Aide mémoire de CAML

Déclarations et instructions

commentaires	`(* ... *)`
définition d'une valeur	`let v = expression`
récursive	`let rec v = ...`
locale	`let v = ... in expression`
	`expression where v = ...`
définitions parallèles	`let v = ... and w = ...`
successives	`let v = ... in let w = ...`
variable modifiable	`let v = ref(expression)`
valeur d'une référence	`!v`
modification d'une référence	`v := ...`
fonction sans argument	`let f() = ...`
fonction à un argument	`let f x = ...`
fonction à n arguments	`let f x1 .. xn = ...`
expression conditionnelle	`if ... then expr-vrai else expr-faux`
choix multiple	`match valeur with`

```
| motif-1 -> expression-1
| motif-2 -> expression-2
  ...
| motif-n -> expression-n
| _       -> expression-par-défaut
```

ne rien faire	`()`
calculs en séquence	`begin ... 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"`

Expressions booléennes

vrai, faux	`true false`
et, ou, non	`& or not`
comparaison	`<  <=  =  <>  >=  >`
booléen ⟼ chaîne	`string_of_bool`
chaîne ⟼ booléen	`bool_of_string`

Expressions entières

opérations arithmétiques	`+  -  *  /`
modulo	`mod`
valeur absolue	`abs`
entier précédent, suivant	`pred succ`
minimum et maximum	`min a b,  max a b`
opérations bit à bit	`land lor lxor lnot`
décalage de bits	`lsl lsr asr`
entier ⟼ chaîne	`string_of_int`
chaîne ⟼ entier	`int_of_string`
entier aléatoire entre 0 et $n-1$	`random__int(n)`

Expressions réelles

opérations arithmétiques	`+.  -.  *.  /.`
puissance	`**` ou `**.`
minimum et maximum	`min a b,  max a b`
fonctions mathématiques	`abs_float exp log sqrt sin cos tan`
	`sinh cosh tanh asin acos atan atan2`
réel ⟼ chaîne	`string_of_float`
réel ⟼ entier	`int_of_float`
chaîne ⟼ réel	`float_of_string`
entier ⟼ réel	`float_of_int`
réel aléatoire entre 0 et a	`random__float(a)`

Expressions rationnelles

utiliser les rationnels	`#open "num"`
opérations arithmétiques	`+/  -/  */  //  **/`
	`minus_num quo_num mod_num square_num`
comparaison	`</  <=/  =/  <>/  >=/  >/`
minimum et maximum	`min_num a b,  max_num a b`
valeur absolue	`abs_num`
numérateur, dénominateur	`numerator_num denominator_num`
simplifier une fraction	`normalize_num`
simplification automatique	`arith_status__set_normalize_ratio true`
partie entière	`floor_num round_num ceiling_num`
rationnel ⟼ chaîne	`string_of_num`
rationnel ⟼ entier	`int_of_num`
rationnel ⟼ réel	`float_of_num`
chaîne ⟼ rationnel	`num_of_string`
entier ⟼ rationnel	`num_of_int`
réel ⟼ rationnel	`num_of_float`

Listes

liste	`[x; y; z; ...]`
liste vide	`[ ]`
tête et queue	`hd tl,  x :: suite`
longueur d'une liste	`list_length`
concaténation	`@`

- image miroir : `rev`
- appliquer une fonction : `map fonction liste`
- itérer un traitement : `do_list traitement liste`
- test d'appartenance : `mem élément liste`
- test de présence : `exists prédicat liste`, `for_all prédicat liste`
- recherche d'un élément : `index élément liste`
- opérations ensemblistes : `union intersect subtract`
- tri : `sort__sort ordre liste`
- association : `assoc b [(a,x); (b,y); (c,z); ...] = y`
- itérer une opération : `it_list op a [x; y; z] = op (op (op a x) y) z`, `list_it op [x; y; z] a = op x (op y (op z a))`

Vecteurs

- vecteur : `[|x; y; z; ...|]`
- vecteur vide : `[| |]`
- i-ème élement : `v.(i)`
- modification : `v.(i) <- qqch`
- longueur d'un vecteur : `vect_length`
- création : `make_vect longueur valeur`
- création d'une matrice : `make_matrix n p valeur`
- extraction : `sub_vect vecteur début longueur`
- concaténation : `concat_vect`
- copie : `copy_vect`
- appliquer une fonction : `map_vect fonction vecteur`
- itérer un traitement : `do_vect traitement vecteur`
- vecteur ⟼ liste : `list_of_vect`
- liste ⟼ vecteur : `vect_of_list`

Chaînes de caractères

- caractère : `‘x‘`
- chaîne de caractères : `"xyz..."`
- i-ème caractère : `chaîne.[i]`
- modification : `chaîne.[i] <- qqch`
- longueur d'une chaîne : `string_length`
- création : `make_string longueur caractère`
- caractère ⟼ chaîne : `make_string 1 caractère`
- extraction : `sub_string chaîne début longueur`
- concaténation : `ch1 ^ ch2`, `concat [ch1; ch2; ch3; ...]`

Graphisme

- utiliser le graphisme : `#open "graphics"`
- initialiser la fenêtre graphique : `open_graph ""`
- refermer la fenêtre : `close_graph()`
- effacer la fenêtre : `clear_graph()`
- position du crayon : `current_point()`
- changer la couleur du crayon : `set_color couleur`
- couleurs : `black white red green blue yellow cyan magenta`, `rgb r g b`
- changer l'épaisseur du crayon : `set_line_width épaisseur`
- tracer un point : `plot x y`
- déplacer, crayon levé : `moveto x y`
- crayon baissé : `lineto x y`
- tracer un cercle : `draw_circle x y rayon`
- écrire un texte : `draw_string "texte"`
- peindre un rectangle : `fill_rect x y largeur hauteur`
- un polygone : `fill_poly [| (x0,y0); (x1,y1); ... |]`
- un disque : `fill_circle x y rayon`
- attendre un évènement : `read_key()`, `wait_next_event [ev1; ev2; ...]`

Entrées-sorties au terminal

- impression de valeurs : `print_int print_float num__print_num print_char print_string print_endline`
- changer de ligne : `print_newline()`
- impression formattée : `printf__printf format valeurs`
- lecture de valeurs : `read_int read_float read_line`

Entrées-sorties dans un fichier

- ouverture en lecture : `let canal = open_in "nom"`
- en écriture : `let canal = open_out "nom"`
- lecture : `input_char input_line input_byte input_value`
- écriture : `output_char output_string output_byte output_value flush`
- fermeture : `close_in close_out`

Commande de l'interpréteur

- tracer une fonction : `trace "fonction"`
- ne plus tracer : `untrace "fonction"`
- utiliser une fonction d'impression : `install_printer "fonction"`
- ne plus l'utiliser : `remove_printer "fonction"`
- charger un fichier source : `load "nom"`, `include "nom"`
- charger un module compilé : `load_object "nom"`
- nom complètement qualifié : `module__nom`
- utiliser les noms courts d'un module : `#open "module"`
- ne plus les utiliser : `#close "module"`
- ajouter un répertoire de recherche : `directory "chemin"`
- quitter l'interpréteur : `quit()`