Programmation Fonctionnelle : des concepts aux applications web (NFP119)

Langages Fonctionnels

Tristan Crolard

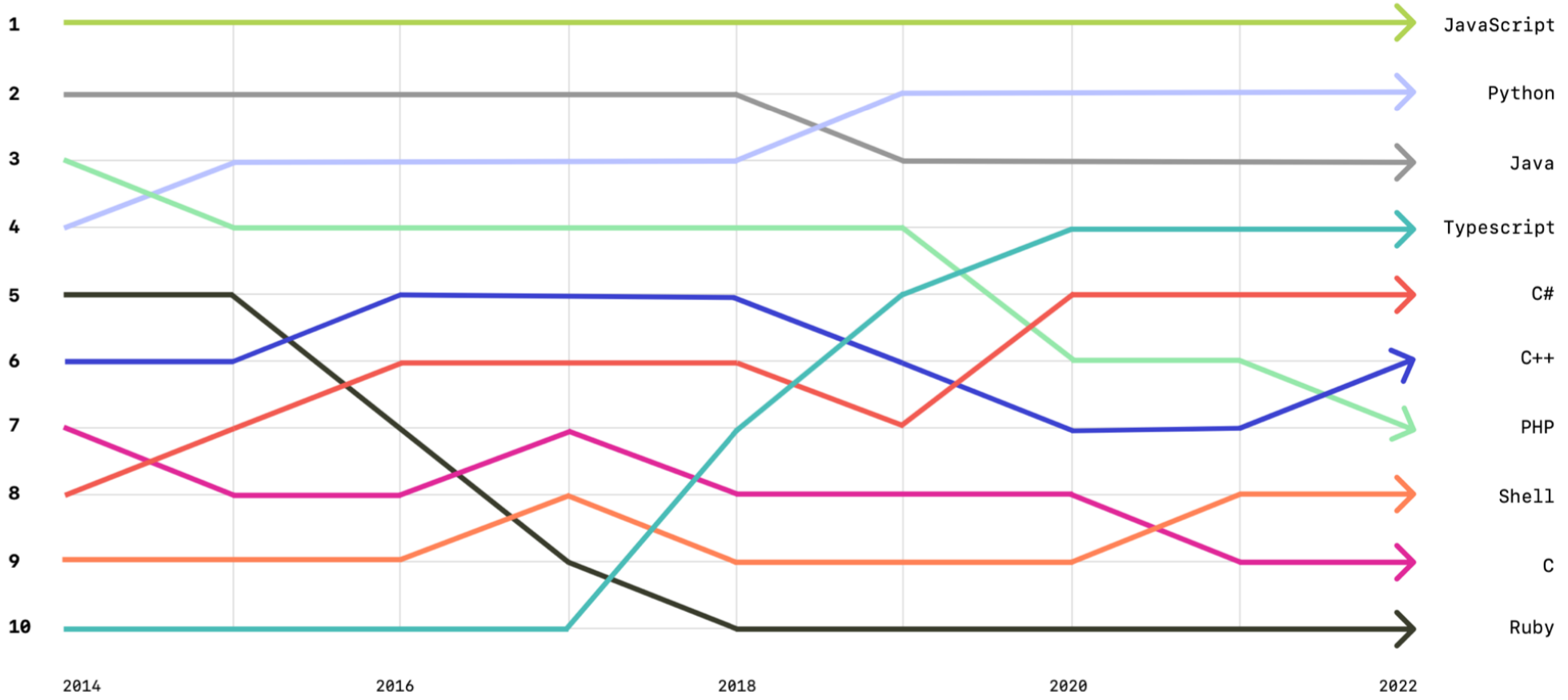
Laboratoire CEDRIC
Equipe « Systèmes Sûrs »

tristan.crolard@cnam.fr

cedric.cnam.fr/sys/crolard

Langages populaires en 2022

https://octoverse.github.com/2022/top-programming-languages



Le paradigme fonctionnel en 2023

	Java	(Java/Type)Script	Python
records	Y (java 17) [2021]	Y	Y (dataclass)
pattern matching	Y (java 21) [2023]	Y (partial)	Y (3.10)
functional programming	Y (java 8) [2014]	Y	Y
exceptions	Y	Y	Y
object-oriented	Y	Y	Y
static type checking	Y	N/Y (gradual)	Y (gradual) [2014]
proper generics	Y (java 5) [2009]	Y	Y (3.12) [2023]

Les langages fonctionnels

- Ils n'ont pas de procédures, uniquement des fonctions.
- Ils n'ont pas d'instructions, uniquement des expressions.

Applications

- Ils sont bien adaptés au prototypage d'applications, à l'écriture de compilateurs et au traitement symbolique. Le code est en général plus concis et plus facile à certifier.

Variantes

- Certains ont un typage statique (comme ML) vérifié à la compilation, d'autres utilisent un typage dynamique (comme LISP/Scheme).
- Certains sont purs (comme Haskell), les autres contiennent des traits impératifs (comme ML et LISP/Scheme).

  $\rightarrow$ références, tableaux, exceptions...
- Certains sont paresseux (comme Haskell). Ils calculent uniquement ce qui est nécessaire.

OCaml

- *OCaml* est un langage fonctionnel avec typage statique (non-paresseux et avec traits impératifs).
- Il est issu du ML (Meta-Language) développé à Edinburgh pour le prouveur LCF (1973), de même que *Standard ML* ou *Microsoft F#* (2005).
- *Standard ML* est standardisé depuis 1990 sous la forme d'une sémantique formelle. Ce standard est révisé en 1997. La bibliothèque standard de *Standard ML* est revisée en 2002.
- *OCaml* ajoute une couche "objet" à ML (non traitée dans ce cours).

Bibliographie OCaml

- B. Pagano, P. Manoury, E.Chailloux : Developpement d'applications avec Objective Caml
- John Whitington : OCaml for the very beginning
- Philippe Narbel : Programmation fonctionnelle, générique et objet: une introduction avec le langage OCaml

Bibliographie ML

- R. Milner, M. Tofte, R. Harper, and D. MacQueen. The Definition of Standard ML (Revised). The MIT Press, 1997.
- R. Milner and M. Tofte. Commentary on Standard ML. The MIT Press, 1990.
- E. R. Gansner and J. H. Reppy. The Standard ML Basis Library. Cambridge University Press, 2004. Disponible en ligne :
  `http://www.standardml.org/Basis`
- L. Paulson. ML for the Working Programmer. Cambridge University Press, second edition, 1996.
- J. D. Ullman. Elements of ML Programming (ML 97 edition). Prentice-Hall, 1998.
- S. Gilmore. Programming in Standard ML '97: An On-line Tutorial. Disponible en ligne :
  `http://www.dcs.ed.ac.uk/home/stg/NOTES/notes.html`

Types simples

- Toutes les expressions ML ont un type.
- Les types de bases de ML sont `int`, `char`, `string`, `boolean` et `float`.
- A partir de valeurs de ces types de bases, il est possible de construire des expressions complexes en utilisant les couples (triplets...) et les listes.
- Les listes sont homogènes : tous leurs éléments ont le même type.

Exemples

Produits

```
(2, "Andrew") : int * string
(true, 42, 'x') : bool * int * char
((4, 2), (7, 3)) : (int * int) * (int * int)
```

Listes

```
[1; 2; 3] : int list
["Andrew"; "Ben"] : string list
[(2, 3); (2, 2); (9, 1)] : (int * int) list
[[]; [1]; [1; 2]] : int list list
```

Déclaration de valeurs (mot clé let)

Déclaration d'une variable

```
# let a = 12;;
val a : int = 12
```

Déclaration d'un couple de variables

```
# let (d,e) = (2,"two");;
val d : int = 2
val e : string = "two"
```

Inférence des types

Le système infère lui-même le type des expressions, mais vous pouvez aussi les préciser (ils sont alors vérifiés).

```
# let a : int = 12;;

val a : int = 12

# let a : char = 12;;

Error: This expression has type int but an expression was expected of
type char
```

Fonctions (mot clé let)

```
# let double x = 2 * x;;

val double : int -> int = <fun>

# let inc x = x+1;;

val inc : int -> int = <fun>

# let adda s = s ^ "a";;

val adda : string -> string = <fun>
```

Remarque. De même, le système infère le type des fonctions. Le typage est inféré et/ou vérifié à la compilation. Ce n'est pas du typage dynamique !

Fonctions et couples

```
# let add (x,y) = x + y;;
val add : int * int -> int = <fun>
# let double0 x = (2 * x, 0);;
val double0 : int -> int * int = <fun>
# let ab (x,y) = (x = "a", y = "b");;
val ab : string * string -> bool * bool = <fun>
```

Utilisation

```
# let (x,y) = double0 3;;
val x : int = 6
val y : int = 0
```

Pour ignorer une partie du résultat

```
# let (k,_) = double0 3;;
val k : int = 6
```

Les listes

- la liste vide s'écrit `[]`
- l'opérateur infixe `::` ajoute un élément en tête de la liste.

Exemples

Les expressions suivantes ont la même valeur à gauche et à droite :

```
[]                          []
1::[]                       [1]
2::(1::[])                  [2; 1]
3::(2::(1::[]))             [3; 2; 1]
4::[3; 2; 1]                [4; 3; 2; 1]
[5; 4; 3]@[2; 1]            [5; 4; 3; 2; 1]
```

Fonctions et listes

```
# let f (x, l): char list = (x::l);;
val f : char * char list -> char list = <fun>
# let g (l1, l2): int list = l1 @ l2;;
val g : int list * int list -> int list = <fun>
```