Aide mémoire OCaml

Déclarations et instructions	
Commentaires	(* ... *)
voir la signature d'une fonction	*fonction ;;*
définition d'une valeur	`let` v `=` *expression*
récursive	`let rec` v `=` ...
local	`let` v `=` ... `in` *expression*
définitions simultanées	`let` u `=` ... `and` v `=` ...
successives	`let` u `=` ... `in let` v `=` ...
variable modifiable (référence)	`let` v `= ref` *expression*
valeur d'une référence	`!`v
modification d'une référence	v `:=` ...
fonction sans argument	`let` f () `=` ...
à un argument	`let` f x `=` ...
à n arguments	`let` f x_1 ... x_n `=` ...
expression conditionnelle	`if` *booléen* `then` *expression* `else` *expression*
choix multiple (ou filtrage)	`match` *valeur* `with` | *motif-1* `->` *expression-1* | *motif-2* `->` *expression-2* ⋮ | *motif-p* `->` *expression-p* | _ `->` *expression par défaut*
choix multiple avec conditions	| *motif* `when` *cond* `->` *expression*
ne rien faire	`()`
calculs en séquence	`begin` *séquence de (expressions* `;` *)* `end`
boucle croissante	`for` i `=` *début* `to` *fin* `do` ... `done`
boucle décroissante	`for` i `=` *début* `downto` *fin* `do` ... `done`
boucle conditionnelle	`while` *condition* `do` ... `done`
déclencher une erreur	`failwith` "*message*"
Fonctions polymorphes	
comparaisons	`< <= = <> >= >`
minimum, maximum	`min` a b, `max` a b
Expressions booléennes	
vrai, faux	`true, false`
et, ou, non	`&&, ||, not`
changement de type : booléen ↦ chaîne de caractères	`string_of_bool` : `bool -> string`
chaîne de caractères ↦ booléen	`bool_of_string` : `string -> bool`
Expressions entières	
opérations arithmétiques	`+ - * /`
modulo	`mod`
valeur absolue	`abs`
entier précédent, suivant	`pred, succ`
opérations bit à bit	`land lor lxor lnot`
décalage de bits	`lsl lsr asr`
changement de type : entier ↦ chaîne	`string_of_int` : `int -> string`
chaîne ↦ entier	`int_of_string` : `string -> int`
entier aléatoire entre 0 et $n-1$	`Random.int` n : `int -> int`
Expressions réelles	
opérations arithmétiques	`+. -. *. /.`
puissance	`**`
fonctions mathématiques	`abs_float exp log log10 sqrt sin cos tan asin acos atan sinh cosh tanh`
réel aléatoire entre 0 et a	`Random.float` a : `float -> float`
changement de types : réel ↦ chaîne	`string_of_float` : `float -> string`
réel ↦ entier	`int_of_float` : `float -> int`
chaîne ↦ réel	`float_of_string` : `string -> float`
entier ↦ réel	`float_of_int` : `int -> float`

Listes : module `List`	
liste	`[x;y;z ...]`
liste vide	`[]`
tête, queue dans un filtrage	t::suite
tête	`List.hd : 'a list -> 'a`
queue	`List.tl : 'a list -> 'a list`
longueur d'une liste	`List.length : 'a list -> int`
concaténation	*lst1* `@` *lst2* `List.append : 'a list -> 'a list -> 'a list`
image miroir	`List.rev : 'a list -> 'a list`
concaténation d'un miroir d'une liste avec une autre	`List.rev_append : 'a list -> 'a list -> 'a list`
concaténation d'une liste de listes	`List.concat : 'a list list -> 'a list`
appliquer une fonction	`List.map : ('a -> 'b) -> 'a list -> 'b list`
itérer un traitement à partir d'une liste	`Liste.iter : ('a -> unit) -> 'a list -> unit`
test d'appartenance	`List.mem : 'a -> 'a list -> bool`

itérer une opération	`List.fold_left` *op* a [b1 ;b2 ;b3]
	= op (op (op a b1) b2) b3
	`('a -> 'b -> 'a) -> 'a -> 'b list -> 'a`
	`List.fold_right` *op* [a1 ;a2 ;a3] b
	= op a1 (op a2 (op a3 b))
	`('a -> 'b -> 'b) -> 'a list -> 'b -> 'b`

test de présence : au moins un	`List.exists : ('a -> bool) -> 'a list -> bool`
: tous	`List.for_all ('a -> bool) -> 'a list -> bool`
Tableaux : module `Array`	
tableau	`[|x;y;z;...|]`
tableau vide	`[| |]`
i-ème élément	v.(i)
modification	v.(i) `<-` *qqch*
longueur d'un tableau	`Array.length : 'a array -> int`
création d'un tableau	`Array.make : int -> 'a -> 'a array`
création à l'aide d'une fonction d'entiers	`Array.init : int -> (int -> 'a) -> 'a array`
création d'une matrice	`Array.make_matrix : int -> int -> 'a -> 'a array array`
concaténation de deux tableaux	`Array.append : 'a array -> 'a array -> 'a array`
concaténation d'une liste de tableaux	`Array.concat : 'a array list -> 'a array`
extraction	`Array.sub` *t début lg* `: 'a array -> int -> int -> 'a array`
copie	`Array.copy : 'a array -> 'a array`
appliquer une fonction	`Array.map : ('a -> 'b) -> 'a array -> 'b array`
itérer un traitement avec les valeurs d'un tableau	`Array.iter : ('a -> unit) -> 'a array -> unit`
tableau $\mapsto$ list	`Array.to_list : 'a array -> 'a list`
liste $\mapsto$ tableau	`Array.of_list : 'a list -> 'a array`

itérer une opération	`Array.fold_left` *op* a `[|`*b1 ;b2 ;b3*`|]`
	= op (op (op a b1) b2) b3
	`('a -> 'b -> 'a) -> 'a -> 'b array -> 'a`
	`Array.fold_right` *op* `[|`*b1 ;b2 ;b3*`|]` b
	= op a1 (op a2 (op a3 b))
	`('b -> 'a -> 'a) -> 'b array -> 'a -> 'a`

Chaînes de caractères : modules Char, String	
caractère	*'x'*
code ASCII du caractère	`Char.code`
chaîne de caractères	"azerty"
i-ème caractère	*chaîne*`.[i]`
modification du i-ème caractère	**impossible depuis 4.02**
longueur d'une chaîne	`String.length : string -> int`
création avec un seul caractère	`String.make : int -> char -> string`
extraction	`String.sub` *chaîne début longueur*
concaténation de deux chaînes	*chaîne1*`^`*chaîne2*
caractère $\mapsto$ chaîne	`Char.escaped : char -> string`

Pour d'autres fonctions plus poussées cf « **The OCaml system** », qu'on trouve sur le site http://caml.inria.fr/pub/docs/manual-ocaml/, en faisant attention la version de OCaml