

Déclarations et instructions

commentaires	(* ... *)
définition d'une valeur	let v = expression
récursive	let rec v = ...
locale	let v = ... in expression
définitions simultanées	let v = ... and w = ...
successives	let v = ... in let w = ...
référence (modifiable)	let v = ref expression
valeur d'une référence	!v
modifier 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 condition then expression ₁ else expression ₂
filtrage simple	match valeur with motif ₁ -> expression ₁ ... motif _k -> expression _k _ -> expression-par-défaut
filtrage avec garde	match valeur with motif ₁ when condition ₁ ->
ne rien faire	()
suite d'instructions	begin ... end
boucle croissante	for i = début to fin do ... done
conditionnelle	while condition do ... done
exception personnalisée	failwith "message d'erreur"
fonction anonyme	function x -> ...
à plusieurs arguments	fun x y -> ...

Expressions booléennes

vrai, faux	true, false (en minuscules)
et, ou, non	& , , not
comparaison	< , <= , = , <> , >= , >
booléen \mapsto chaîne	string_of_bool
chaîne \mapsto booléen	bool_of_string

Expressions entières

opérations arithmétiques	+, -, *, / , mod
valeur absolue	abs
minimum, maximum	min a b, max a b
entier \mapsto chaîne	string_of_int
chaîne \mapsto entier	int_of_string
entier aléatoire de {0,...,n-1}	Random.int n
afficher un entier	print_int

Expressions flottantes

opérations arithmétiques	+, -, *, , / .
puissance	**
minimum, maximum	min a b, max a b
fonctions mathématiques	abs_float, exp, log, sqrt, sin, cos, tan, sinh, cosh, tanh, asin, acos, atan, atan2
flottant \mapsto entier	int_of_float
entier \mapsto flottant	float_of_int
flottant \mapsto chaîne	string_of_float
chaîne \mapsto flottant	float_of_string
flottant aléatoire entre 0 et a	Random.float a
afficher un flottant	print_float

Listes

liste	[x ; y ; z ; ...]
liste vide	[]
tête et queue	List.hd, List.tl, h : :t
longueur d'une liste	List.length
concaténation	@
image miroir	List.rev
appliquer une fonction	List.map fonction liste
appliquer un traitement	List.iter traitement liste
itérer une opération	List.fold_left f e liste
test d'appartenance	List.mem élément liste
test de présence	List.exists prédicat liste, List.for_all prédicat liste
indice d'un élément	List.nth liste élément
tri	Sort.list ordre liste

Tableaux

tableau	[[x ; y ; z ; ...]]
tableau vide	[]
i-ème élément	v.(i)
modification	v.(i) <- qqch
longueur d'un tableau	Array.length
création d'un tableau d'une matrice	Array.make longueur valeur Array.make_matrix n p valeur
extraction	Array.sub tableau début longueur
concaténation	Array.append tableau ₁ tableau ₂
copie	Array.copy tableau
appliquer une fonction	Array.map fonction tableau
appliquer un traitement	Array.iter traitement tableau
tableau \mapsto liste	Array.to_list
liste \mapsto tableau	Array.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 d'une chaîne	String.make longueur caractère
caractère \mapsto chaîne	String.make 1 caractère
extraction	String.sub chaîne début longueur
concaténation	chaîne ₁ ^ chaîne ₂ , String.concat
afficher un caractère	print_char
une chaîne	print_string

Commande de l'interpréteur

tracer une fonction	#trace fonction
ne plus tracer	#untrace fonction
charger une bibliothèque	open nom, #load "nom"
charger un fichier source	#use "nom"

Entrées-sorties

impression formatée	Printf.fprintf
impression de valeurs	print_int, print_float, print_char, print_string
changer de ligne	print_newline ()
lecture de valeurs	read_int, read_float, read_line

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, closeout