

	OCaml	Python
test d'égalité	=	==
test de différence	<>	!=
division euclidienne	/	//
modulo	mod	%
et	&&	and
ou		or

Opérateurs en OCaml/Python

	OCaml
Définition	let r = ref ...
Accéder à la valeur	!r
Modifier la valeur	r := ...

Références

- () est une valeur de type **unit**, qui signifie rien.
- Si $f : 'a \rightarrow 'b \rightarrow 'c$ alors f prend deux arguments de type $'a$ et $'b$ et renvoie un résultat de type $'c$.
De plus, $f\ x$ (application partielle) est la fonction de type $'b \rightarrow 'c$ qui à y associe $f\ x\ y$.

- On crée souvent une liste avec une fonction récursive :

```
let rec range n = (* renvoie [n - 1; ...; 1; 0] *)
  if n = 0 then []
  else (n-1)::range (n - 1)
```

On peut éventuellement utiliser une référence sur une liste :

```
let range n =
  let l = ref [] in (* moins idiomatique *)
  for i = n - 1 downto 0 do
    l := i::!l
  done; !l
```

- Impossible de modifier une liste 1.**
 $e::l$ renvoie une **nouvelle liste** mais ne modifie pas l .
`let l = [] in ...` ne sert à rien.
- Impossible d'écrire $l.(i)$ pour une liste l :** il faut utiliser une fonction récursive pour parcourir une liste.

```
let rec appartient e l = match l with
| [] -> false
| t::q -> e = t || appartient e q
```

- Chaque cas d'un **match** sert à **définir de nouvelles variables**, et ne permet pas de comparer des valeurs.

Exemple : Dans `appartient`, il ne faut pas écrire
`| e::q -> ...` (ceci remplacerait le `e` en argument).
 Pour tester l'égalité : `| t::q -> if t = e then ...` ou
`| t::q when t = e -> ...`.

- Les indices d'un tableau à n éléments vont de 0 à $n - 1$:

```
let appartient e t =
  let r = ref false in
  for i = 0 to n - 1 do
    if t.(i) = e then r := true
  done; !r
```

- Impossible de renvoyer une valeur à l'intérieur d'une boucle** (pas de **return** en OCaml) :

```
let appartient e t =
  for i = 0 to n - 1 do
    if t.(i) = e then true (* NE MARCHE PAS !!! *)
  done; false
```

- Types union et enregistrement :

```
(* type union : l'un OU l'autre *)
type 'a option = None | Some of 'a
(* on utilise match (et fonction récursive)
sur les types union *)
let add x y = match x, y with
| None, _ | _, None -> None
| Some x1, Some y1 -> Some (x1 + y1)

(* type enregistrement : l'un ET l'autre *)
type fraction = { num : int; denum : int }
let f = { num = 1; denum = 4 }
f.denom (* donne 4 *)
```

Remarque : l'égalité (avec `=`) est automatiquement définie avec un nouveau type. Exemple : si t_1 et t_2 sont des arbres binaires, alors $t_1 = t_2$ vaut **true** si $t_1 = V$ et $t_2 = V$ ou si $t_1 = N(r_1, g_1, d_1)$, $t_2 = N(r_2, g_2, d_2)$ avec $r_1 = r_2$, $g_1 = g_2$ et $d_1 = d_2$.

- Quelques fonctions utiles (avec les équivalents sur **Array**) :
 - `Array.make_matrix n p e` renvoie une matrice $n \times p$ dont chaque élément est e
 - `List.iter f l` appelle f sur chaque élément de l
 - `List.map f [a1; ...; an]` renvoie $[f\ a1; \dots; f\ an]$
 - `List.filter f l` renvoie la liste des e tels que $f\ e$
 - `List.exists f l` et `List.for_all f l`.

Type	Exemple	Obtenir valeur	Modification	Taille	Création
array	<code>[1; 2] : int array</code>	<code>t.(i)</code>	<code>t.(i) <- ...</code>	<code>Array.length</code> (en $O(1)$)	<code>Array.make n e</code> (en $O(n)$)
string	<code>"abc" : string</code>	<code>s.[i]</code>		<code>String.length</code> (en $O(1)$)	<code>String.make n e</code>
list	<code>[1; 2] : int list</code>	Fonction récursive		<code>List.length</code> (en $O(n)$)	
tuple	<code>(1, "abc", [1; 2]) : int*string*int list</code>	<code>let a, b, c = t</code>			

Remarque : une « liste » Python est en réalité un tableau.