OCaml (あるいは F#) のプログラミングに関する課題 の解答 (例)

- 課題 1

  「入力は正の整数 1 つ。出力はその (正の) 約数の個数」という関数を書きなさい。

  解答例:

  `ndiv_sub n i` は,「n の約数で，i 以上のものの個数」を返す.

```
let rec ndiv_sub n i =
  if i > n then 0
  else if n mod i = 0 then (ndiv_sub n (i+1)) + 1
  else ndiv_sub n (i+1)
```

  `ndiv_sub n` は,「n の (正の) 約数の個数」を返す .

```
let ndiv n = ndiv_sub n 1
```

  実行例:

```
# ndiv 4 ;;
- : int = 3
# ndiv 5 ;;
- : int = 2
# ndiv 6 ;;
- : int = 4
```

- 課題 2

  以下の関数を書きなさい。入力は正の整数 2 つ：1 引数が 1 あるいは第 2 引数が 1 なら 第 1 引数を返して終了、そうでなければ「第 1 引数が偶数なら 2 で割り、3 以上の奇数なら 3 倍して 1 を足す」という操作を行い再帰的に動作する。ただし、第 2 引数は 1 を引く。つまり、f(4,5) を呼ぶと、次に f(2,4) を呼ぶ。また、f(7,5) を呼ぶと、次に、f(22,4) を呼ぶ。

  第 2 引数が 1 つずつ減っていくので、いつかは終了するので、そのときの第 1 引数の値が返るはずである。

  解答例: 関数 f を素朴に実装すると以下のようになる.

```
let rec f n c =
   if n = 1 then n
   else if c = 1 then n
   else if n mod 2 = 0 then f (n / 2) (c - 1)
   else f (n * 3 + 1) (c - 1)
```

  実行例:

```
# f 10 1 ;;
- : int = 10
# f 10 10 ;;
f 10 1 ;;
- : int = 10
# f 10 10 ;;
- : int = 1
# f 13 3 ;;
- : int = 20
# f 13 5 ;;
- : int = 5
# f 13 10 ;;
- : int = 1
```

- 課題 3 (発展課題)

  講義スライドにかいてあるインタプリタを改良して、整数だけでなく、浮動小数点数 (float 型の値) の四則演算ができるようにしなさい。

解答例: 浮動小数点数だけを扱えるインタプリタは以下の通り。

```
type expr =
  | CstF of float
  | Prim of string * expr * expr

let rec eval e =
 match e with
  | CstF f   → f
  | Prim("+", e1, e2) → (eval e1) +. (eval e2)
  | Prim("−", e1, e2) → (eval e1) −. (eval e2)
  | Prim("*", e1, e2) → (eval e1) *. (eval e2)
  | Prim("/", e1, e2) → (eval e1) /. (eval e2)
  | _ → failwith "unknown expression"
```

実行例:

```
# eval (Prim("*",CstF(3.1),CstF(2.5))) ;;
- : float = 7.75
# eval (Prim("*",CstF(3.1),Prim("−",CstF(2.5),CstF(−5.6)))) ;;
- : float = 25.11
```