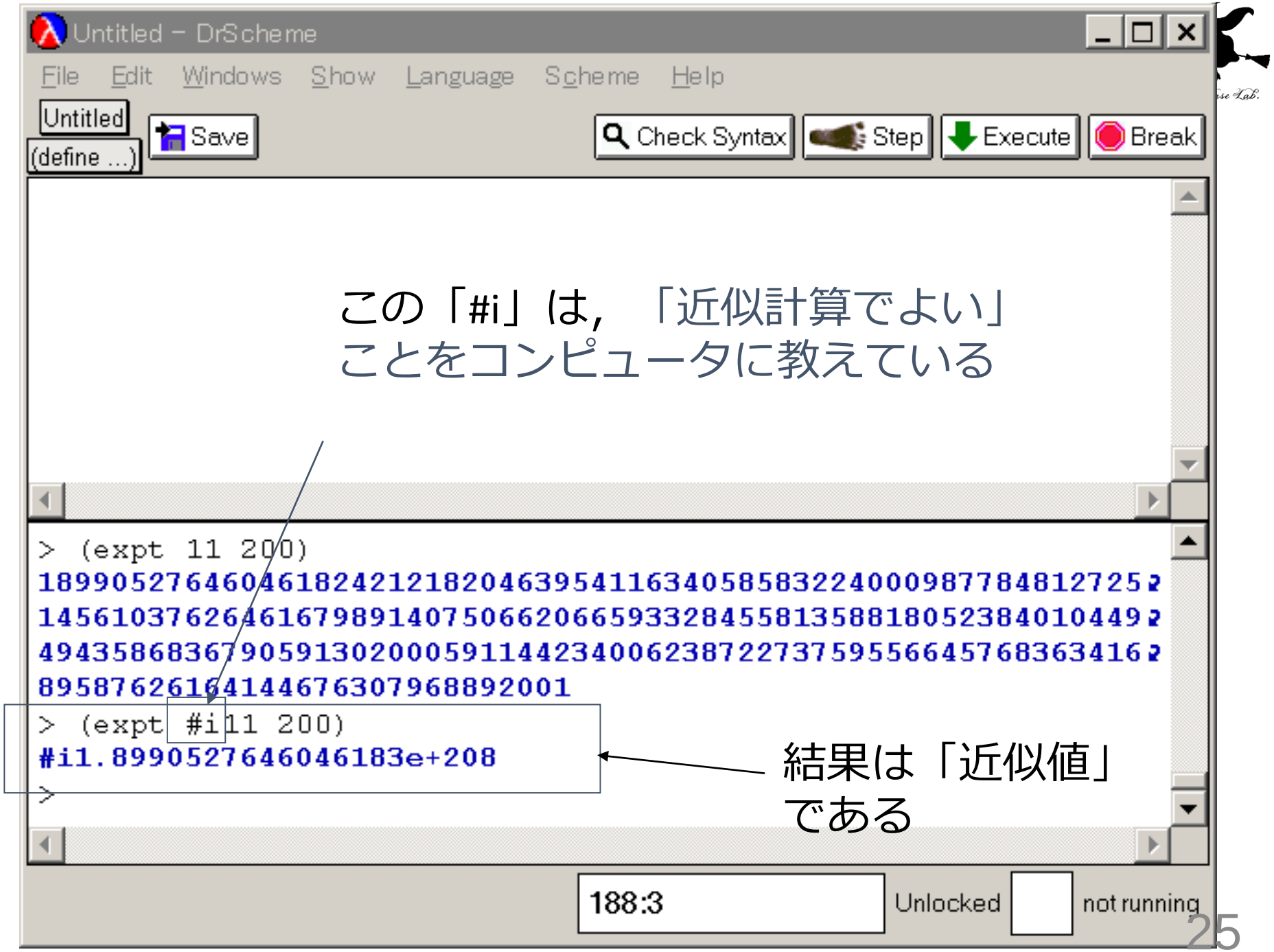
「#i」 は近似値という意味．近似計算でよいことをコンピュータに教えている

「例題２．ｘのｎ乗の近似値」の手順

1. 次を「実行用ウインドウ」で実行しなさい

```
(expt 11 200)
(expt #i11 200)
```

☆　次は，例題３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
この「#i」は，「近似計算でよい」
ことをコンピュータに教えている
> (expt 11 200)
189905276460461824212182046395411634058583224000987784812725?
145610376264616798914075066206659332845581358818052384010449?
494358683679059130200059114423400623872273759556645768363416?
8958762616414467630796892001
> (expt #i11 200)
#i1.8990527646046183e+208
>
結果は「近似値」
である
188:3
Unlocked
not running

数列の和

- 数列の和を求める高階関数 series を作る
  - 数値 n と関数 f から，(f 1), (f 2), …. (f n) の和を求める　⇒　(f i) は数列の第 i 項

つまり $\sum_{k=1}^{n} f(k)$

- series を使って，簡単な数列の和を求めてみる
  - 2 乗の和 $1^2 + 2^2 + \ldots + n^2$
  - 3 乗の和 $1^3 + 2^3 + \ldots + n^3$

数列の和

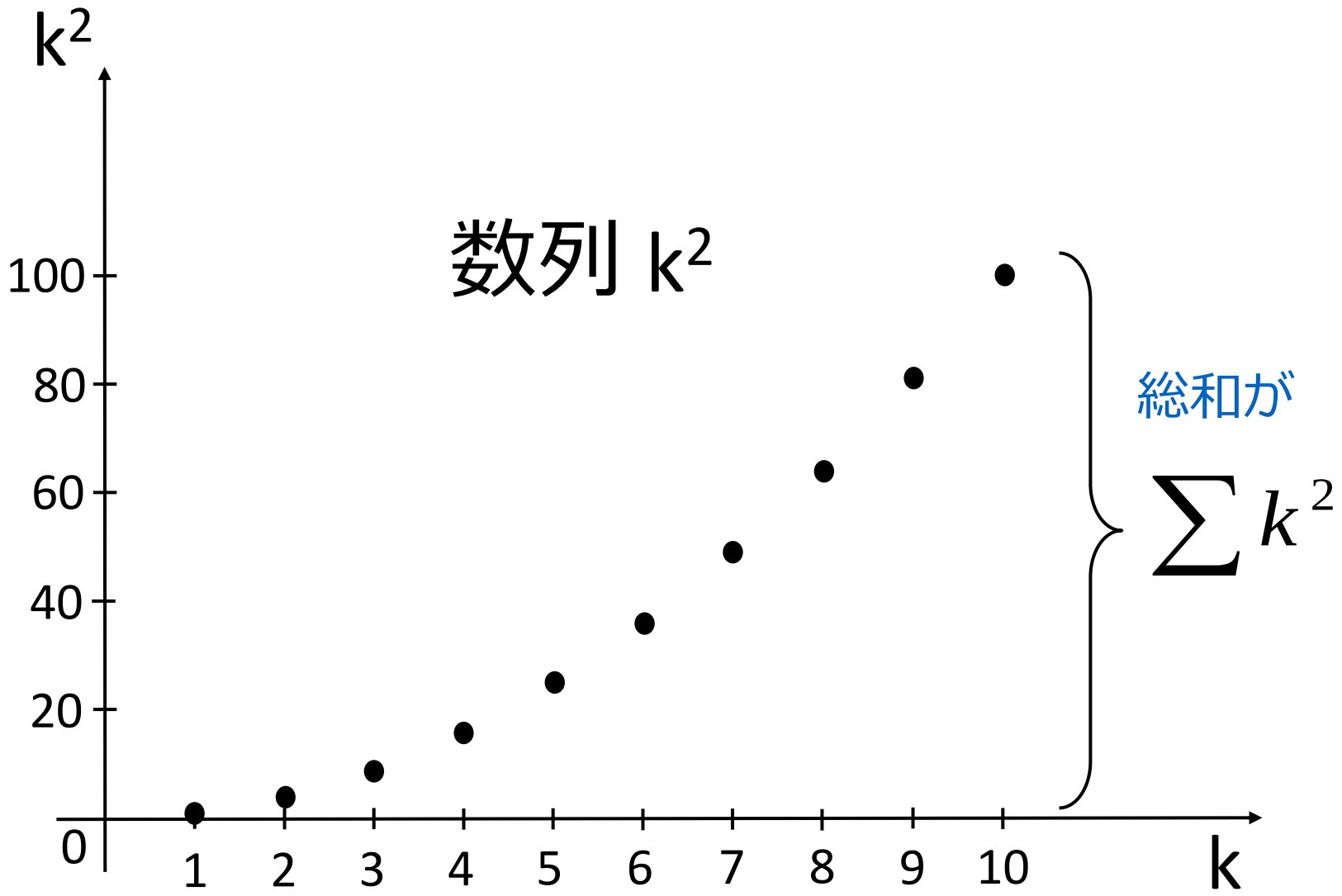
- n = 1 のとき

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

- n > 1 のとき

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

数列の和



「例題３．数列の和」の手順

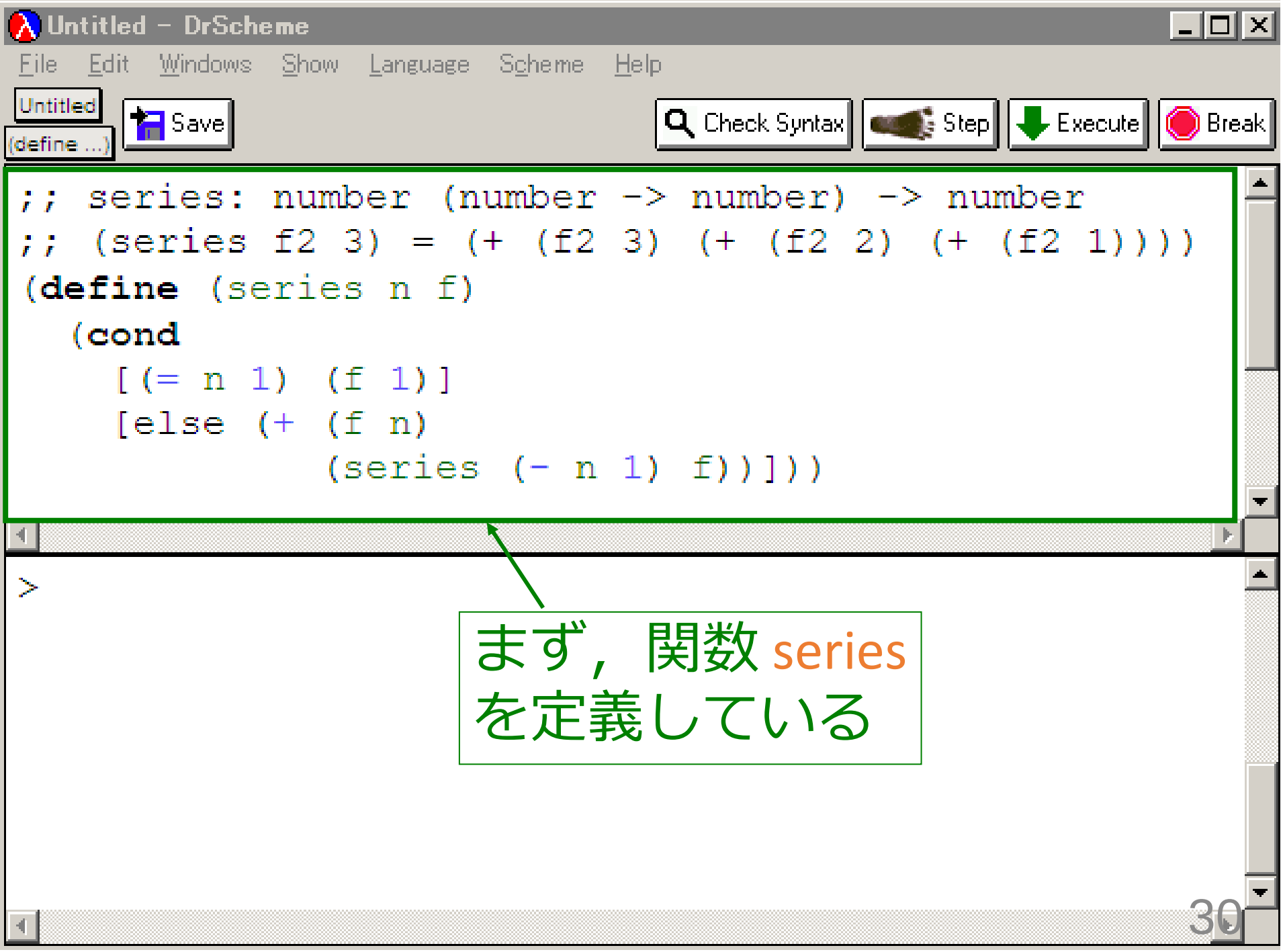
1. 次を「定義用ウインドウ」で，実行しなさい
   - 入力した後に，Execute ボタンを押す

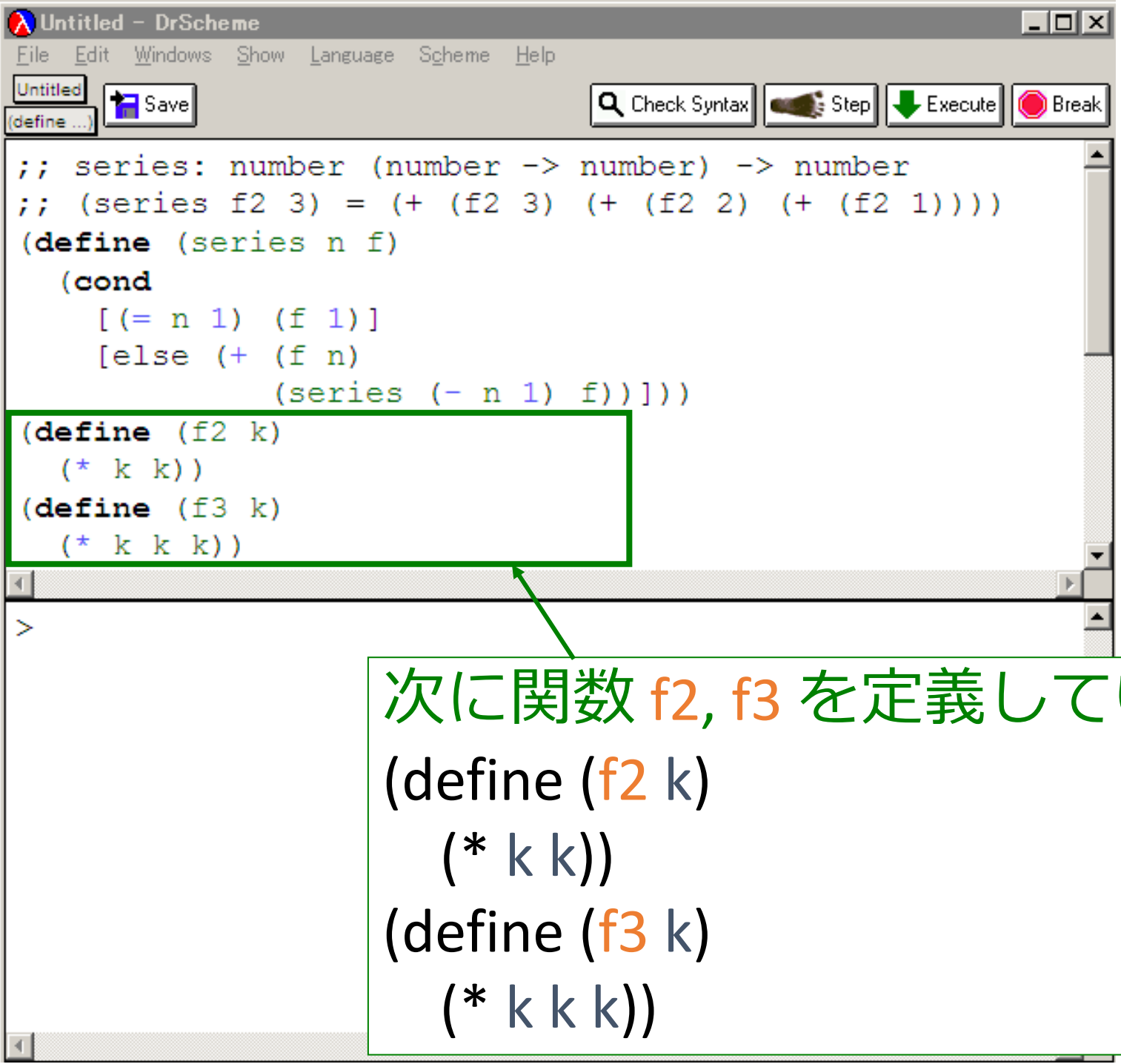
```
;; series: number (number -> number) -> number
;; (series f2 3) = (+ (f2 3) (+ (f2 2) (+ (f2 1))))
(define (series n f)
   (cond
      [(= n 1) (f 1)]
      [else (+ (f n)
               (series (- n 1) f))]))
(define (f2 k)
   (* k k))
(define (f3 k)
   (* k k k))
```

2. その後，次を「実行用ウインドウ」で実行しなさい

```
(series 3 f2)
(series 4 f3)
```

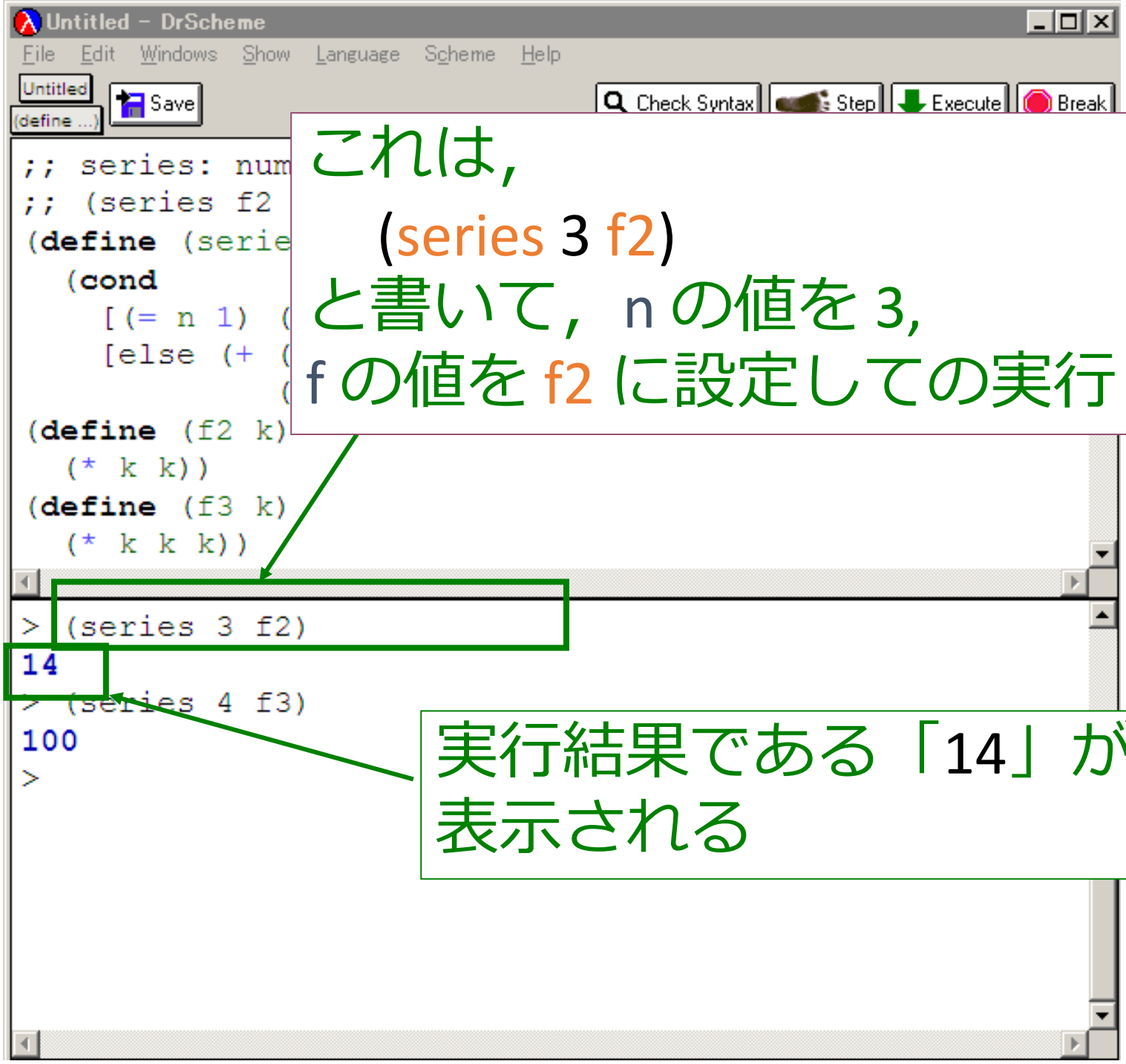
☆　次は，例題４に進んでください


Untitled – DrScheme
File Edit Windows Show Language Scheme Help
Untitled
(define ...)
Save
Check Syntax
Step
Execute
Break
;; series: number (number -> number) -> number
;; (series f2 3) = (+ (f2 3) (+ (f2 2) (+ (f2 1))))
(define (series n f)
(cond
[(= n 1) (f 1)]
[else (+ (f n)
(series (- n 1) f))]))
>
まず，関数 series
を定義している



次に関数 f2, f3 を定義している

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



これは,

(series 3 f2)

と書いて，n の値を 3,
f の値を f2 に設定しての実行

実行結果である「14」が表示される

入力と出力

入力は１つの数値と関数

出力は数値

series は，関数を入力とするような関数（つまり高階関数）

```
;; series: number (number -> number) -> number
;; (series f2 3) = (+ (f2 3) (+ (f2 2) (+ (f2 1))))
(define (series n f)
    (cond
        [(= n 1) (f 1)]
        [else (+ (f n)
                 (series (- n 1) f))]))
```

関数 f が，
「series」の入力になっている

(series 3 f2) から 14 に至る過程の概略

(series 3 f2)　最初の式

= …

= (+ (f2 3) (series 2 f2))

= …

= (+ (f2 3) (+ (f2 2) (series 1 f2)))

= …

= (+ (f2 3) (+ (f2 2) (+ (f2 1))))

コンピュータ内部での計算

= …

= 14　実行結果

但し，f2 は
(define (f2 k)
(* k k))

(series 3 f2) から (+ (f2 3) (series 2 f2)) に至る過程

```
(series 3 f2)
= …
= (+ (f2 3) (series 2 f2))
= …
= (+ (f2 3) (+ (f2 2) (series
= …
= (+ (f2 3) (+ (f2 2) (+ (f2 1
= …
= 14
```

この部分は

```
(series 3 f2)
= (cond
    [(= 3 1) (f2 1)]
    [else (+ (f2 3)
             (series (- 3 1) f2))])
= (cond
    [false (f2 1)]
    [else (+ (f2 3)
             (series (- 3 1) f2))])
= (+ (f2 3) (series (- 3 1) f2))
= (+ (f2 3) (series 2 f2))
```

(series 3 f2) から (+ (f2 3) (series 2 f2)) に至る過程

(series 3 f2)

= …

= (+ (f2 3) (series 2 f2))

= …

=

=

=

=

=

この部分は

(series 3 f2)

= (cond
  [(= 3 1) (f2 1)]
  [else (+ (f2 3)
    (series (- 3 1) f2))])

= (cond

)])

これは,

```
(define (series n f)
  (cond
    [(= n 1) (f 1)]
    [else (+ (f n)
             (series (- n 1) f))]))
```

の n を 3 で，f を f2 で置き換えたもの

例題４．数列の和のリスト

- 例題３の「series」を使って，数列の和のリストを作る高階関数 series-iter を作る
  - 数値 n と関数 f から，

    (f 1), (f 2), …. (f n) の和　　・・・　n 項までの和

    (f 1), (f 2), …. (f (- n 1)) の和　・・・　n-1 項までの和

    ...

    (f 1) の値　　・・・　1 項までの和

    のリストを作る
- series-iter を使って，次の数列の和が Π²/6 に近づく様子を観察する

$$\underbrace{\sum_{n=1}^{\infty}\frac{1}{n^2}=1+\frac{1}{2^2}+\frac{1}{3^2}+\cdots}_{\text{数列の和}}=\frac{\pi^2}{6}$$

「例題４．数列の和のリスト」の手順

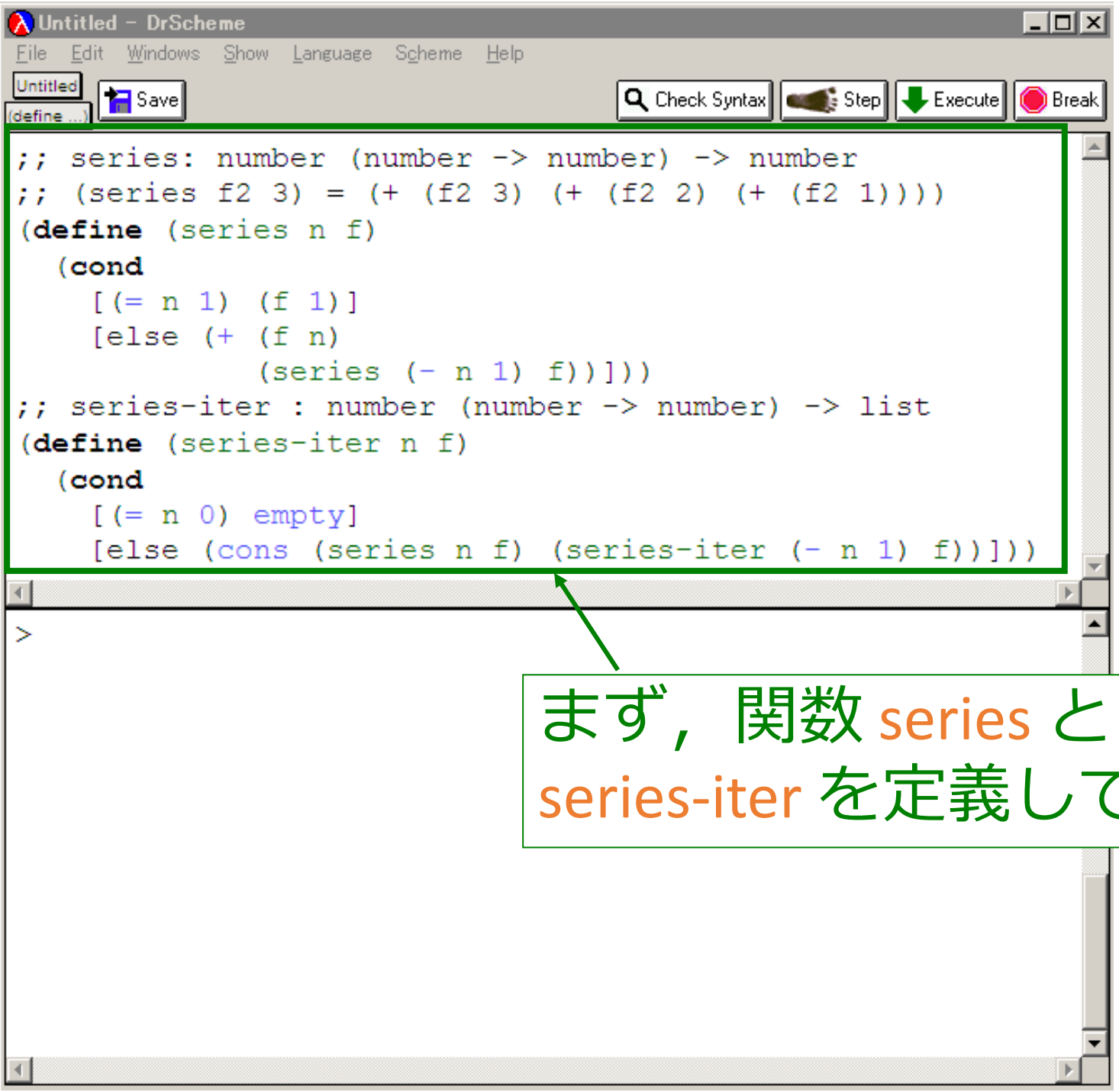
1. 次を「定義用ウインドウ」で，実行しなさい
   - 入力した後に，Execute ボタンを押す

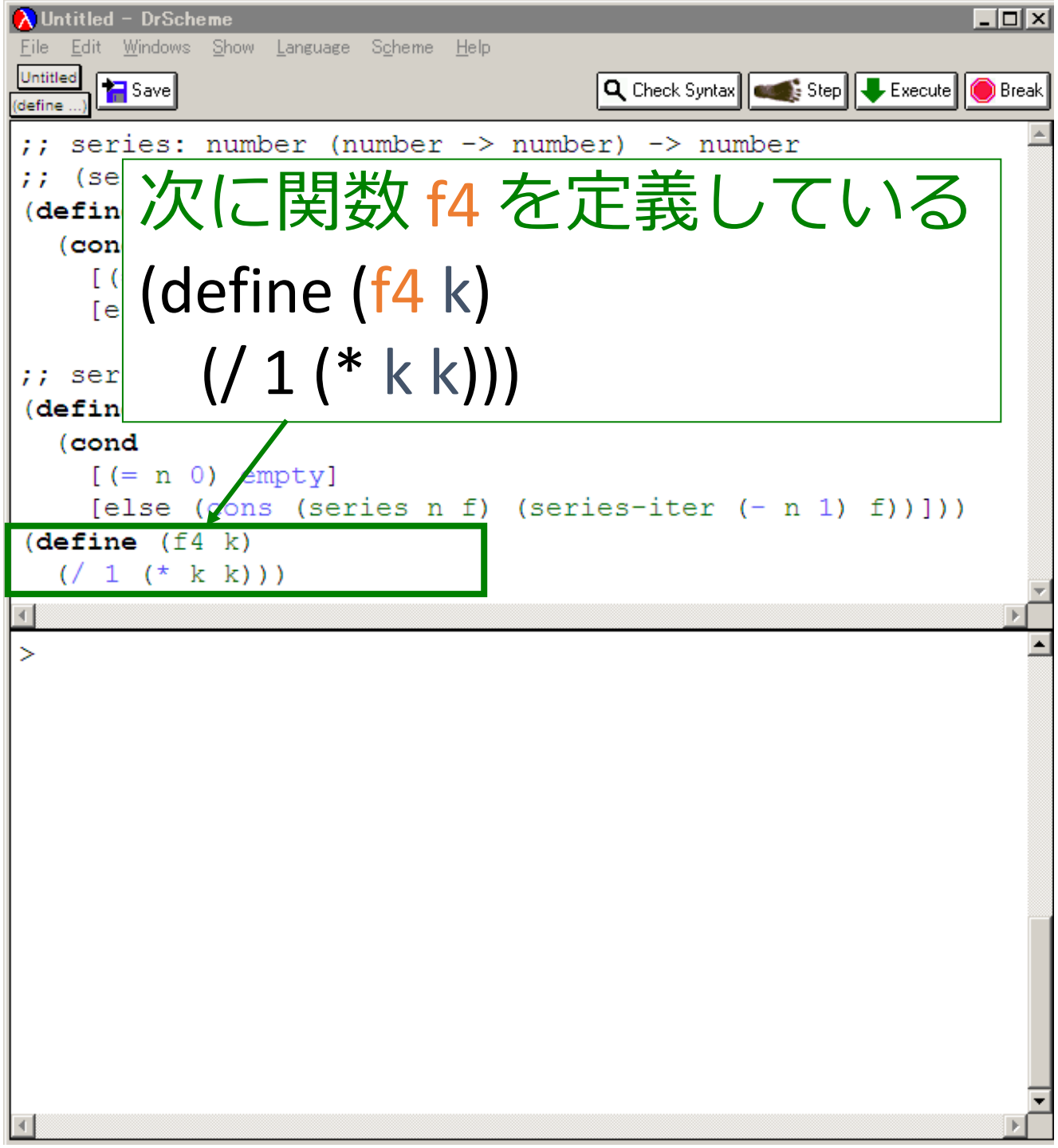
```
;; series: number (number -> number) -> number
;; (series f2 3) = (+ (f2 3) (+ (f2 2) (+ (f2 1))))
(define (series n f)
   (cond
      [(= n 1) (f 1)]
      [else (+ (f n)
               (series (- n 1) f))]))
;; series-iter : number (number -> number) -> list
(define (series-iter n f)
   (cond
      [(= n 0) empty]
      [else (cons (series n f) (series-iter (- n 1) f))]))
(define (f4 k)
   (/ 1 (* k k)))
```

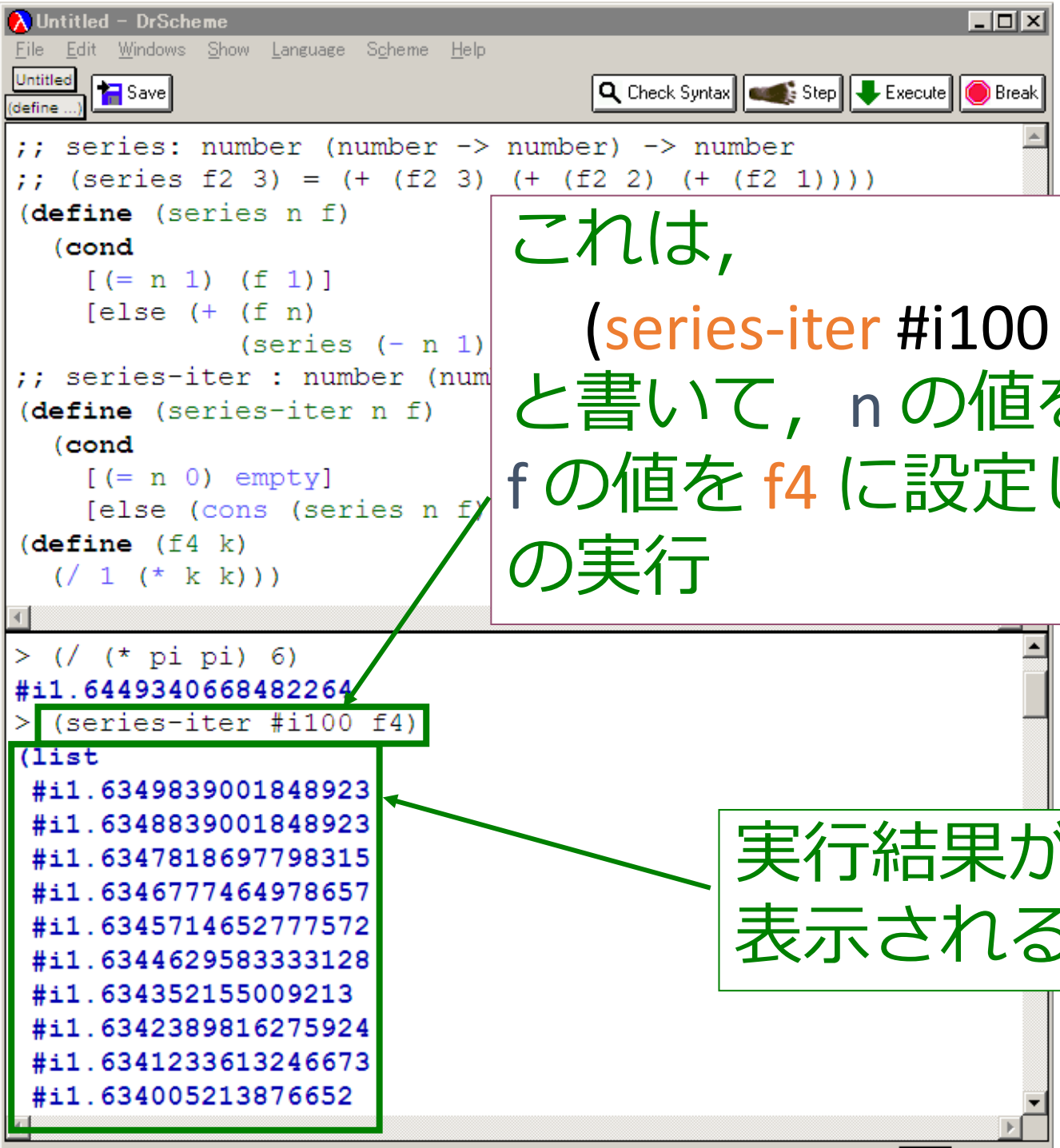
2. その後，次を「実行用ウインドウ」で実行しなさい

```
(series-iter #i100 f4)
```

☆　次は，例題５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
;; series: number (number -> number) -> number
;; (series f2 3) = (+ (f2 3) (+ (f2 2) (+ (f2 1))))
(define (series n f)
(cond
[(= n 1) (f 1)]
[else (+ (f n)
(series (- n 1) f))]))
;; series-iter : number (number -> number) -> list
(define (series-iter n f)
(cond
[(= n 0) empty]
[else (cons (series n f) (series-iter (- n 1) f))]))
>
まず，関数 series と
series-iter を定義している


Untitled – DrScheme
File Edit Windows Show Language Scheme Help
Untitled
(define ...)
Save
Check Syntax
Step
Execute
Break
;; series: number (number -> number) -> number
;; (se
(defin
(con
[(
[e
;; ser
(defin
(cond
[(= n 0) empty]
[else (cons (series n f) (series-iter (- n 1) f))]))
(define (f4 k)
(/ 1 (* k k)))
>
次に関数 f4 を定義している
(define (f4 k)
(/ 1 (* k k)))



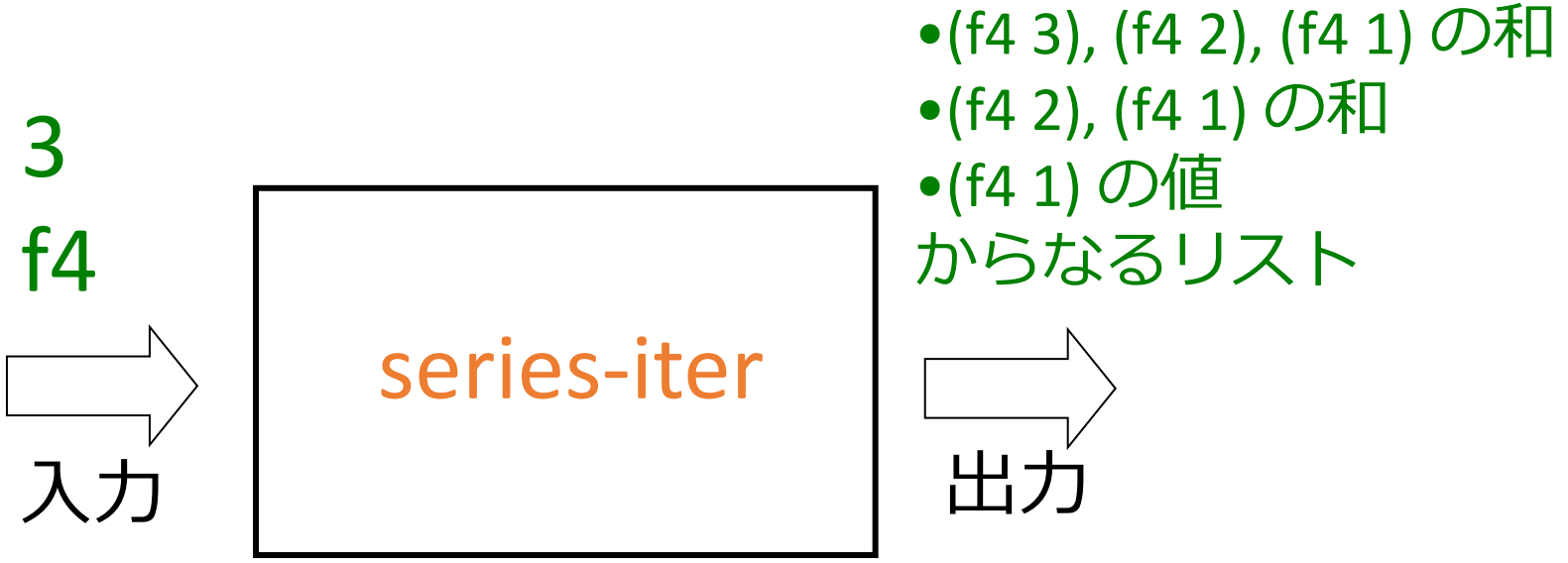
これは,

(series-iter #i100 f4)

と書いて，n の値を #i100,
f の値を f4 に設定して
の実行

実行結果が
表示される

入力と出力



入力は１つの数値と関数

出力は数値のリスト

series-iter は，関数を入力とするような関数（つまり高階関数）

```
;; series: number (number -> number) -> number
;; (series f2 3) = (+ (f2 3) (+ (f2 2) (+ (f2 1))))
(define (series n f)
    (cond
        [(= n 1) (f 1)]
        [else (+ (f n)
                 (series (- n 1) f))]))
;; series-iter : number (number -> number) -> list
(define (series-iter n f)
    (cond
        [(= n 0) empty]
        [else (cons (series n f) (series-iter (- n 1) f))]))
```

数列の和のリスト

1. n = 0 ならば： → 終了条件

   empty → 自明な解

2. そうで無ければ：
   - 長さが n-1 の「数列の和のリスト」を作り，その先頭に「第 n 項」をつなげる
   - この第 n 項は，(series n f) で得られる

例題５．べき級数

- べき級数を求める高階関数 taylor-series を作る
  - 数値 x, n と関数 g から，(g 0), (* (g 1) (expt x 1)), ..., (* (g n) (expt x n)) の和を求める

つまり $\sum_{k=0}^{n} g(k) \cdot x^k$

- taylor-series を使って，簡単なべき級数 $1 + x + 2x^2 + \ldots + nx^n$ を求めてみる

「例題５．べき級数」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，Execute ボタンを押す

```
;; taylor-series: number number (number -> number)
;;                      -> number
;; (taylor-series 2 3 g) =
;;       (+ (* (g 3) (expt 2 3)
;;             (+ (* (g 2) (expt 2 2))
;;                (+ (* (g 1) (expt 2 1))
;;                   (g 0))))
(define (taylor-series x n g)
   (cond
      [(= n 0) (g 0)]
      [else (+ (* (g n) (expt x n))
               (taylor-series x (- n 1) g))]))
(define (g1 k)
   k)
```

2. その後，次を「実行用ウインドウ」で実行しなさい

```
(taylor-series 2 3 g1)
(taylor-series 2 4 g1)
```

☆　次は，例題６に進んでください

級数

- 級数とは：　　数列の和
- べき級数とは：　　$\sum_{k=0}^{n} g(k) \cdot x^k$ の形をした級数

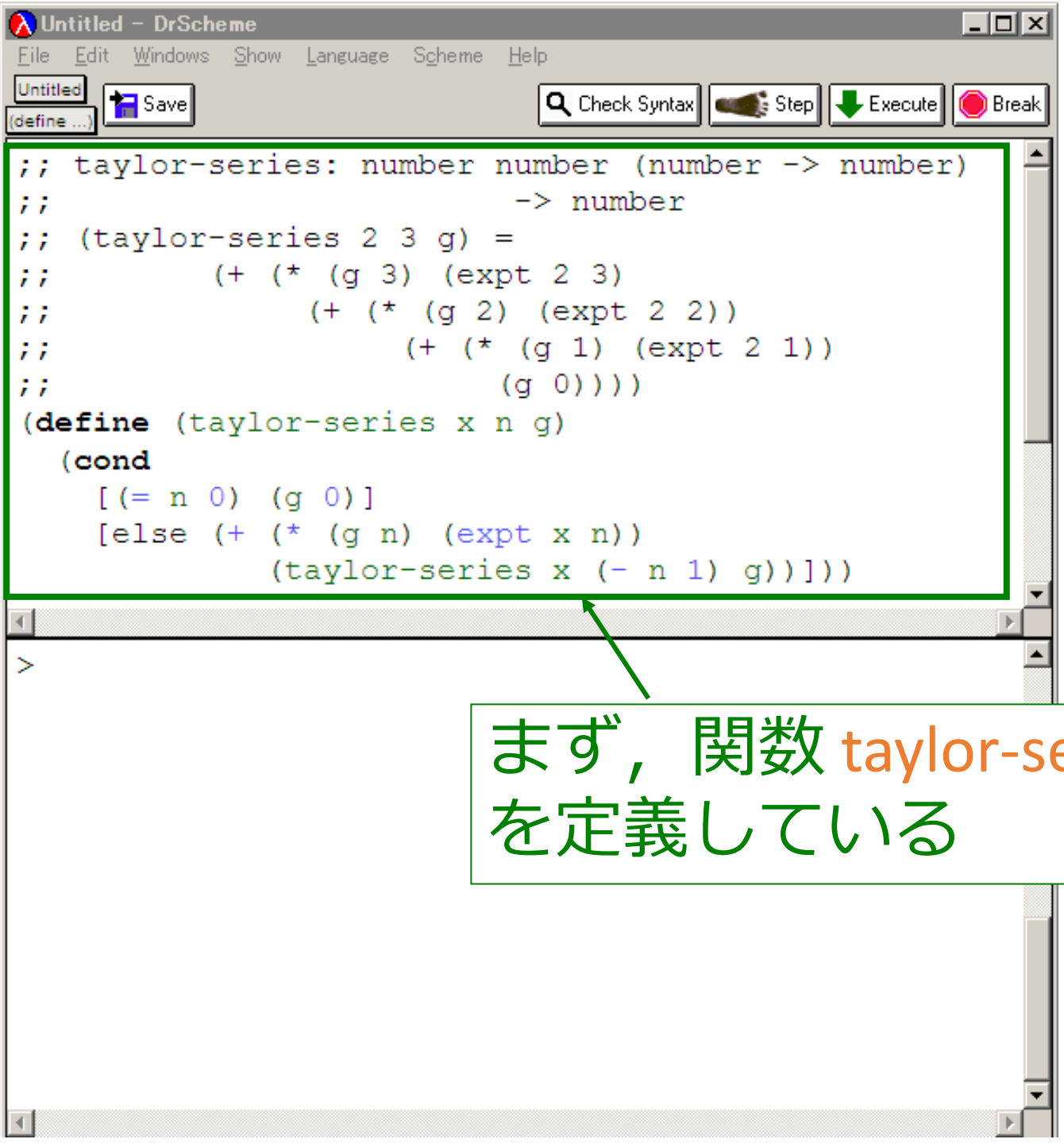
べき級数

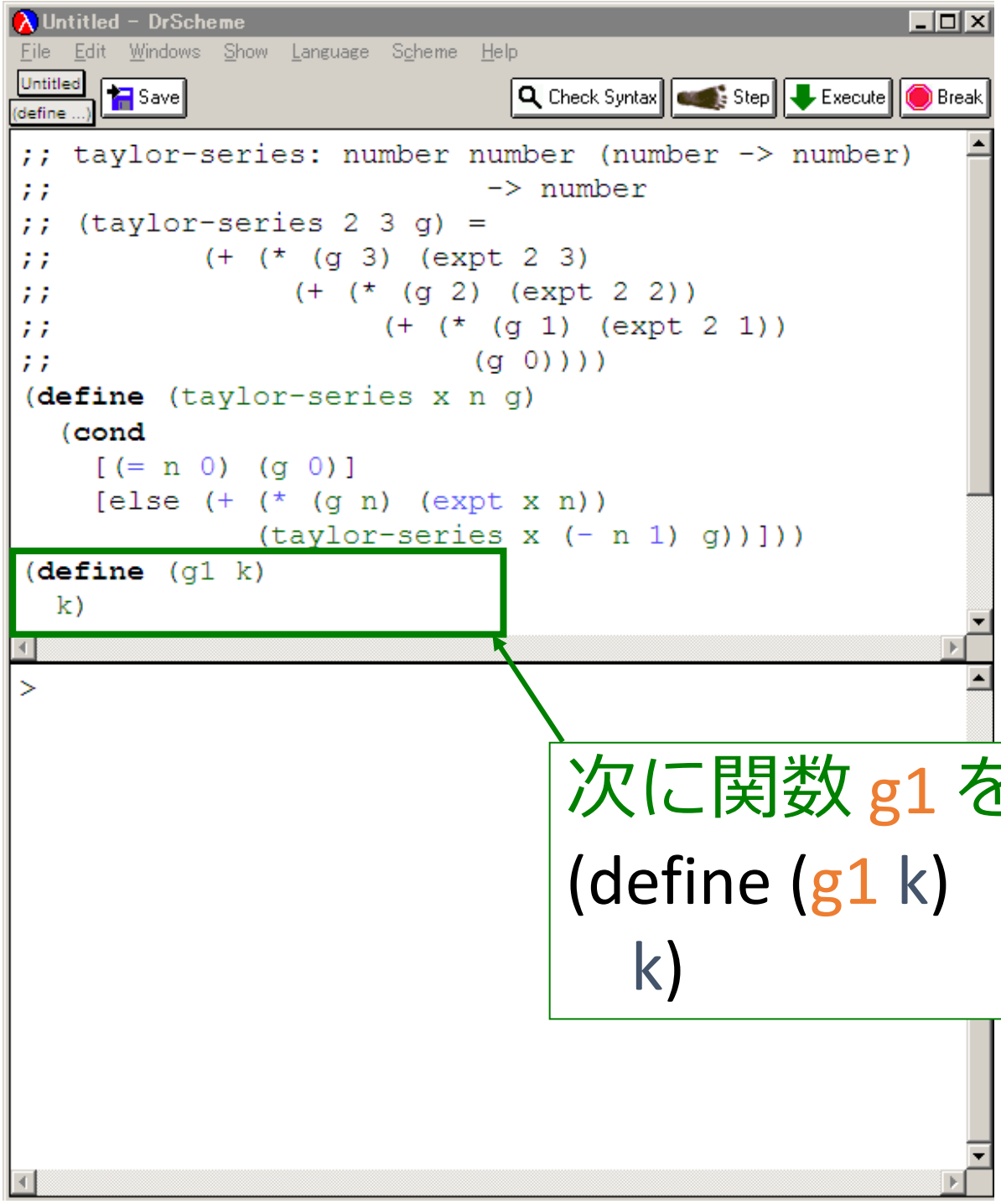
- n = 0 のとき

$$\sum_{k=0}^{n} g(k) \cdot x^k = g(0)$$

- n > 0 のとき

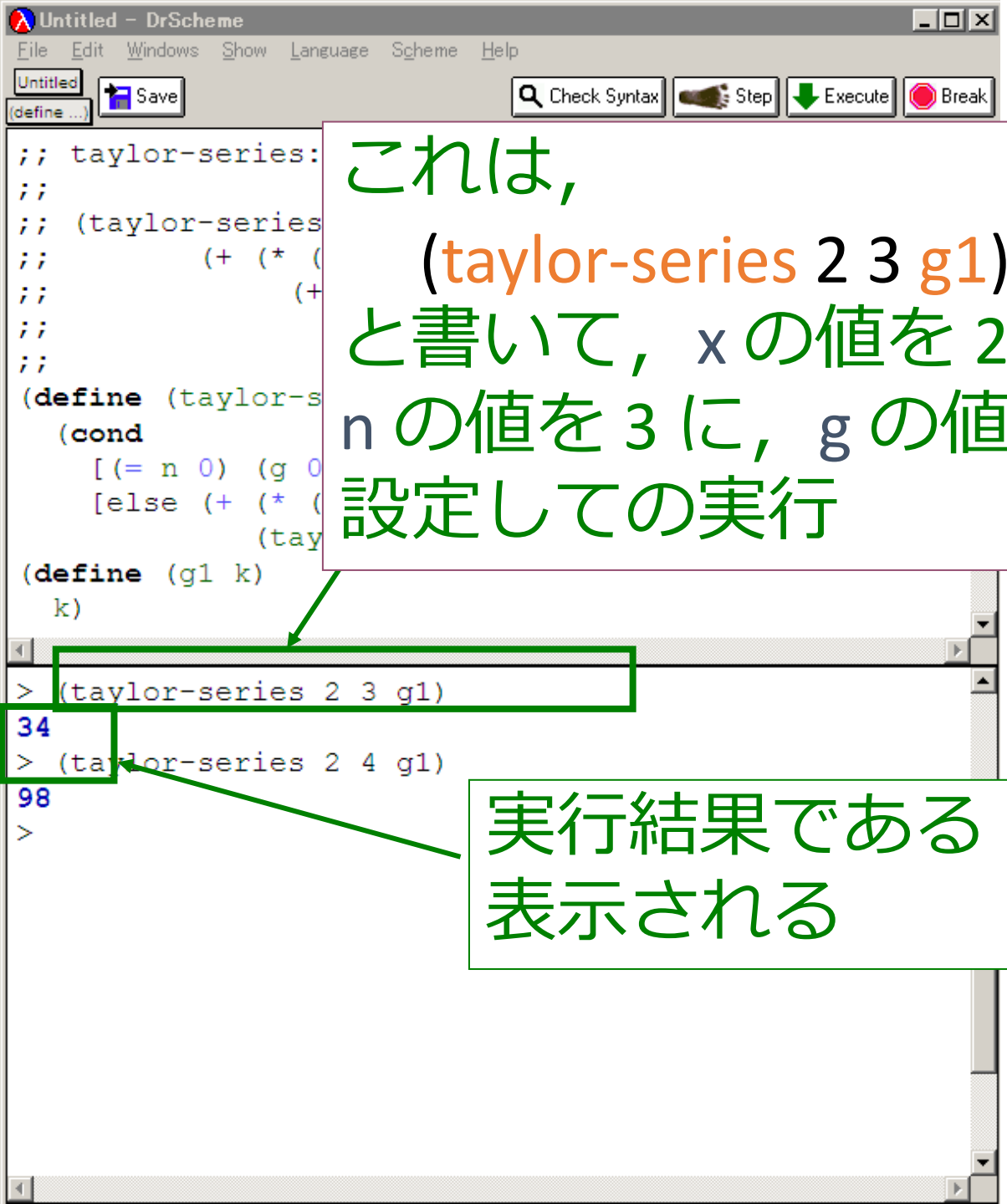
$$\sum_{k=0}^{n} g(k) \cdot x^k = g(n) \cdot x^n + \sum_{k=0}^{n-1} g(k) \cdot x^k$$


Untitled – DrScheme
File Edit Windows Show Language Scheme Help
Untitled
(define ...)
Save
Check Syntax
Step
Execute
Break
;; taylor-series: number number (number -> number)
;;   -> number
;; (taylor-series 2 3 g) =
;;   (+ (* (g 3) (expt 2 3)
;;   (+ (* (g 2) (expt 2 2))
;;   (+ (* (g 1) (expt 2 1))
;;   (g 0))))
(define (taylor-series x n g)
  (cond
    [(= n 0) (g 0)]
    [else (+ (* (g n) (expt x n))
             (taylor-series x (- n 1) g))]))
>
まず，関数 taylor-series を定義している



次に関数 g1 を定義している

```
(define (g1 k)
    k)
```



これは,

(taylor-series 2 3 g1)

と書いて，x の値を 2 に，
n の値を 3 に，g の値を g1 に
設定しての実行

実行結果である「34」が
表示される

入力と出力

入力は２つの数値と関数

出力は数値

taylor-series は，関数を入力とするような関数
= 高階関数

taylor-series 関数

```
;; taylor-series: number number (number -> number)
;;                      -> number
;; (taylor-series 2 3 g) =
;;       (+ (* (g 3) (expt 2 3)
;;             (+ (* (g 2) (expt 2 2))
;;                   (+ (* (g 1) (expt 2 1))
;;                         (g 0))))
(define (taylor-series x n g)
    (cond
        [(= n 0) (g 0)]
        [else (+ (* (g n) (expt x n))
                 (taylor-series x (- n 1) g))]))
```

(taylor-series 2 3 g1) から 34 に至る過程の概略

(taylor-series 2 3 g1)

最初の式

= …

= (+ (* (g1 3) (expt 2 3)) (taylor-series 2 2 g1))

= …

= (+ (* (g1 3) (expt 2 3)) (+ (* (g1 2) (expt 2 2)) (taylor-series 2 1 g1)))

= …

= (+ (* (g1 3) (expt 2 3)) (+ (* (g1 2) (expt 2 2)) (+ (* (g1 1) (expt 2 1)) (taylor-series 2 0 g1))))

= …

= (+ (* (g1 3) (expt 2 3)) (+ (* (g1 2) (expt 2 2)) (+ (* (g1 1) (expt 2 1)) (g1 2 0))))

コンピュータ内部での計算

= ...

= 34

実行結果

但し，g1 は
(define (g1 k)
k)

(taylor-series 2 3 g1) から 34 に至る過程の概略

(taylor-series 2 3 g1)

= …

= (+ (* (g1 3) (expt 2 3)) (taylor-series 2 2 g1))

= …

= (+ (* (g1 3) (expt 2 3

= …

= (+ (* (g1 3) (expt 2 3)) (+ (* (g1
series 2 0 g1))))

= …

= (+ (* (g1 3) (expt 2 3)) (+ (* (g1

= ...

= 34

この部分は

(taylor-series 2 3 g)

= (cond
[(= 3 0) (g1 0)]
[else (+ (* (g1 3) (expt 2 3))
(taylor-series 2 (- 3 1) g1))])

= (cond
[false (g1 0)]
[else (+ (* (g1 3) (expt 2 3))
(taylor-series 2 (- 3 1) g1))])

= (+ (* (g1 3) (expt 2 3))
(taylor-series 2 (- 3 1) g1))

= (+ (* (g1 3) (expt 2 3))
(taylor-series 2 2 g1))

(taylor-series 2 3 g1) から 34 に至る過程の概略


```
(taylor-series 2 3 g1)
= …
= (+ (* (g1 3) (expt 2 3)) (taylor-series 2 2 g1))
= …
= (+ (* (g1 3) (expt 2 3)) (+ (* (g1
= …
= (+ (* (g1 3) (expt 2 3)) (+ (* (g1
```

この部分は

```
(taylor-series 2 3 g)
= (cond
    [(= 3 0) (g1 0)]
    [else (+ (* (g1 3) (expt 2 3))
             (taylor-series 2 (- 3 1) g1))])
= (cond
```

これは，

```
(define (taylor-series x n g)
  (cond
    [(= n 0) (g 0)]
    [else (+ (* (g n) (expt x n))
             (taylor-series x (- n 1) g))]))
```

の x を 2 で，n を 3 で，g を g1 で置き換えたもの

テーラー展開の式

関数 f が，0 の近傍で定義されていて，
点 0 でn 階微分可能であるとき

$$f(x) = f(0) + \frac{f'(0)}{1!}x + \frac{f''(0)}{2!}x^2 + \cdots + \frac{f^{(n-1)}}{(n-1)!}x^{n-1} + \frac{f^{n}(\xi)}{n!}x^n$$

但し，$\xi = \theta \cdot x, 0 < \theta < 1$

テーラー展開

$$e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!} = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$ 収束半径：∞

$$\sin x = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)!} = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$ 収束半径：∞

$$\cos x = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \cdots$$ 収束半径：∞

$$\log_e(x+1) = \sum_{i=0}^{\infty} (-1)^i \frac{x^{i+1}}{i+1} = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots$$ 収束半径： 1

$$\tan^{-1}(x) = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{2i+1} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots$$ 収束半径： 1

例題６．指数関数のテーラー展開

- 例題５の「taylor-series」を使って，指数関数のテーラー展開の値を求める

$$e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!} = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

「例題６．指数関数のテーラー展開」の手順 (1/2)

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

```
;; taylor-series: number number (number -> number)
;;                    -> number
;; (taylor-series 2 3 g) =
;;       (+ (* (g 3) (expt 2 3)
;;           (+ (* (g 2) (expt 2 2))
;;              (+ (* (g 1) (expt 2 1))
;;                 (g 0))))
(define (taylor-series x n g)
   (cond
      [(= n 0) (g 0)]
      [else (+ (* (g n) (expt x n))
               (taylor-series x (- n 1) g))]))
(define (factorial k)
   (cond
      [(= k 0) 1]
      [else (* k (factorial (- k 1)))]))
(define (exp-term k)
  (/ 1 (factorial k)))
(define (my-exp x)
  (taylor-series x 20 exp-term))
```

例題５と同じ

☆　次ページに進んでください

「例題６．指数関数のテーラー展開」の手順 (2/2)

2. その後，次を「実行用ウインドウ」で実行しなさい

```
(my-exp 0)
(my-exp 1)
(my-exp 2)
(my-exp 3)
(my-exp #i1)
(my-exp #i2)
(my-exp #i3)
```

☆　次は，例題７に進んでください

File Edit Windows Show Language Scheme Help

Untitled (define ...) Save | Check Syntax | Step | Execute | Break

```
;; taylor-series: number number (number -> number)
;;                              -> number
;; (taylor-series 2 3 g) =
;;         (+ (* (g 3) (expt 2 3)
;;               (+ (* (g 2) (expt 2 2))
;;                     (+ (* (g 1) (expt 2 1))
;;                           (g 0))))
(define (taylor-series x n g)
  (cond
    [(= n 0) (g 0)]
    [else (+ (* (g n) (expt x n))
             (taylor-series x (- n 1) g))]))
(define (factorial k)
    (cond
      [(= k 0) 1]
      [else (* k (factorial (- k 1)))]))
(define (exp-term k)
  (/ 1 (factorial k)))
(define (my-exp x)
  (taylor-series x 20 exp-term))
```

```
Welcome to DrScheme, version 103p1.
Language: Advanced Student.
Teachpack: C:\Program Files\PLT\teachpack\htdp\draw.ss.
> (my-exp 0)
1
> (my-exp 1)
661331331924808000l/2432902008176640000
> (my-exp 2)
68576238333199/9280784638125
> (my-exp 3)
7447971820629961/370812682240000
> (my-exp #i1)
#i2.7182818284590455
> (my-exp #i2)
#i7.389056098930604
> (my-exp #i3)
#i20.085536922950844
>
```

```
(define (taylor-series x n g)
  (cond
    [(= n 0) (g 0)]
    [else (+ (* (g n) (expt x n))
             (taylor-series x (- n 1) g))]))
```

べき級数
（例題 5 と同じ）

```
(define (factorial k)
  (cond
    [(= k 0) 1]
    [else (* k (factorial (- k 1)))]))
```

k の階乗
を求める

```
(define (exp-term k)
  (/ 1 (factorial k)))
```

係数

```
(define (my-exp x)
  (taylor-series x 20 exp-term))
```

taylor-series を使って，
e^x （の近似値）を計算

指数関数のテーラー展開

$$e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!} = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

```
(define (exp-term k)
  (/ 1 (factorial k)))
```

my-exp での誤差

```
(define (my-exp x)
  (taylor-series x 20 g2))
```

$$e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!} = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^{20}}{20!} + \frac{x^{21}}{21!} + \cdots$$

$x^{20}/20!$ の項まで計算

$x^{21}/21!$ 以降の部分が誤差

例題７．cos 関数のテーラー展開

- 例題５の「taylor-series」を使って，cos 関数のテーラー展開の値を求める

$$\cos x = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \cdots$$

「例題７．cos 関数のテーラー展開」の手順 (1/2)

1. 次を「定義用ウインドウ」で，実行しなさい

```
;; taylor-series: number number (number -> number)
;;                    -> number
;; (taylor-series 2 3 g) =
;;       (+ (* (g 3) (expt 2 3)
;;           (+ (* (g 2) (expt 2 2))
;;               (+ (* (g 1) (expt 2 1))
;;                   (g 0))))
(define (taylor-series x n g)
   (cond
      [(= n 0) (g 0)]
      [else (+ (* (g n) (expt x n))
               (taylor-series x (- n 1) g))]))
(define (factorial k)
   (cond
      [(= k 0) 1]
      [else (* k (factorial (- k 1)))]))
(define (cos-term k)
  (cond
    [(odd? k) 0]
    [else
     (cond
       [(= (remainder k 4) 0) (/ 1 (factorial k))]
       [else (/ -1 (factorial k))])]))
(define (my-cos x)
  (taylor-series x 20 cos-term))
```

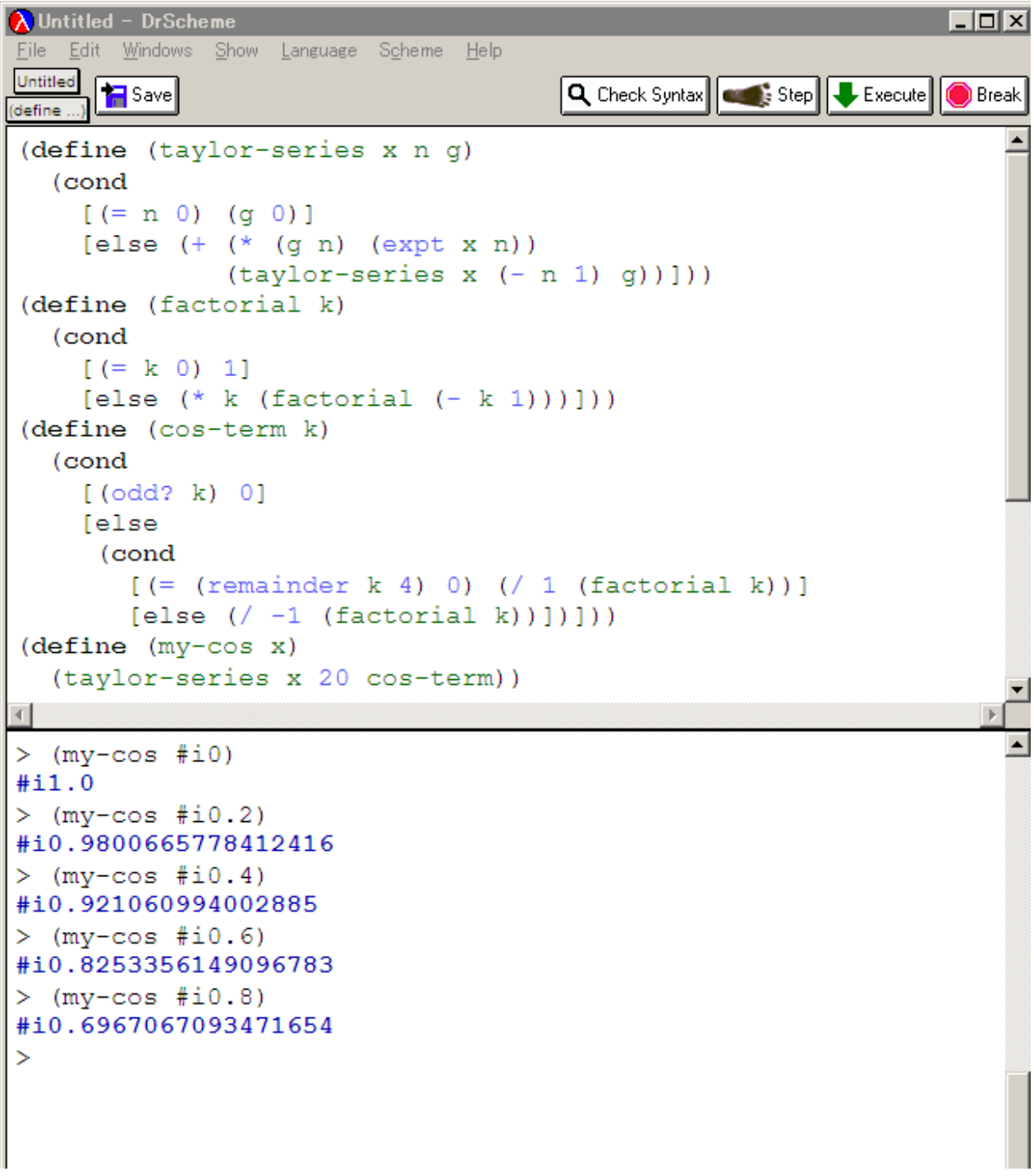
例題６と同じ

「例題７．指数関数のテーラー展開」の手順 (2/2)

2. その後，次を「実行用ウインドウ」で実行しなさい

```
(my-cos #i0)
(my-cos #i0.2)
(my-cos #i0.4)
(my-cos #i0.6)
(my-cos #i0.8)
```

☆　次は，例題８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 (taylor-series x n g)
(cond
[(= n 0) (g 0)]
[else (+ (* (g n) (expt x n))
(taylor-series x (- n 1) g))]))
(define (factorial k)
(cond
[(= k 0) 1]
[else (* k (factorial (- k 1)))]))
(define (cos-term k)
(cond
[(odd? k) 0]
[else
(cond
[(= (remainder k 4) 0) (/ 1 (factorial k))]
[else (/ -1 (factorial k))])]))
(define (my-cos x)
(taylor-series x 20 cos-term))
> (my-cos #i0)
#i1.0
> (my-cos #i0.2)
#i0.980066577841241 6
> (my-cos #i0.4)
#i0.921060994002885
> (my-cos #i0.6)
#i0.8253356149096783
> (my-cos #i0.8)
#i0.6967067093471654
>

```
(define (taylor-series x n g)
  (cond
    [(= n 0) (g 0)]
    [else (+ (* (g n) (expt x n))
             (taylor-series x (- n 1) g))]))
```

べき級数
（例題５と同じ）

```
(define (factorial k)
  (cond
    [(= k 0) 1]
    [else (* k (factorial (- k 1)))]))
```

k の階乗
を求める
（例題６と同じ）

```
(define (cos-term k)
  (cond
    [(odd? k) 0]
    [else
     (cond
       [(= (remainder k 4) 0) (/ 1 (factorial k))]
       [else (/ -1 (factorial k))])]))
```

係数

```
(define (my-cos x)
  (taylor-series x 20 cos-term))
```

taylor-series を使って，
cos x （の近似値）を計算

cos 関数のテーラー展開

$$\cos x = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \cdots$$

$$= 1 + 0 \cdot x - \frac{x^2}{2!} + 0 \cdot x^3 + \frac{x^4}{4!} + 0 \cdot x^5 - \frac{x^6}{6!} + \cdot x^7 + \frac{x^8}{8!} - \cdots$$

```
(define (cos-term k)
  (cond
    [(odd? k) 0]
    [else
     (cond
       [(= (remainder k 4) 0) (/ 1 (factorial k))]
       [else (/ -1 (factorial k))])]))
```

odd? は
奇数ならば true

例題８．対数関数のテーラー展開

- 例題５の「taylor-series」を使って，対数関数のテーラー展開の値を求める

$$\log_e(x+1) = \sum_{i=0}^{\infty} (-1)^i \frac{x^{i+1}}{i+1} = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots$$

「例題８．対数関数のテーラー展開」の手順 (1/2)

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

```
;; taylor-series: number number (number -> number)
;;                    -> number
;; (taylor-series 2 3 g) =
;;       (+ (* (g 3) (expt 2 3)
;;           (+ (* (g 2) (expt 2 2))
;;              (+ (* (g 1) (expt 2 1))
;;                 (g 0))))
(define (taylor-series x n g)
   (cond
      [(= n 0) (g 0)]
      [else (+ (* (g n) (expt x n))
               (taylor-series x (- n 1) g))]))
(define (log-term k)
  (cond
    [(= k 0) 0]
    [else (/ (expt -1 (- k 1)) k)]))
(define (my-log x)
  (taylor-series (- x 1) 20 log-term))
```

例題７と同じ

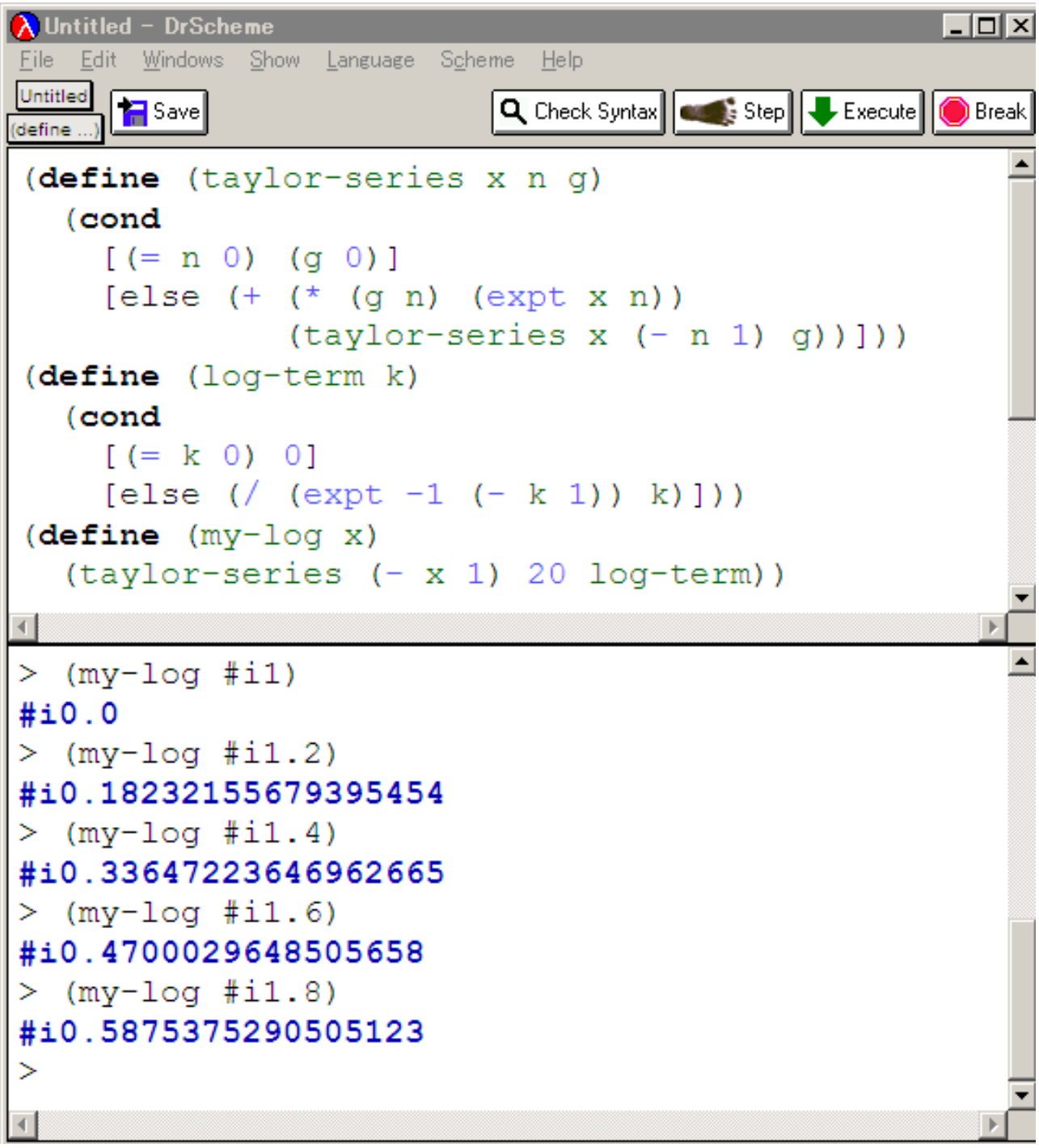
☆　次ページに進んでください

「例題８．対数関数のテーラー展開」の手順 (2/2)

2. その後，次を「実行用ウインドウ」で実行しなさい

```
(my-log #i1)
(my-log #i1.2)
(my-log #i1.4)
(my-log #i1.6)
(my-log #i1.8)
```

☆　次は，例題９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 (taylor-series x n g)
(cond
[(= n 0) (g 0)]
[else (+ (* (g n) (expt x n))
(taylor-series x (- n 1) g))]))
(define (log-term k)
(cond
[(= k 0) 0]
[else (/ (expt -1 (- k 1)) k)]))
(define (my-log x)
(taylor-series (- x 1) 20 log-term))
> (my-log #i1)
#i0.0
> (my-log #i1.2)
#i0.1823215567939454
> (my-log #i1.4)
#i0.3364722366962665
> (my-log #i1.6)
#i0.4700029648505658
> (my-log #i1.8)
#i0.587537529050512 3
>

```
(define (taylor-series x n g)
  (cond
    [(= n 0) (g 0)]
    [else (+ (* (g n) (expt x n))
             (taylor-series x (- n 1) g))]))
```

べき級数
（例題５と同じ）

```
(define (log-term k)
  (cond
    [(= k 0) 0]
    [else (/ (expt -1 (- k 1)) k)]))
```

係数

```
(define (my-log x)
  (taylor-series (- x 1) 20 log-term))
```

taylor-series を使って，
log x （の近似値）を計算

対数関数のテーラー展開

$$\log_e(x+1) = \sum_{i=0}^{\infty}(-1)^i \frac{x^{i+1}}{i+1} = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots$$

$$= 0 + x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} \cdots$$

```
(define (log-term k)
  (cond
    [(= k 0) 0]
    [else (/ (expt -1 (- k 1)) k)]))
```

9-3 課題

課題 1

- 次の数値の意味は何か

  #i9.313e+020

- 12^{100} を求めたい．次の 3 つを実行して，実行結果を比較せよ

  1. (expt 12 100)
  2. (expt #i12 100)
  3. (expt 12 #i100)

課題２

- 関数 my-exp （授業の例題６）についての問題
  - 次の５つの値を報告しなさい．但し x = 1 とする

$$1+\frac{x}{1!} \quad \text{の値}$$

$$1+\frac{x}{1!}+\frac{x^2}{2!} \quad \text{の値}$$

$$1+\frac{x}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!} \quad \text{の値}$$

$$1+\frac{x}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+\frac{x^4}{4!} \quad \text{の値}$$

$$1+\frac{x}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+\frac{x^4}{4!}+\frac{x^5}{5!} \quad \text{の値}$$

課題３．数列の和の列１

- 関数 series-iter （授業の例題４）についての問題
  - series-iter を使って，次の数列の和のリストを求めなさい
  - 実行結果を報告しなさい
  - また、(series-iter #i10 f4) のように実行した場合と，(series-iter 10 f4) のように実行した場合の違いを報告しなさい

$$\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{2n-1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots = \frac{\pi}{4}$$

課題４．数列の和の列２

- 関数 series-iter （授業の例題４）についての問題
  - series-iter を使って，次の数列の和のリストを求めなさい
  - 実行結果を報告しなさい
  - また、(series-iter #i10 f4) のように実行した場合と，(series-iter 10 f4) のように実行した場合の違いを報告しなさい

$$\sum_{n=1}^{\infty} \frac{1}{n!} = 1 + \frac{1}{2!} + \frac{1}{3!} + \cdots = e - 1$$

課題５．sin 関数のテーラー展開

- 関数 taylor-series（授業の例題５）を使って，sin 関数のテーラー展開の値を求める関数 my-sin を作成し，実行結果を報告しなさい
  - even? （偶数ならば true）を使うこと
  - 正しい値が得られているか確認すること

$$\sin x = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)!} = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

課題６．tan^{-1} 関数のテーラー展開

- 関数 taylor-series（授業の例題５）を使って，tan^{-1} 関数のテーラー展開の値を求める関数 my-atan を作成し，実行結果を報告しなさい
  - even? （偶数ならば true）を使うこと
  - 正しい値が得られているか確認すること

$$\tan^{-1}(x) = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{2i+1} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots$$