関数型言語（OCaml 演習）(11)

例外と例外処理

浅井 健一

お茶の水女子大学

例外的な状況

シナリオ１　円からドルへの変換を行うウェブサイトを作った。そうしたらユーザが数字ではなく文字列を入力してしまった。

シナリオ２　八百屋でいろいろな野菜を買いたい。合計金額を出そうとしたら、八百屋においていない野菜があった。

シナリオ３　割り算をしようとしたら、割る数が 0 だった。

例外的な状況は日常的に起こりうる。

問題設定

学生の成績のリストがある。その中から、指定した名前の学生について、成績通知文を作る。

```
(* 学生データの例 *)
let g1 = {namae="asai"; tensuu=70; seiseki="B"}
let g2 = {namae="tanaka"; tensuu=90; seiseki="A"}
let g3 = {namae="yamada"; tensuu=60; seiseki="C"}

(* 学生リストの例 *)
let gakusei_list = [g1; g2; g3]

(* テスト *)
let test1 = tsuuchi "tanaka" gakusei_list =
                                "tanaka さんの成績は A です"
```

option 型

通常の値に「値がない」ことを表す値を加えた型。

```
type 'a option = None
               | Some of 'a

# Some (3) ;;                (* 整数の値があって 3 *)
- : int option = Some 3
# None ;;                    (*（整数の）値がない *)
- : 'a option = None
```

通常は整数値を取るが、たまに値がないこともあるという状況は、int option 型で表現できる。

学生リストから成績を取ってくる

```
(* 目的：指定された学生の成績を返す *)
(* get_seiseki : string -> gakusei_t list ->
                                    string *)
let rec get_seiseki name lst = match lst with
    [] -> ???
  | {namae=n; tensuu=t; seiseki=s} :: rest ->
      if n = name then s
                  else get_seiseki name rest
```

学生リストから成績を取ってくる

```
(* 目的：指定された学生の成績を返す *)
(* get_seiseki : string -> gakusei_t list ->
                                    string *)
let rec get_seiseki name lst = match lst with
    [] -> ???
  | {namae=n; tensuu=t; seiseki=s} :: rest ->
      if n = name then s
                  else get_seiseki name rest
```

この関数は常に string の結果を返せるわけではない。
返せないこともある。

学生リストから成績を取ってくる

```
(* 目的：指定された学生の成績を返す *)
(* get_seiseki : string -> gakusei_t list ->
                                    string option *)
let rec get_seiseki name lst = match lst with
    [] -> None
  | {namae=n; tensuu=t; seiseki=s} :: rest ->
      if n = name then Some (s)
                  else get_seiseki name rest
```

この関数は常に string の結果を返せるわけではない。
返せないこともある。→ string option 型を使う。

指定した学生の成績通知文を作る

```
(* 目的：指定された学生の成績通知文を返す *)
(* tsuuchi : string -> gakusei_t list ->
                                        string *)
let tsuuchi name lst =
  let s = get_seiseki name lst in


        name ^ " さんは " ^    s    ^ " です"
```

s は string 型ではなく string option 型。
したがって最後の行で型エラーが起きる。

指定した学生の成績通知文を作る

```
(* 目的：指定された学生の成績通知文を返す *)
(* tsuuchi : string -> gakusei_t list ->
                                        string *)
let tsuuchi name lst =
  match   get_seiseki name lst with
      None -> ???

    | Some (seiseki) ->
        name ^ " さんは " ^ seiseki ^ " です"
```

string option 型の値を使うには match 文でばらす。

指定した学生の成績通知文を作る

```
(* 目的：指定された学生の成績通知文を返す *)
(* tsuuchi : string -> gakusei_t list ->
                                        string *)
let tsuuchi name lst =
  match   get_seiseki name lst with
      None ->
        name ^ " さんのデータがありません"
    | Some (seiseki) ->
        name ^ " さんは " ^ seiseki ^ " です"
```

string option 型の値を使うには match 文でばらす。

option 型を使った例外処理

例外の定義　通常の処理には Some を使い、
例外的な場合には None を使う。

例外の発生　Some ではなく None を返す。

例外の処理　match 文で None が返されたときの処理を書く。

特徴：

- 専用の構文を使うことなく、例外処理ができる。
- プログラム全体に Some と None があふれかえり、プログラムの可読性が下がる傾向にある。

例外の発生

raise 例外

例外を起こす。

```
# raise Not_found ;;
Exception: Not_found.
# 1 + 2 * (raise Not_found) ;;
Exception: Not_found.
```

Not_found は、あらかじめ OCaml に定義されている例外。

例外の定義

exception 例外 of 型

例外を定義する。(引数不要なら of 以下を省略する。)

```
# exception DataNashi ;;                (* 引数なし *)
exception DataNashi
# raise DataNashi ;;
Exception: DataNashi.
# exception Urikire of string ;; (* 引数あり *)
exception Urikire of string
# raise (Urikire ("tomato")) ;;
Exception: Urikire "tomato".
```

例外は、大文字で始まらなくてはならない。

例外処理

```
try
   式0
with 例外のパターン1 -> 式1
    | 例外のパターン2 -> 式2 | ...
```

1. 式$_0$ を実行する。
2. 実行中に例外が起きなければ、その結果が `try` 文全体の結果となる。（`with` 以下は無視される。）
3. 例外が起きたら `with` 以下の例外のパターンから合うものを選んで、矢印の右の式を実行する。
4. 合うものがなければ、外側の `try` 文を探す。
5. 最後まで合うものがなかったら、プログラム全体がエラー終了する。

例外処理

```
try                                  式_i は皆、同じ型でなくてはならない
   式_0
with 例外のパターン_1 -> 式_1
   | 例外のパターン_2 -> 式_2 | ...
```

1. 式$_0$ を実行する。
2. 実行中に例外が起きなければ、その結果が `try` 文全体の結果となる。（`with` 以下は無視される。）
3. 例外が起きたら `with` 以下の例外のパターンから合うものを選んで、矢印の右の式を実行する。
4. 合うものがなければ、外側の `try` 文を探す。
5. 最後まで合うものがなかったら、プログラム全体がエラー終了する。

```
let f n =
  if n = 0 then raise DataNashi
  else if n = 1 then raise (Urikire "tomato")
  else n + 1

let g n = try
            10 + f n
          with DataNashi -> 0
```

```
let f n =
  if n = 0 then raise DataNashi
  else if n = 1 then raise (Urikire "tomato")
  else n + 1

let g n = try
            10 + f n
          with DataNashi -> 0
(* g 0 → 0, g 1 → 例外 Urikire ("tomato"),
   g 2 → 13 *)
```

```
let f n =
  if n = 0 then raise DataNashi
  else if n = 1 then raise (Urikire "tomato")
  else n + 1

let g n = try
            10 + f n
          with DataNashi -> 0
(* g 0 → 0, g 1 → 例外 Urikire ("tomato"),
   g 2 → 13 *)

let h n = try
            100 + g n
          with Urikire (s) -> String.length s
```

```
let f n =
  if n = 0 then raise DataNashi
  else if n = 1 then raise (Urikire "tomato")
  else n + 1

let g n = try
            10 + f n
          with DataNashi -> 0
(* g 0 → 0, g 1 → 例外 Urikire ("tomato"),
   g 2 → 13 *)

let h n = try
            100 + g n
          with Urikire (s) -> String.length s
(* h 0 → 100, h 1 → 6, h 2 → 113 *)
```

学生リストから成績を取ってくる

```
(* 目的：指定された学生の成績を返す *)
(* 見つからなかったら例外 DataNashi を起こす *)
(* get_seiseki : string -> gakusei_t list ->
                                    string *)
let rec get_seiseki name lst = match lst with
    [] -> raise DataNashi
  | {namae=n; tensuu=t; seiseki=s} :: rest ->
      if n = name then s
                  else get_seiseki name rest
```

get_seiseki は string option ではなく string そのものを返す。「値がない」ケースは、例外で区別。

指定した学生の成績通知文を作る

```
(* 目的：指定された学生の成績通知文を返す *)
(* tsuuchi : string -> gakusei_t list ->
                                        string *)
let tsuuchi name lst =

    let seiseki = get_seiseki name lst in
    name ^ " さんは " ^ seiseki ^ " です"
```

get_seiseki の返すものは string なので、
その結果を直接使うことができる。

指定した学生の成績通知文を作る

```
(* 目的：指定された学生の成績通知文を返す *)
(* tsuuchi : string -> gakusei_t list ->
                                        string *)
let tsuuchi name lst =
  try
    let seiseki = get_seiseki name lst in
    name ^ " さんは " ^ seiseki ^ " です"
  with DataNashi ->
    name ^ " さんのデータがありません"
```

get_seiseki が例外を起こしたときの対処は、
try ... with ... で行う。

例外構文を使った例外処理

例外の定義 `exception` 文を使って定義する。

例外の発生 `raise` を使う。実行が中断され、直近の `try` 文に制御を移す。

例外の処理 `try` 文の `with` 以下に、処理する例外と処理内容を書く。

特徴：

- 例外を起こすとき、処理するときのみ記述すれば良く、他の部分では例外を意識しなくて良い。
- どのような例外が起こりうるかは、自動ではチェックされない。

  → 起きうる例外をヘッダに書き、注意を促す。

まとめ

- 値がないかも知れない型を表す 'a option 型。
- Some, None

exception 例外 (of 型)	(引数付き) 例外の宣言。例外は大文字で始める。
raise 例外	例外を起こす。直近の try 文までジャンプする。
try 式 with 例外 -> 式	例外を処理する。

これらの専用構文を使うと、プログラムの構造を乱すことなく例外的な場合を処理できる。

関数が値を返さずに例外を起こす可能性のあるときはそれをヘッダにコメントとして書き残す。