Aide-mémoire OCaml

Booléens

et, ou, non, vrai, faux	`&&, ||, not, true, false`

Entiers

comparaison (aussi pour flottants)	`< <= = <> >= >`
opérations arithmétiques	`+ - * /`
reste de la division euclidienne	`mod`
valeur absolue	`abs`
entier précédent, suivant	`pred, succ`
minimum, maximum de deux entiers	`min, max`
entier aléatoire de $[\![0, n[\![$	`Random.int n`

Remarque :

- `mod` n'est pas le vrai reste de la division euclidienne : il peut être négatif lorsque son premier argument est négatif.
- Pas de puissance pour les entiers.

Flottants

opérations arithmétiques	`+. -. *. /.`
puissance	`** ou **.`
minimum, maximum de deux flottants	`min a b, max a b`
fonctions mathématiques	`abs_float exp log sqrt sin cos tan`
flottant aléatoire dans $[0, a[$	`Random.float a`
conversion entier → flottant	`float_of_int`
conversion flottant → entier	`_int_of_float`

Listes (persistantes)

module List : faire précéder les commandes suivantes de « `List.` », sauf les trois premières.

liste vide	`[]`
le constructeur des listes	`::`
liste déjà remplie	`[x; y ; z;...]`
tête, queue	`hd, tl`
longueur	`length`
concaténation	`@`
retourner une liste	`rev`
tri	`sort relation_d_ordre liste`
test de présence	`mem élément liste`
test « pour tout »	`for_all prédicat liste`
test « il existe »	`exists prédicat liste`
garder les éléments d'une liste vérifiant une condition :	`filter prédicat liste`
appliquer une fonction à chaque élément	`map fonction liste`
appliquer une procédure à chaque élément	`iter traitement liste`
itérer un opérateur	`fold_right op a [x;y;z]` qui renvoie `op ( op ( op a x) y) z` ou : `fold_left op [x;y;z] a` qui renvoie `op x ( op y ( op z a )`

Tableaux (mutables)

Module Array, faire précéder les commandes de «`Array.`»

créer un tableau	`make longueur valeurInitiale`
créer un tableau à l'aide d'une fonction	`init`
tableau déjà rempli	`[|x; y; z;...|]`
i-ème élément	`t.(i)`
modifier le i-ème élément	`t.(i) <- nouvelle valeur`
longueur	`length`

extraction	`sub t debut longueur`
concaténation	`concat`
vraie copie	`copy`
appliquer une fonction	`map fonction tableau`
appliquer une procédure	`iter procédure tableau`

Chaînes de caractères

Module String

caractère	`'x'`
création	`make longueur carInitial`
chaîne de caractères	`"blabla"`
i-ème élément	`c.[i]`
longueur	`length`
extraction	`sub chaîne début longueur`
concaténation	`c1 ^ c2` ou `concat [c1;c2;...]`

Remarque : Les chaînes de caractères étaient mutables mais ne le sont plus pour les dernières versions d'Ocaml.

Piles (mutables)

module Stack

pile vide	`create ()`
empiler	`push élément pile`
dépiler	`pop pile`
voir si une pile est vide	`is_empty pile`

Files d'attente (mutables)

Module Queue

pile vide	`create ()`
ajouter	`add élément file`
extraire	`take file`
est vide ?	`is_empty file`

Tables de hachage (mutables)

module Hashtbl

dictionnaire vide (indiquer une taille quelconque)	`create taille`
insérer une valeur	`add dico clef valeur`
chercher une valeur	`find dico clef`
tester si une clef est présente	`mem dico clef`
supprimer une valeur	`remove dico clef`
fonction de hachage	`hash`

Lecture écriture de fichier

Ouvrir un fichier	`let entree = open_in "chemin du fichier"`
Lire une ligne	`input_line entree`
Fermer le fichier	`close_in entree`
Ouvrir en écriture	`let sortie = open_out "chemin du fichier"`
Écrire une chaîne	`output string sortie texte`
Fermer le fichier	`close_out sortie`

Graphisme

Charger le module	`#load "graphics.cma";;`
Ouvrir une fenêtre graphique	`Gaphics.open_graph " 800x600";;`
Déplacer le crayon	`Graphics.moveto x y;;`
Tracer un trait	`Graphics.lineto x y;;`
Afficher un texte	`Graphics.draw_string "blabla";;`

Directive (spécifiques au toplevel)

Rajouter un répertoire où chercher les fichiers ..... `#directory "chemin du repertoire";;`
Lire un fichier .ml ..... `#use "chemin du fichier";;`
Charger un module ..... `#load "module.cma" ; ;`

Graphisme (module `Graphics`)

Ouvrir une fenêtre graphique ..... `open_graph " 800*600";;`
Fermer la fenêtre ..... `close_graph () ; ;`
Choisir une couleur ..... `set_color couleur ; ;`
Aller à une position ..... `moveto x y ; ;`
Aller à une position en traçant un trait ..... `lineto x y ; ;`
colorier un rectangle : ..... `fill_rect x y largeur hauteur`
Écrire un texte : ..... `draw_string texte`

Dans Wincaml le module `Graphics` devrait être présent. Dans linux, `opam install graphics`, et `#require "graphics" ; ;` au début du document. Dans certains cas, il faudra installer ocamlfind (`opam intall ocamlfind`) et rajouter `#use "topfind" ; ;` pour permettre à Caml de trouver le module.

Grands entiers, et fractions

Charger la biliothèque ..... `#load nums.cma;;` puis `open Num;;`
Opération arithmétiques ..... `+/, */, -/, //`
Conversions ..... `int_of_num, num_of_int`
Reste de la division euclidienne ..... `mod_num`
Quotient ..... `quo_num`