

# sp-4. 条件式

(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



4-1 条件式

4-2 パソコン演習

4-3 課題

# 4-1 条件式

- 条件式を使って、より役に立つプログラムを作れるようになる。
  - 比較演算 (<, <=, >, >=, =)
  - 条件式のキーワード : cond, else
  - 奇数, 偶数の判定 (odd?, even?)
  - 論理演算 (and, or, not)

## 演算子

<

<=

>

>=

=

## 意味

より小さい

以下

より大きい

以上

等しい

	数式	意味
$(= x y)$	$x=y$	x is equal to y
$(< x y)$	$x<y$	x is less than y
$(> x y)$	$x>y$	x is greater than y
$(<= x y)$	$x\leq y$	x is less than and equal to y
$(>= x y)$	$x\geq y$	x is greater than and equal to y

# 比較演算を実行してみると



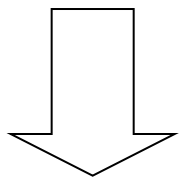
```
> (= 100 50)
false
> (= 100 100)
true
> (< 5 3)
false
> (< 5 5)
false
> (< 5 7)
true
```

比較演算では、条件が成り立てば true, 成り立たなければ false となる

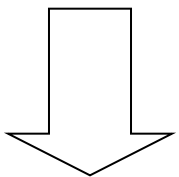
# コンピュータが行っていること



Scheme の式



コンピュータ  
(Scheme 搭載)



式の実行結果

例えば：

(= 100 50)

を入力する  
と・・・



false

が表示される



# 関数の中で比較演算を行ってみると



```
Untitled - DrScheme
File Edit Windows Show Language Scheme Help
Untitled Save Check Syntax Step Execute Break
(define (is-child age)
  (< age 12))

> (is-child 10)
true
> (is-child 11)
true
> (is-child 12)
false
> (is-child 13)
false
```

ここでは、  
(define (is-child age)  
 (< age 12))  
という関数を作ってみた

実行結果として、  
true, false の値が  
得られている

# よくある間違い



- 「 $\leq$ 」, 「 $\geq$ 」を使うことはできない

これは間違い

```
[(<= amount 1000) 0.040]
```

正しくは

```
[(<= amount 1000) 0.040]
```

# 条件式の例



```
(define (interest-rate amount)
```

```
(cond
```

```
  [(<= amount 1000) 0.040]
```

```
  [(<= amount 5000) 0.045]
```

```
  [(> amount 5000) 0.050]))
```

この部分で  
1つの大きな式

amount の値によって、結果が変わってくる  
⇒ 条件式と呼ばれる由来

# 条件式の例



```
(define (interest-rate amount)
  (cond
    [(<= amount 1000) 0.040]
    [(<= amount 5000) 0.045]
    [(> amount 5000) 0.050]))
```

条件の並び

A green arrow originates from the text '条件の並び' and points upwards and to the right, ending at the first condition in the code block: '[(<= amount 1000) 0.040]'. The text '条件の並び' is written in green.

```
(define (interest-rate amount)
  (cond
    ① [(<= amount 1000) 0.040]
    ② [(<= amount 5000) 0.045]
    ③ [(> amount 5000) 0.050]))
```

- cond 文に並んだ複数の「条件」は、上から順に判定される

上の例では、①、②、③の順に判定が行われ、  
①が成り立てば、②、③は判定されない  
②が成り立てば、③は判定されない

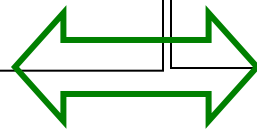
- 「条件」の並んだ順序に意味がある

# else の使用例



```
(define (interest-rate amount)
  (cond
    [(<= amount 1000) 0.040]
    [(<= amount 5000) 0.045]
    [else 0.050]))
```

```
(define (interest-rate amount)
  (cond
    [(<= amount 1000) 0.040]
    [(<= amount 5000) 0.045]
    [(> amount 5000) 0.050]))
```



この2つは、結局は  
同じことを行っている

else とは

「上記の条件がどれも成り立たなければ」という意味

★ else の方が、自分にとって分かりやすければ、else を使う

# even?



```
> (even? 10)
true
> (even? 11)
false
> (even? 12)
true
> (even? 13)
false
```

- **even? の意味** :
  - 偶数ならば true (さもないければ false)

(and A B)            A かつ B

例)    (and (= x y) (< y z))

(or A B)            A または B

例)    (or (= x y) (< x z))

(not A)            A でない

例)    (not (= x y))

真, 偽に関する論理的な演算を行う.



# 条件式での「条件」として書けるもの



「true, false の値をとるもの」なら, 何でも書くことができる

- 比較演算 (<, <=, >, >=, =)

例: (<= amount 1000), (<= amount 5000), (> amount 5000),  
(= (remainder year 400) 0)

- 奇数, 偶数の判定 (odd?, even?)

例: (odd? k), (is-leap? k)

- true, false 値を出力とするような関数

例: (is-child a)

但し, is-child が例えば次のように定義されていること

- and, or, not による組み合わせ

例: (or (= m 1) (= m 2)), (not (= a 0))

など

```
(define (is-child age)
  (< age 12))
```

# Scheme の式の構成要素

- 数値 :

5, -5, 0.5 など

- true, false 値

true, false

- 変数名

- 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

- 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

## 4-2 パソコン演習

- 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

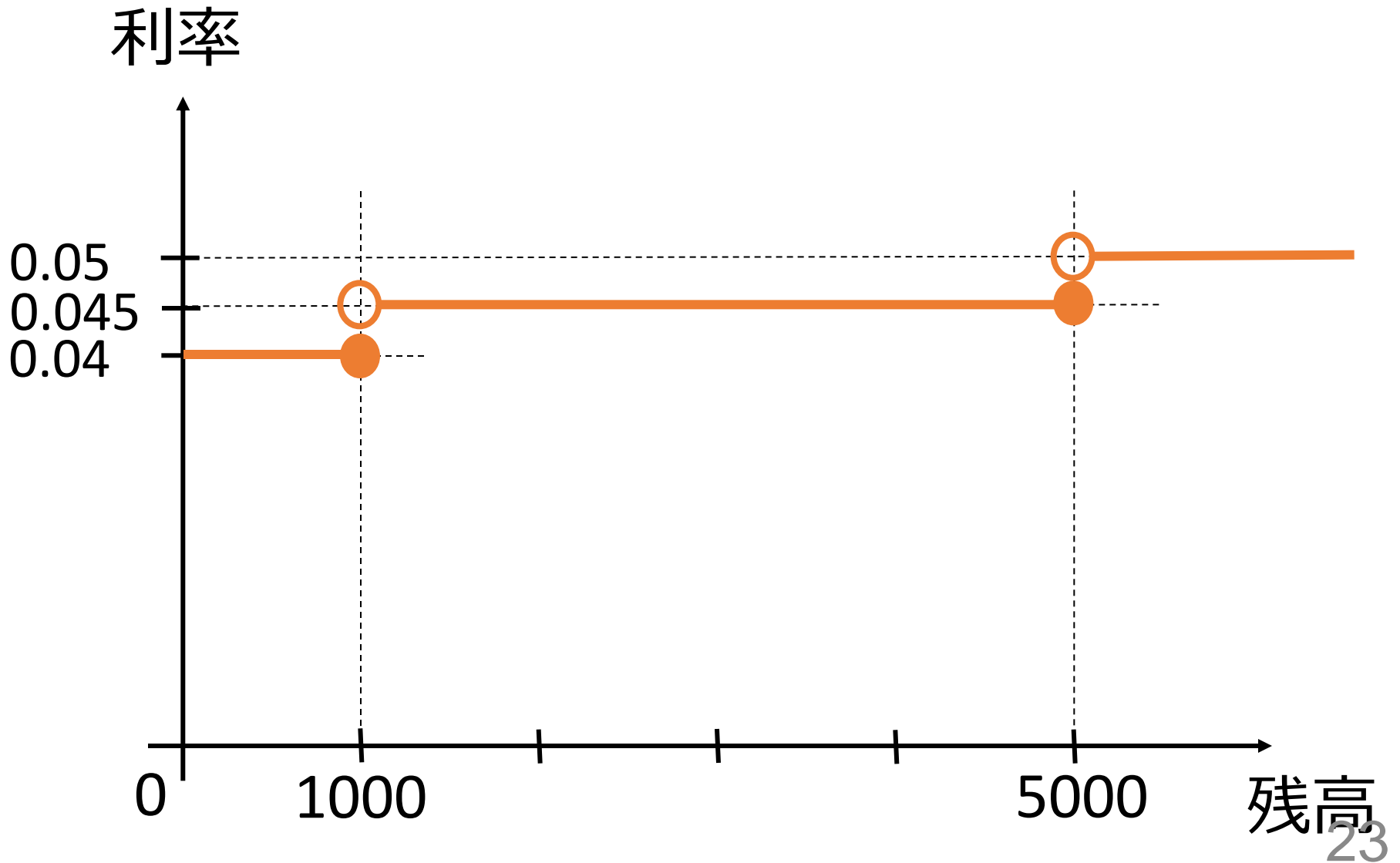
- DrScheme の起動  
プログラム → PLT Scheme → DrScheme
- 今日の演習では「Intermediate Student」  
に設定  
Language  
→ Choose Language  
→ Intermediate Student  
→ Execute ボタン

# 例題 1 . 条件式



- 残高 amount から利率を求める関数 **interest-rate** を作り, 実行する
  - 残高が \$1000 以下ならば : 4%
  - 残高が \$5000 以下ならば : 4.5%
  - 残高が \$5000 より多ければ : 5%
- 残高を条件とする条件式を使う
- cond 句が登場

# 例題 1 . 条件式



# 「例題 1 . 条件式」の手順



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

```
;; interest-rate: number -> number
;; to determine the interest rate
;; for the given amount
(define (interest-rate amount)
  (cond
    [(<= amount 1000) 0.040]
    [(<= amount 5000) 0.045]
    [(> amount 5000) 0.050]))
```

2. その後, 次を「実行用ウィンドウ」で実行しなさい

```
(interest-rate 1000)
```



Untitled - DrScheme

File Edit Windows Show Language Scheme Help

Untitled Save Check Syntax Step Execute Break

```
(define (interest-rate amount)
  (cond
    [(<= amount 1000) 0.040]
    [(<= amount 5000) 0.045]
    [(> amount 5000) 0.050]))
```

>

5:5 Unlocked not running

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



Check Sy

ここでは、  
(interest-rate 1500)  
と書いて、amount の値を  
1500 に設定しての実行

```
(define (interest  
  (cond  
    [(<= amount 1  
    [(<= amount 5  
    [(> amount 5000) 0.050]))])
```

> (interest-rate 1500)

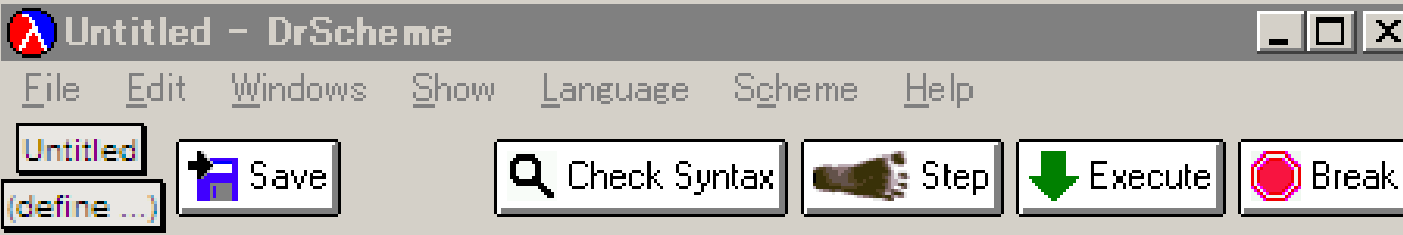
0.045

実行結果である「0.045」が  
表示される

5:3

Unlocked

not running



```
(define (interest
  (cond
    [(<= amount 1
    [(<= amount 5
    [(> amount 50
```

今回は、  
(interest-rate 6000)  
と書いて、amount の値を  
6000 に設定しての実行

```
> (interest-rate 1500)
0.045
> (interest-rate 6000)
0.05
>
```

実行結果である「0.05」が  
表示される

# 入力と出力



# interest-rate 関数



「関数である」ことを  
示すキーワード 関数の名前

```
(define (interest-rate amount)
  (cond
    [(<= amount 1000) 0.040]
    [(<= amount 5000) 0.045]
    [(> amount 5000) 0.050]))
```

} 1つの関数

amount の値から  
利率を求める (出力)

値を1つ受け取る (入力)

# 条件式のプログラム



```
(define (interest-rate amount)
```

```
(cond
```

```
  [( $\leq$  amount 1000) 0.040]
```

```
  [( $\leq$  amount 5000) 0.045]
```

```
  [( $>$  amount 5000) 0.050]))
```

amount  $\leq$  1000 のとき

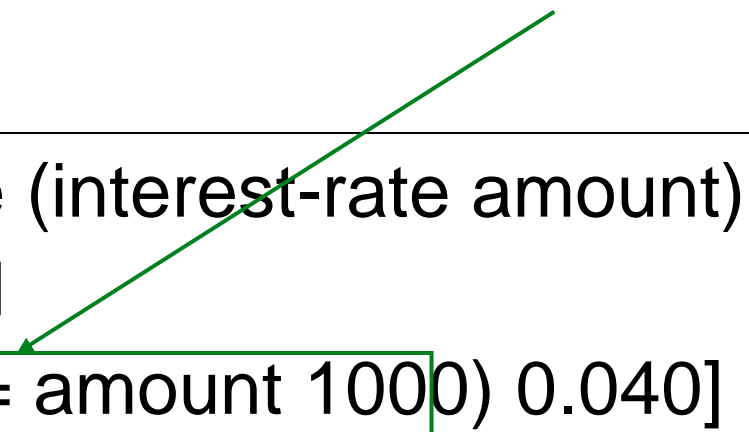
1000 < amount  $\leq$  5000 のとき

5000 < amount のとき

条件式

## 「amount ≤ 1000」の意味

```
(define (interest-rate amount)
  (cond
    [(≤ amount 1000) 0.040]
    [(≤ amount 5000) 0.045]
    [(> amount 5000) 0.050]))
```



```
(define (interest-rate amount)
  (cond
    ① [(<= amount 1000) 0.040]
    ② [(<= amount 5000) 0.045]
    ③ [(> amount 5000) 0.050]))
```

- cond 文に並べた条件式は、上から順に判定される  
上の例では、①、②、③の順に判定が行われ、  
①が成り立てば、②、③は判定されない  
②が成り立てば、③は判定されない
- 条件式の並んだ順序に意味がある



# 字下げを忘れないこと



- 字下げを忘れると
  - プログラムは動くが, 読みづらい

```
(define (interest-rate amount)
  |(cond
    |[(<= amount 1000) 0.040]
    |[(<= amount 5000) 0.045]
    |[(> amount 5000) 0.050]))
```

字下げ

## 例題 2. ステップ実行



- 関数 `interest-rate` (例題 1) について, 実行結果に至る過程を見る
  - (`interest-rate 1500`) から 0.045 に至る過程を見る
  - DrScheme の stepper を使用する

```
(define (interest-rate amount)
  (cond
    [(<= amount 1000) 0.040]
    [(<= amount 5000) 0.045]
    [(> amount 5000) 0.050]))
```

```
(interest-rate 1500)
= (cond
  [(<= 1500 1000) 0.040]
  [(<= 1500 5000) 0.045]
  [(> 1500 5000) 0.050])
= (cond
  [false 0.040]
  [(<= 1500 5000) 0.045]
  [(> 1500 5000) 0.050])
= (cond
  [(<= 1500 5000) 0.045]
  [(> 1500 5000) 0.050])
= (cond
  [true 0.045]
  [(> 1500 5000) 0.050])
= 0.045
```

## 「例題 2. ステップ実行」の手順

1. 次を「定義用ウィンドウ」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

```
;; interest-rate: number -> number  
;; to determine the interest rate  
;; for the given amount  
(define (interest-rate amount)  
  (cond  
    [(<= amount 1000) 0.040]  
    [(<= amount 5000) 0.045]  
    [(> amount 5000) 0.050]))  
(interest-rate 1500)
```

例題 1 と同じ

例題 1 に  
1 行書き加える

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 3 に進んでください

# (interest-rate 1500) から 0.045 が得られる過程



(interest-rate 1500) 最初の式

```
= (cond  
  [(<= 1500 1000) 0.040]  
  [(<= 1500 5000) 0.045]  
  [(> 1500 5000) 0.050])
```

```
(cond  
  [(<= amount 1000) 0.040]  
  [(<= amount 5000) 0.045]  
  [(> amount 5000) 0.050])
```

(こ amount = 1500 が代入される

```
= (cond  
  [false 0.040]  
  [(<= 1500 5000) 0.045]  
  [(> 1500 5000) 0.050])
```

(<= 1500 1000) → false

```
= (cond  
  [(<= 1500 5000) 0.045]  
  [(> 1500 5000) 0.050])
```

[false 0.040] → 消える

```
= (cond  
  [true 0.045]  
  [(> 1500 5000) 0.050])
```

(<= 1500 5000) → true

コンピュータ内部での計算

```
= 0.045 実行結果
```

# (interest-rate 1500) から 0.045 が得られる過程

(interest-rate 1500)

```
= (cond  
  [(<= 1500 1000) 0.040]  
  [(<= 1500 5000) 0.045]  
  [(> 1500 5000) 0.050])
```

= (cond

```
[ これは,  
[  
[ (define (interest-rate amount)  
[ (cond  
= (c  
[ [(<= amount 1000) 0.040]  
[ [(<= amount 5000) 0.045]  
= (c  
[ [(> amount 5000) 0.050]])  
[ の amount を 1500 で置き換えたもの  
[ (> 1500 5000) 0.050])
```

= 0.045

## 例題 3 . 月の日数



- 月 month から, 日数を求めるプログラム `easy-get-num-of-days` を書く. ここでは、うるう年のことは考えない
  - 比較演算と論理演算を組み合わせる

# 「例題 3 . 月の日数」の手順

1. 次を「定義用ウィンドウ」で, 実行しなさい

```
(define (easy-get-num-of-days month)
  (cond
    [(= month 2) 28]
    [(or (= month 4)
         (= month 6)
         (= month 9)
         (= month 11)) 30]
    [else 31]))
```

2. その後, 次を「実行用ウィンドウ」で実行しなさい

```
(easy-get-num-of-days 1)
(easy-get-num-of-days 2)
(easy-get-num-of-days 3)
(easy-get-num-of-days 4)
```

☆ 次は, 例題 4 に進んでください



```
(define (easy-get-num-of-days month)
  (cond
    [(= month 2) 28]
    [(or (= month 4)
         (= month 6)
         (= month 9)
         (= month 11)) 30]
    [else 31]))
```

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



ここでは、  
(easy-get-num-of-days 1)  
と書いて、month の値を 1  
に設定しての実行

```
(define (easy-get-num-of-days month)
  (cond
    [(= month 2) 28]
    [(or (= month 1)
         (= month 3)
         (= month 4)
         (= month 11)) 30]
    [else 31]))
```

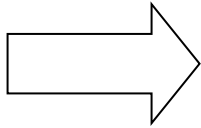
```
> (easy-get-num-of-days 1)
31
> (easy-get-num-of-days 2)
28
> (easy-get-num-of-days 3)
31
> (easy-get-num-of-days 4)
30
>
```

実行結果である「31」が  
表示される

# 入力と出力



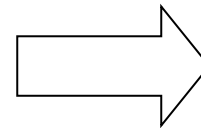
1



入力

easy-get-num-of-days

31



出力

入力は  
1つの数値

出力は  
1つの数値

# easy-get-num-of-days 関数



「関数である」ことを  
示すキーワード      関数の名前

```
(define (easy-get-num-of-days month)
  (cond
    [(= month 2) 28]
    [(or (= month 4)
          (= month 6)
          (= month 9)
          (= month 11)) 30]
    [else 31]))
```

値を1つ受け取る (入力)

## 例題 4 . ステップ実行

- 関数 `easy-get-num-of-days` (例題 3) について, 実行結果に至る過程を見る
  - (`easy-get-num-of-days 1`) から 31 に至る過程を見る
  - DrScheme の stepper を使用する

```
(define (easy-get-num-of-days month)
  (cond
    [(= month 2) 28]
    [(or (= month 4)
         (= month 6)
         (= month 9)
         (= month 11)) 30]
    [else 31]))
```

## 「例題 4 . ステップ実行」の手順

1. 次を「定義用ウィンドウ」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 (easy-get-num-of-days month)
  (cond
    [(= month 2) 28]
    [(or (= month 4)
         (= month 6)
         (= month 9)
         (= month 11)) 30]
    [else 31]))
```

```
(easy-get-num-of-days 1)
```

例題 3 と同じ

例題 3 に  
1 行書き加える

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 5 に進んでください

# (easy-get-num-of-days 1) から 31 が得られる過程 (1/3)

## (easy-get-num-of-days 1) 最初の式

```
= (cond  
  [(= 1 2) 28]  
  [(or (= 1 4)  
        (= 1 6)  
        (= 1 9)  
        (= 1 11)) 30]  
  [else 31]))
```

```
(cond  
  [(= month 2) 28]  
  [(or (= month 4)  
        (= month 6)  
        (= month 9)  
        (= month 11)) 30]  
  [else 31]))
```

に month = 1 が代入される

```
= (cond  
  [false 28]  
  [(or (= 1 4)  
        (= 1 6)  
        (= 1 9)  
        (= 1 11)) 30]  
  [else 31]))
```

(= 1 2) → false

```
= (cond  
  [(or (= 1 4)  
        (= 1 6)  
        (= 1 9)  
        (= 1 11)) 30]  
  [else 31]))
```

[false 0.040] → 消える

# (easy-get-num-of-days 1) から 31 が得られる過程 (1/3)

(easy-get-num-of-days 1)

```
= (cond
  [(= 1 2) 28]
  [(or (= 1 4)
        (= 1 6)
        (= 1 9)
        (= 1 11)) 30]
  [else 31]))
```

= (cond



これは,

(define (easy-get-num-of-days month)

```
(cond
  [(= month 2) 28]
  [(or (= month 4)
        (= month 6)
        (= month 9)
        (= month 11)) 30]
  [else 31]))
```

の month を 1 で置き換えたもの

# (easy-get-num-of-days 1) から 31 が得られる過程 (2/3)



Database Lab.

```
= (cond
  [(or false
        (= 1 6)
        (= 1 9)
        (= 1 11)) 30]
  [else 31]))
```

(= 1 4) → false

```
= (cond
  [(or (= 1 9)
        (= 1 11)) 30]
  [else 31]))
```

false → 消える

```
= (cond
  [(or false
        (= 1 11)) 30]
  [else 31]))
```

(= 1 9) → false

```
= (cond
  [(or (= 1 11)) 30]
  [else 31]))
```

false → 消える

```
= (cond
  [(or false) 30]
  [else 31]))
```

(= 1 11) → false



# (easy-get-num-of-days 1) から 31 が得られる過程 (3)



```
= (cond  
  [false 30]          (or false) → false  
  [else 31]))
```

コンピュータ内部での計算

= 31 出力結果

## 例題 5. うるう年の判定



- 西暦年 `year` から, うるう年であるかを求める関数 `is-leap?` を作り, 実行する
  - うるう年の判定のために, 比較演算と論理演算を組み合わせる
  - 西暦年が 4, 100, 400 の倍数であるかを調べるために `remainder` を使う

例) 2001 → うるう年でない

2004 → うるう年である

# グレゴリオ暦でのうるう年



- うるう年とは： 2月が29日までである年
- うるう年は400年に97回で， 1年の平均日数は365.2422日
- うるう年の判定法
  - 年数が4の倍数の年 → うるう年
  - 但し， 100の倍数の年で400の倍数でない年  
→ うるう年ではない  
(4の倍数なのだが例外とする)

(例) 2008年： うるう年 (4の倍数)  
2004年： うるう年 (4の倍数)  
2000年： うるう年 (4の倍数)  
1900年： うるう年ではない

(100の倍数だが400の倍数でない)  
1800年： うるう年ではない

(100の倍数だが400の倍数でない) 51

# うるう年の判定式



$(\text{or } (= (\text{remainder year } 400) 0) (\text{and } (\text{not } (= (\text{remainder year } 100) 0)) (= (\text{remainder year } 4) 0)))$

400の倍数である

100の倍数でない

4の倍数である

かつ

または

# 「例題 5. うるう年の判定」の手順



1. 次を「定義用ウィンドウ」で、実行しなさい

```
(define (is-leap? year)
  (cond
    [(or (= (remainder year 400) 0)
          (and (not (= (remainder year 100) 0))
                (= (remainder year 4) 0))) true]
    [else false]))
```

2. その後、次を「実行用ウィンドウ」で実行しなさい

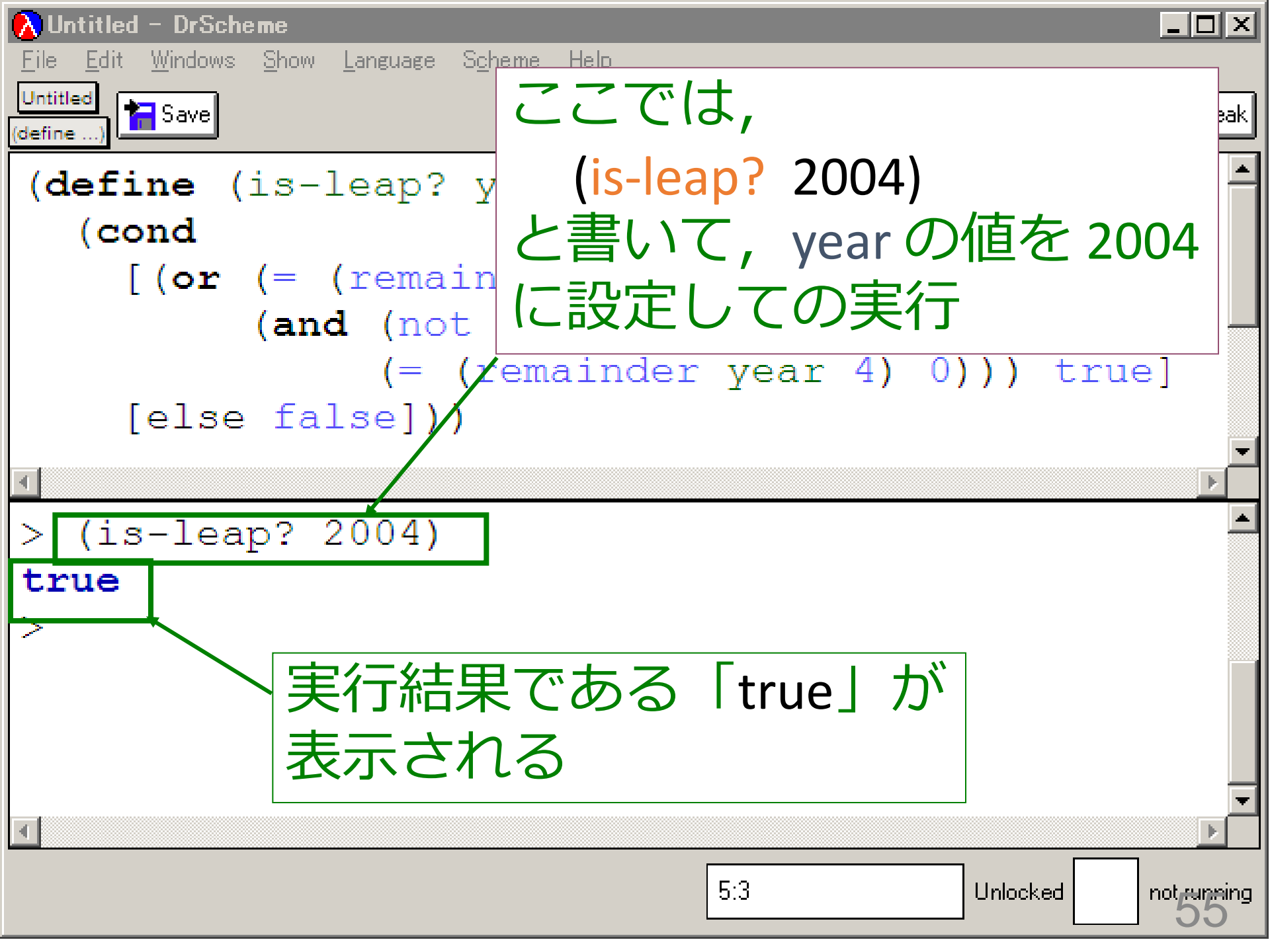
```
(is-leap? 2004)
(is-leap? 2005)
```

☆ 次は、例題 6 に進んでください

Untitled - DrScheme  
File Edit Windows Show Language Scheme Help  
Untitled Save Check Syntax Step Execute Break

```
(define (is-leap? year)
  (cond
    [(or (= (remainder year 400) 0)
         (and (not (= (remainder year 100) 0))
              (= (remainder year 4) 0))) true]
    [else false]))
```

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



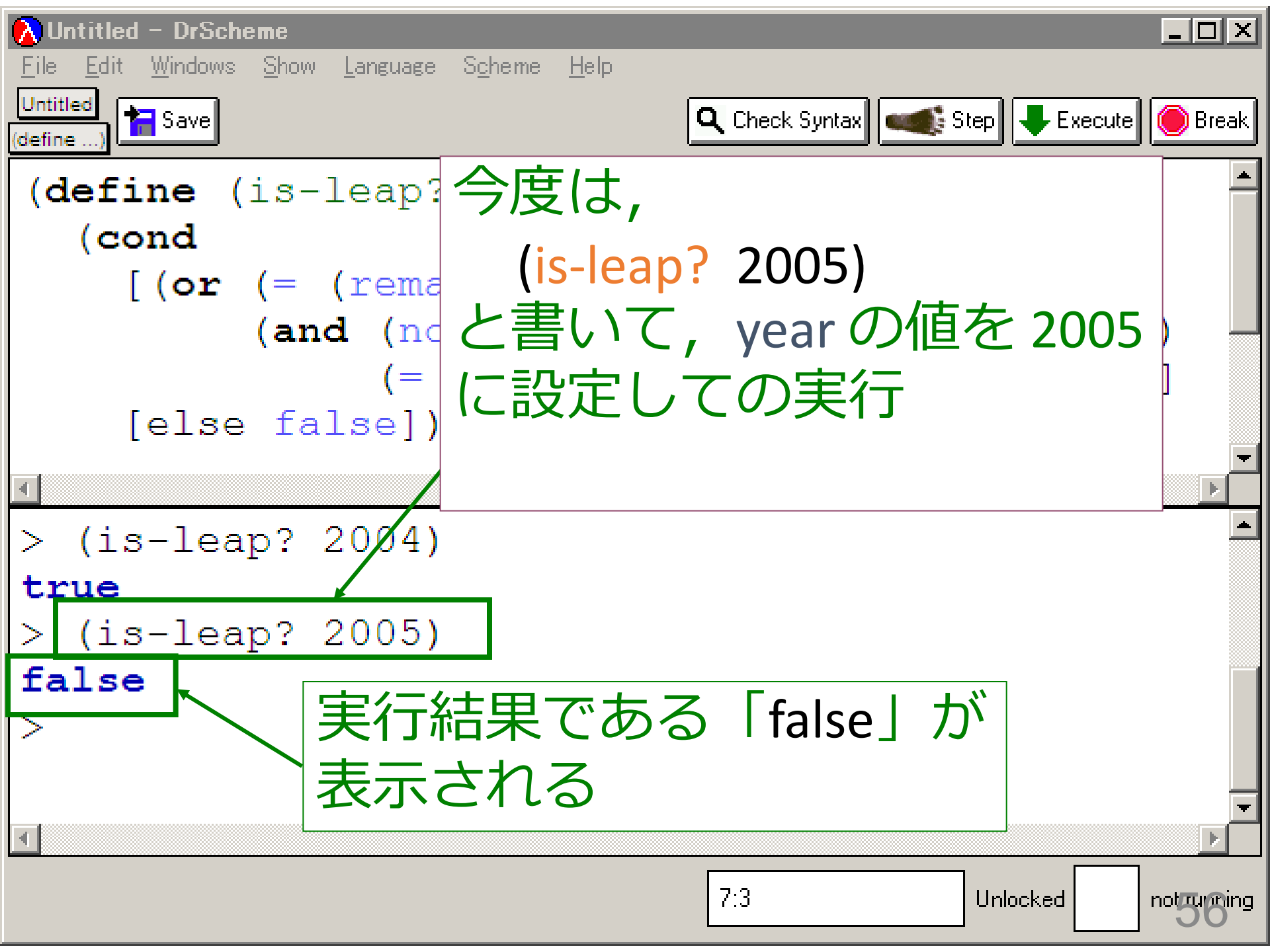
ここでは,  
(is-leap? 2004)  
と書いて, year の値を 2004  
に設定しての実行

```
(define (is-leap? year)
  (cond
    [(or (= (remainder year 4) 0)
          (and (not (= (remainder year 4) 0))
                (= (remainder year 4) 0))) true]
    [else false]))
```

```
> (is-leap? 2004)
```

```
true
```

実行結果である「true」が  
表示される



```
(define (is-leap? year)
  (cond
    [(or (= (remainder year 4) 0)
         (and (not (= (remainder year 100) 0))
              (= (remainder year 400) 0)))]
    [else false]))
```

今回は,  
(is-leap? 2005)  
と書いて, year の値を 2005  
に設定しての実行

```
> (is-leap? 2004)
true
> (is-leap? 2005)
false
>
```

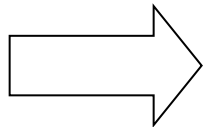
実行結果である「false」が表示される



# 入力と出力



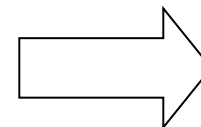
2004



入力



true



出力

出力は  
true あるいは false 値

# is-leap? 関数



「関数である」ことを  
示すキーワード      関数の名前

```
(define (is-leap? year)
  (cond
    [(or (= (remainder year 400) 0)
          (and (not (= (remainder year 100) 0))
                (= (remainder year 4) 0))) true]
    [else false]))
```

year の値から true  
あるいは false を求める (出力)

値を 1 つ受け取る (入力)

# うるう年の判定



1つの条件式

```
(define (is-leap? year)
```

```
(cond
```

```
[ (or (= (remainder year 400) 0)  
      (and (not (= (remainder year 100) 0))  
            (= (remainder year 4) 0))) true]
```

```
[else false]))
```

## 例題 6 . ステップ実行



- 関数 `is-leap?` (例題 5) について, 実行結果に至る過程を見る
  - (`is-leap?` 2004) から `true` に至る過程を見る
  - DrScheme の `stepper` を使用する

```
(define (is-leap? year)
  (cond
    [ (or (= (remainder year 400) 0)
          (and (not (= (remainder year 100) 0))
                (= (remainder year 4) 0))) true]
    [else false]))
```

## 「例題 6 . ステップ実行」の手順

1. 次を「定義用ウィンドウ」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 (is-leap? year)
  (cond
    [(or (= (remainder year 400) 0)
          (and (not (= (remainder year 100) 0))
                (= (remainder year 4) 0))) true]
    [else false]))
```

```
(is-leap? 2004)
```

例題 5  
と同じ

例題 5 に  
1 行書き加える

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



# (is-leap? 2004) から true が得られる過程 (1/3)

## 最初の式

(is-leap? 2004)

```
= (cond
  [(or (= (remainder 2004 400) 0)
        (and (not (= (remainder 2004 100) 0))
              (= (remainder 2004 4) 0)))] true]
 [else false]))
```

```
(cond
 [(or (= (remainder year 400) 0)
       (and (not (= (remainder year 100) 0))
             (= (remainder year 4) 0)))] true]
 [else false])
```

ここで year = 2004 が代入される

```
= (cond
  [(or (= 4 0)
        (and (not (= (remainder 2004 100) 0))
              (= (remainder 2004 4) 0)))] true]
 [else false]))
```

(remainder 2004 400) → 4

```
= (cond
  [(or false
        (and (not (= (remainder 2004 100) 0))
              (= (remainder 2004 4) 0)))] true]
 [else false]))
```

(= 4 0) → false

# (is-leap? 2004) から true が得られる過程 (1/3)



(is-leap? 2004)

```
= (cond
  [(or (= (remainder 2004 400) 0)
        (and (not (= (remainder 2004 100) 0))
              (= (remainder 2004 4) 0))) true]
  [else false]))
```

```
= (cond
  [(or (= 4 0)
```

これは,

```
(define (is-leap? year)
```

```
(cond
```

```
  [(or (= (remainder year 400) 0)
```

```
        (and (not (= (remainder year 100) 0))
```

```
              (= (remainder year 4) 0))) true]
```

```
  [else false]))
```

の year を 2004 で置き換えたもの



# (is-leap? 2004) から true が得られる過程 (2/3)

= (cond  
 [(or (and (not (= remainder 2004 100) 0))  
 (= (remainder 2004 4) 0))) true]  
 [else false]))

(or false 式) → (or 式)

= (cond  
 [(or (and (not (= 4 0))  
 (= (remainder 2004 4) 0))) true]  
 [else false]))

(remainder 2004 100) → 4

= (cond  
 [(or (and (not false)  
 (= (remainder 2004 4) 0))) true]  
 [else false]))

(= 4 0) → false

= (cond  
 [(or (and true  
 (= (remainder 2004 4) 0))) true]  
 [else false]))

(not false) → true



# (is-leap? 2004) から true が得られる過程 (3/3)

```
= (cond  
  [(or (and (= (remainder 2004 4) 0))) true]  
  [else false]))
```

(and true 式) → (and 式)

```
= (cond  
  [(or (and (= 0 0))) true]  
  [else false]))
```

(remainder 2004 4) → 0

```
= (cond  
  [(or (and true)) true]  
  [else false]))
```

(= 0 0) → true

```
= (cond  
  [(or true) true]  
  [else false]))
```

(and true) → true

```
= (cond  
  [true true]  
  [else false]))
```

(or true) → true

コンピュータ内部での計算

= true 実行結果

## 例題 7. 曜日を求める



- ツエラーの公式 **zellar** を作り, 実行する
  - ツエラーの公式とは : 西暦の年, 月, 日から曜日を求める公式
  - 年 : 西暦の年
  - 月 : 3 から 14
  - 計算された曜日は「数字」. 次の意味になる.
    - 0 : 日曜日
    - 1 : 月曜日
    - 2 : 火曜日
    - 3 : 水曜日
    - 4 : 木曜日
    - 5 : 金曜日
    - 6 : 土曜日

- ツエラーの公式では、「1年の起点を3月とし、月は3月から14月までである」と考えている
  - 1月, 2月は, 前年の13, 14月と考えるということ
  - うるう年があるので, 1年の起点を3月とする方が計算が簡単

# ツエラーの公式



$$[(y+[y/4]+[y/400]-[y/100]+[(13m+8)/5]+d) \bmod 7]$$

y: 年

m: 3月を起点とする月（3から14まで）

1月, 2月は, 前年の13, 14月と考える

d: 日

- この値が0なら日曜, 1なら月曜 . . .
- 「[]」 とあるのは, 小数点以下切り捨て
- 「mod」 とあるのは剰余. 例えば  $2003 \bmod 4$  は 3

# remainder

A screenshot of the DrScheme IDE window titled "Untitled - DrScheme". The menu bar includes File, Edit, Windows, Show, Language, Scheme, and Help. Below the menu bar are buttons for "Check Syntax", "Step", "Execute", and "Break". The main text area contains the following code:

```
> (remainder 5 2)  
1  
> (remainder 200 3)  
2
```

The status bar at the bottom shows the time "9:3", the state "Unlocked", and "not running".

- remainder の意味：
  - 割り算の余りを求める
  - (remainder x y) は, ツエラーの公式の  $x \bmod y$  に対応

# quotient



The screenshot shows the DrScheme IDE window titled "Untitled - DrScheme". The menu bar includes "File", "Edit", "Windows", "Show", "Language", "Scheme", and "Help". Below the menu bar are buttons for "Check Syntax", "Step", "Execute", and "Break". The main text area contains the following Scheme code and its output:

```
> (quotient 5 2)
2
> (quotient 200 3)
66
```

At the bottom of the window, there is a status bar with a text field containing "7:3", the word "Unlocked", a checkbox, and the text "not running".

- **quotient の意味** :
  - 整数の割り算の商を求める
  - (quotient x y) は、 ツエラーの公式の  $[x / y]$  に対応

# ツエラーの公式のプログラム



```
[(y+[y/4]+[y/400]-[y/100]+[(13m+8)/5]+d) mod 7]
```

```
(define (zellar_f y m d)
  (remainder
    (+ y
      (quotient y 4)
      (quotient y 400)
      (- (quotient y 100))
      (quotient (+ (* 13 m) 8) 5)
      d)
    7))
```

;; zellar : number number number -> number

;; to determine youbi (the day of the week) from year, month, and day.

;; The result is 0 ... 6 meaning 0 is Sunday, 1 Monday, and so on

```
(define (zellar y m d)
  (cond
    [(or (= m 1) (= m 2)) (zellar_f (- y 1) (+ m 12) d)]
    [else (zellar_f y m d)]))
```

# 4-3 課題



# 課題 1



- 次のような関数を作りなさい
  - cond を使うこと

5 0 グラム以下なら → 1 2 0

7 5 グラム以下なら → 1 4 0

1 0 0 グラム以下なら → 1 6 0

1 5 0 グラム以下なら → 2 0 0

- ここにあるのは「間違い」の例です。同じ間違いをしないこと

## 1. 関数名の付け方の間違い

```
(define (decice price w)  
  ... 以下省略
```

⇒ 「decice price」ではなく、  
「decide\_price」のように1単語で書くこと

## 2. $\leq$ , $\geq$ をプログラム中に使うことはできない

⇒ 代わりに,  $\leq$ ,  $\geq$  を使うこと

- ある年  $y$  のある月  $m$  の日数を返す関数 `get-num-of-days` を作成し, 実行結果を報告しなさい
- `easy-get-num-of-days` (授業の例題 3) と, `is-leap?` (授業の例題 5) を利用しなさい
- うるう年の 2 月についても, 正しい日数を求めること

# 課題 3



- 次の値を求める関数 **foo2** を書きなさい
  - 「 $X \bmod 7$ 」は、 $X$  を 7 で割った時の余り（剰余）。  
例えば  $2003 \bmod 4$  は 3 である。
  - `define` と `remainder` を使いなさい。

$$[(20x + 8) / 7] \bmod 10$$