関数型言語（OCaml 演習）(1)

基本データ型、変数、関数

浅井 健一

お茶の水女子大学

整数（int 型）

```
# 3 ;;                          <- 入力
- : int = 3                     <- 出力
# (3 + 4) * 2 / 3 ;;
- : int = 4
```

+ (加算)、- (減算)、* (乗算)、/ (商)、mod (余り)

実数（float 型）

```
# 3.14 *. 2. +. 10.0 ;;
- : float = 16.28
```

+.(加算)、-.(減算)、*.(乗算)、/.(除算)、**(冪乗)

文字列（string 型）

```
# "Hi" ;;
- : string = "Hi"
# "Hello " ^ "World!" ;;
- : string = "Hello World!"
```

真偽値（bool 型）

```
# true ;;
- : bool = true
# false ;;
- : bool = false
```

型は厳格に区別される

```
# 3.14 + 5 ;;
Error: This expression has type float
but an expression was expected of type int
```

心得：

- エラーメッセージは丹念に読む。
- 衝突しているふたつの型のうち、どちらが自分の望む型なのかを考える。
- 望む型ではない方を訂正する。

下線部は `int` が正しい → `3 + 5` などとする

下線部は `float` で正しい → `3.14 +. 5.` などとする

変数の定義（let 文）

let 変数 = 式

「式」の実行結果に「変数」という名前をつける。

```
# let pi = 3.1415 ;;
val pi : float = 3.1415
# pi *. 2. ;;
- : float = 6.283
```

OCaml の変数名、関数名はアルファベットの小文字で始まらなくてはならない。（２文字目以降は、大文字、小文字、数字などを自由に使うことができる。）

関数の定義（同じく let 文）

let 関数 引数 ... = 式

「引数 ...」を受け取ったら「式」を計算する「関数」を定義する。

```
# let square x = x *. x ;;
val square : float -> float = <fun>
# square pi ;;
- : float = 9.86902225
# let kyori x y = sqrt (square x +. square y);;
val kyori : float -> float -> float = <fun>
# kyori 3.0 4.0 ;;
- : float = 5.
```

$\cdots square(x) = x^2$

$\cdots square(pi)$

$\cdots kyori(x, y) = \sqrt{x^2 + y^2}$

$\cdots kyori(3.0, 4.0)$

型推論

型を入力しなくても、自動で型を推論してくれる。例：

```
# let kyori x y = sqrt (square x +. square y);;
val kyori : float -> float -> float = <fun>
```

型を書くことなく、型のありがたみを享受できる。

型チェック

プログラム実行前に、型が合っていることを確認する。

実行時に型エラーが起きないことが保証される。

まとめ

- 整数（int 型）。3 + - * / mod
- 実数（float 型）。3.14 +. -. *. /. **
- 文字列（string 型）。"Good " ^ "morning!"
- 真偽値（bool 型）。true false
- 型は厳格に区別される。
- 変数。let pi = 3.14
- 関数。let square x = x *. x
- 関数の型。float -> float
- 型推論と型チェック。