

花子



Agenda

- 出会いは・・・
- どんな子?
- まずは挨拶
- 話しかけてみよー
- 名前を覚えてもらっちゃお!
- 嫌われないようにしなくっちゃ!
- 遊んでみよー
- 友達になれそうですか?

Agenda

Ocamlくんのらぶらぶポイントを説明しつつ Ocamlくんとの距離を縮めていきたい と思います。

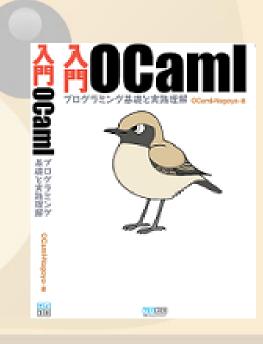


出会いは・・・

2008/8/9のOSC名古屋で聞いた

OCaml-Nagoya のセッションでした。

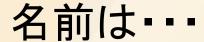
- ・名古屋で活動している OCamlの勉強会グループ
- ・隔週で勉強会を開催
- -2007/05に解説本を出版





知らない人にいきなり話しかけられない 小心者なので・・・

まずは、 ちょっぴり予備知識を仕入れてみます。



OCamlで、略さないとObjective Caml 読み方は、キャムルとか、キャメルとか

属性は・・・

関数型プログラミング言語 型推論を持つ静的型付け言語

生まれは・・・ おフランスの国立INRIA研究所





家系図は・・・

1970年代 ML(Meta Language)

1986年 Caml

+ オブジェクト指向機能

1996年 OCaml

最新は・・・ OCaml 3.10.2



まずは挨拶

コンパイルもできますが、 とりあえずインタープリタで

といっても、デモはしなくって・・・ 実行画面を書き写してきました(^^)v



まずは挨拶

print_string "Hello OCamlくん";;

Hello OCamlくん-: unit = ()



話しかけてみよー

$$#1 + 2;;$$

-: int = 3



型推論

0.1 + . 0.2;;

-: float = 0.30000000000000004



話しかけてみよー

Characters 4-7:

This expression has type float but is here used with type int

名前を覚えてもらっちゃお!

let x = 1;;

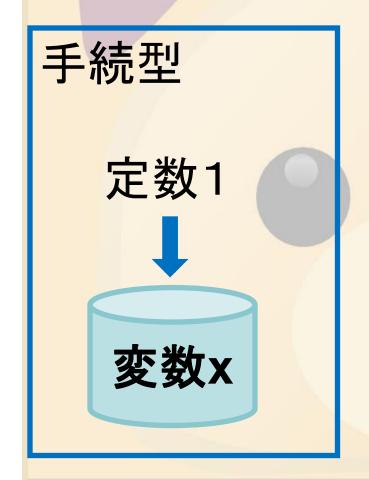
val x : int = 1

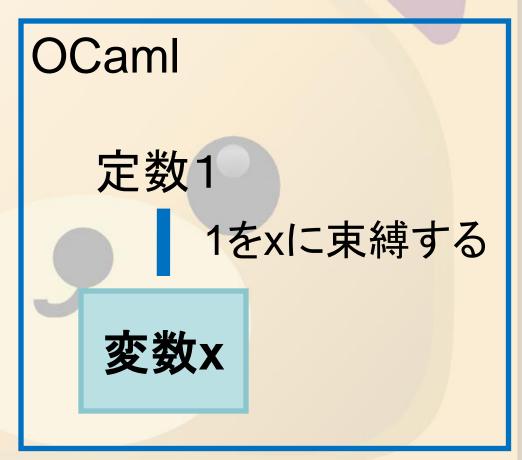
x + 2;;

-: int = 3



名前を覚えてもらっちゃお!





名前を覚えてもらっちゃお!

let x;;

Characters 5-7:

let x;;

 $\Lambda\Lambda$

Syntax error



必ず初期化が必要なので 初期化漏れがない

代入もできない

嫌われないようにしなくっちゃ!

参照型を使うと・・・

```
# let a = ref 1;;
```

val a : int ref = {contents = 1}

$$#a := 1 + 2 ;;$$

- : unit = ()
- #!a;;
- -: int = 3

副作用 aの状態が変わる 参照型、for文



1加算する関数を作ってみる

```
# let aaa = fun x \rightarrow x + 1;;
```

val aaa : int -> int = <fun>

let aaa x = x + 1;;

aaa 3;; # aaa 0.1;;

-: int = 4 Characters 4-7:

aaa 0.1;;

 $\Lambda\Lambda\Lambda$

This expression has type float but is here used with type int



リストの長さを数える関数を作ってみる

let aaa list =
 match list with

$$[] -> 0$$

| [a] -> 1

| [a;b] -> 2

_ -> 3;;



便利なパターンマッチング パターンの漏れも警告 (右辺の型を揃える)

val aaa: 'a list -> int = <fun>



多相型が簡単に作れる

```
# let aaa = function
[] -> 0
    |[a] -> 1
    |[a;b] -> 2
    |_ -> 3;;
```

```
# aaa [];;
-: int = 0
# aaa ["abc"];;
-: int = 1
# aaa [1; 2];;
-: int = 2
# aaa [0.1; 0.2; 0.3; 0.4];;
-: int = 3
```

リストの要素を加算する関数を作ってみる

```
# let sum = function

[] -> 0

| [a] -> a

| [a;b] -> a + b

| _ -> 100;;
```

val sum : int list -> int = <fun>

```
# sum [2; 3];;
-: int = 5
```

```
# sum [1; 0.2];;
Characters 8-11:
sum [1; 0.2];;
```

This expression has type float but is here used with type int



再帰関数を使って リストの全要素を加算する関数を作ってみる

let rec sum = function

[] -> 0

| hd::tl -> hd + sum tl;;

val sum: int list -> int = <fun>

sum [1; 2; 3; 4; 5];;

-: int = 15



```
遊んでみよー
# let rec sum = function
   [] -> 0
  | hd::tl -> hd + sum tl;;
                                -: int = 6
     # sum [1; 2; 3];;
       1 + sum [2; 3]
                                  1 + 5
       2 + sum [3]
                                  2 + 3
                                  3 + 0
       3 + sum []
```



ちょっと作り変えて

```
# let rec sum ans = function
[] -> ans
| hd::tl -> sum (ans + hd) tl;;
```

val sum: int list -> int = <fun>

sum 0 [1; 2; 3; 4; 5];;

-: int = 15



```
遊んでみよー
# let rec sum ans = function
  [] -> ans
 | hd::tl -> sum (ans + hd) tl;;
  # sum 0 [1; 2; 3];;
   sum (0 + 1) [2; 3]
   sum (1 + 2) [3]
                       末尾再帰関数
                       スタックを使用しないように
   sum (3 + 3) []
                       最適化
   -: int = 6
```



末尾再帰関数とは・・・ 再帰呼び出しを右辺でのみ行う 再帰関数の戻り値に計算を行わない

- # let rec sum ans = function
 [] -> ans
 - | hd::tl -> sum (ans + hd) tl;;
- # let rec sum = function
 - [] -> 0
 - | hd :: tl -> hd + sum tl;;



固定値の引数0をなくしてみる

ラボラボ ポイント

関数内関数が使える

val sum : int list -> int = <fun>

sum [1; 2; 3; 4; 5];;

-: int = 15



高階関数を使ってみる

```
# let rec fold_left ff init = function
   [] -> init
 | hd::tl -> fold_left ff (ff init hd) tl;;
val fold_left: ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a = <fun>
# fold_left (+) 0 [1; 2; 3; 4; 5];;
-: int = 15
# fold_left (+.) 0.0 [0.1; 0.2; 0.3; 0.4; 0.5];;
-: float = 1.5
```



let sum list = fold_left (+) 0 list;;

val sum : int list -> int = <fun>

sum [1; 2; 3; 4; 5];;

-: int = 15



便利な高階関数が簡単に作れる 再帰関数なしでリスト処理ができる

let sum list = List.fold_left (+) 0 list;;

val sum: int list -> int = <fun>



標準ライブラリには、 map、fold_leftなど 便利な機能がいっぱい

友達になれそうですか?

関数型っぽい特長をいくつか説明しましたが、 まだまだ他にもらぶらぶポイントはいっぱいあります。

友達になれそうですか?

おもしろそうだな一と思ってもらえたら、 フリーでダウンロードできますので、 ぜひぜひ遊んでみてください。

http://caml.inria.fr/download.en.html

ありがとうございました m(_'_)m

