

Prof: letonzey@pro.fr

La programmation fonctionnelle utilise des mécanismes totalement différents.

Exemple:  $rs : liste \rightarrow liste$  (=fonction)

- $rs(\emptyset) = \emptyset$
- $rs(x \text{ puis } l) = rs(l \text{ filtrée par } < x) \text{ puis } x \text{ puis } rs(l \text{ filtrée par } \geq x)$

A.N:  $rs(1, 3, 2, 5)$

$\hookrightarrow rs(\emptyset) \text{ et } rs(3, 2, 5)$   
 $\hookrightarrow \emptyset \text{ et } rs(\emptyset) \text{ et } rs(2, 5)$   
 $\vdots$   
 $\hookrightarrow 1 \quad 2 \quad 3 \quad 5$

On fait  
des  
remplacement  
d'expressions

•  $rs$  ne modifie pas  
↳ elle **CONSTRUIT**

•  $rs$  est défini par induction

Remarque: Garbage collector en Ocaml

Définition: Une fonction est pure  
ssi elle ne modifie pas son  
environnement

$$\forall x, y \text{ tq } x = y, F(x) = F(y)$$

POO c'est "Modéliser par classes"  
Fonctionnel c'est "instancier différemment l'existant"

En Ocaml, l'opération  $x :: xs$   
fait  $[1, 2, \dots, n]$   
 $\quad \quad \quad \underbrace{\hspace{1cm}}_{x} \quad \underbrace{\hspace{1cm}}_{[xs]}$

Exemples de code:

Sites pour coder: [try.ocamlpro.com](http://try.ocamlpro.com)  
[sketch.sh](http://sketch.sh)

Dans un

```
if ...  
then  
  type1  
else  
  type2
```

type1 et type2 doivent être égaux

- Le type string existe.

"abc" ^ "def" donne "abcdef" et "abc".[0] donne 'a'

- Char avec ('a') simples mots

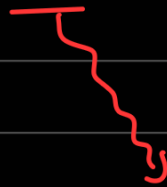
Si on a

let x = 5;;  
→ 5

let x = 10 in x + x;;  
→ 20

x;;

→ 5



Rend le  
x précédent

LOCAL

Donc

let x = 5;;

let z = 2 in x \* z;;

z

Ne fonctionne pas.

Les commentaires s'encadrent avec (\* ... \*)  
Ex: (\* Description \*)

## Les Fonctions:

Introduites par Fun

Fun  $x \rightarrow 2^x x$  ;;  $\rightarrow$  Comme une fonction anonyme lambda  
 $\rightarrow$  fonction double

Mais on ne peut pas l'appeler...  
Sauf si on l'applique directement,

$(\text{fun } x \rightarrow 2^x x)(10)$   
 $\rightarrow 20$

Pour réutiliser :  $\text{let double } x = 2^x x$  ;;  
argument(s)  $\rightarrow$  expression retournée

ou  $\text{let } (\text{double} = \text{fun } x \rightarrow 2^x x)$   
nom  $\rightarrow$  déclaration

et l'appel est :

$\text{double}(10)$  ou  $\text{double } 10$   
 $\rightarrow 20$   $\rightarrow 20$

Si l'arg est une constante positive/variable, on peut choisir la deuxième façon

Sinon (ex  $\text{double } 10 + 10$  ;;)  
 $\rightarrow 30$  et non  $40$   
on écrivant  $\text{double}(10 + 10)$

En pratique on parenthèse partout

```
if True
  then
    fun x → 2x x      est un code
  else                valable
    fun x → x / 2
```

Attention la "vraie" comparaison est un  
simple égal...  
la comparaison des pointeurs  
est ==

La récursivité se note avec un **rec**

```
let rec fact n =
  if n = 0
  then
    1
  else
    fact(n-1) * n
```