The "Unknown:"s below indicate that an entry is incomplete.

- either the entry exist in the language, and please tell.
- either the entry doesn't exist in the language, and please tell so. The entry will be marked as such and won't appear as missing anymore.

- Category: Object Oriented, Functional, Statically typed

- Various

`nothing needed`	breaking lines (useful when end-of-line and/or indentation has a special meaning)
`(* ... *)`	commenting (nestable)
`< > <= >=`	comparison
`min / max`	comparison (min / max (binary or more))
`compare`	comparison (returns 3 values (i.e. inferior, equal or superior))
`(** ... *)`	documentation comment
`= <>`	equality / inequality (deep)
`== !=`	equality / inequality (shallow)
`Gc.full_major()`	force garbage collection
`( ... )`	grouping expressions
`begin ... end`	grouping expressions
`case-sensitive`	tokens (case-sensitivity (keywords, variable identifiers...))
`[_A-Z][_a-zA-Z0-9']*`	tokens (constant regexp (if different from variable identifier regexp))
`[_a-z][_a-zA-Z0-9']*`	tokens (type regexp (if different from variable identifier regexp))
`[_a-z][_a-zA-Z0-9']*`	tokens (variable identifier regexp)
`underscores for functions / types, unclear for modules / constructors`	tokens (what is the standard way for scrunching together multiple words)
`<-`	variable assignment or declaration (assignment)

`let v = e in`	variable assignment or declaration (declaration)

- Functions

`(>) a`	partial application (in the examples below, a normal call is "f(a,b)") (give the first argument to operator ">")
`f a`	partial application (in the examples below, a normal call is "f(a,b)") (give the first argument)
`fun a b -> ...`	anonymous function
`f a b ...`	function call
`f()`	function call (with no parameter)
`let f para1 para2 = ...`	function definition
no syntax needed(1)	function return value (function body is the result)

- Control Flow

`try a with exn -> ...`	exception (catching)
`raise`	exception (throwing)
`if c then ...`	if_then
`if c then b1 else b2`	if_then_else
`for i = 10 downto 1 do ... done`	loop (for each value in a numeric range, 1 decrement)
`for i = 1 to 10 do ... done`	loop (for each value in a numeric range, 1 increment (see also the entries about ranges))
`while c do ... done`	loop (while condition do something)

`match val with`
`  | v1 -> ...`
`  | v2 | v3 -> ...`
`  | _ -> ...`
	multiple selection (switch)

`;`	sequence

- Types

`:`	annotation (or variable declaration)

e :> t	cast (upcast)
type n = t	declaration
constness is the default	mutability, constness (type of a constant value)
T ref	mutability, constness (type of a mutable value)

- Object Oriented & Reflexivity

class	class declaration
inherit	inheritance
object#method para	method invocation
object#method	method invocation (with no parameter)
{< >} or Oo.copy o	object cloning
new class_name ...	object creation

- Package, Module

module P = struct ... end	declare
automatically done based on the file name	declare
module type PType = sig val name1 : type1 ... end module P : PType = struct ... end	declare (selective export)
open p	import (everything into current namespace)
automatically done(2)	import (package (ie. load the package))
.	package scope

- Strings

s.[n]	accessing n-th character
chr	ascii to character
'z'	character "z"
code	character to ascii
char	character type name

`sub`	extract a substring
`rindex`	locate a substring (starting at the end)
`all strings allow multi-line strings`	multi-line
`Marshal.to_string`	serialize (marshalling)
`print_string`	simple print (on strings)
`print_endline`[3]	simple print (on strings)
`printf`	simple print (printf-like)
`sprintf`	sprintf-like
`^`	string concatenation
`= <>`	string equality & inequality
`length`	string size
`"\n"`	strings (end-of-line (without writing the real CR or LF character))
`"..."`	strings (with no interpolation of variables)
`string`	type name
`Marshal.from_string`	unserialize (un-marshalling)
`uppercase / lowercase`	upper / lower case character
`uppercase/lowercase`	uppercase / lowercase / capitalized string

Unknown:

> strings (with interpolation of variables)

- Booleans

`false`	false value
`not`[4]	logical not
`|| / &&`	logical or / and (short circuit)
`true`	true value
`bool`	type name

- Bags and Lists

`split`	2 lists from a list of couples
`::`	adding an element at the beginning (list cons) (return the new list (no side-effect))
`tl`	all but the first element
`fold_left`	f(... f(f(init, e1), e2) ..., en)
`fold_right`	f(e1, f(e2, ... f(en, init) ...))
`find`	find an element
`hd`	first element
`iter`	for each element do something
`mem`	is an element in the list
`exists`	is the predicate true for an element
`for_all`	is the predicate true for every element
`iteri`	iterate with index
`concat`	join a list of strings in a string using a glue string
`find_all`	keep elements (matching)
`filter`	keep elements (matching)
`@`	list concatenation
`[ a ; b ; c ]`	list constructor
`flatten`	list flattening (one level depth)
`combine`	list of couples from 2 lists
`length`	list size
`a.(i)`	list/array indexing
`nth`	list/array indexing
`assoc`	lookup an element in a association list
`partition`	partition a list: elements matching, elements non matching
`rev`	reverse

sort(5)	sort
map	transform a list (or bag) in another one
map2	transform two lists in parallel
a list	type name

Unknown:

split a list

- Various Data Types

find	dictionary (access: read)
add, replace	dictionary (access: write)
mem	dictionary (has the key ?)
remove	dictionary (remove by key)
Hashtbl.t	dictionary (type name)
type typ = N1 | N2 | ...	enumerated type declaration
None	optional value (null value)
option	optional value (type name)
Some v	optional value (value)
.	record (selector)
type typ = { n1 : typ1; n2 : typ2 }	record (type declaration)
:=	reference (pointer) (assigning (when dereferencing doesn't give a lvalue))
ref	reference (pointer) (creation)
!(6)	reference (pointer) (dereference)
a, b, c	tuple constructor
typ1 * ... * typn	tuple type
type typ = N1 of typ1 | N2 of typ2 | ...	union type declaration

Unknown:

optional value (null coalescing)

- Mathematics

`+ +. / - -. / * *. / / /.`(7)	addition / subtraction / multiplication / division
`land / lor / lxor`	bitwise operators (and / or / xor)
`lnot`	bitwise operators (bitwise inversion)
`lsl / lsr or asr`	bitwise operators (left shift / right shift / unsigned right shift)
`**`	exponentiation (power)
`log10`	logarithm (base 10)
`log`	logarithm (base e)
`mod`	modulo (modulo of -3 / 2 is -1)
`- -.`	negation
`1000., 1E3`	numbers syntax (floating point)
`1_000, 10_00, 100_0`	numbers syntax (integer thousand-separator)
`0b1, 0o7, 0xf`	numbers syntax (integers in base 2, octal and hexadecimal)
`1000`	numbers syntax (integers)
`negation first`	operator priorities and associativities (exponentiation vs negation (is -3^2 equal to 9 or -9))
`Random.int`	random (random number)
`Random.init, Random.self_init`	random (seed the pseudo random generator)
`sqrt / exp / abs`	square root / e-exponential / absolute value
`sin / cos / tan`	trigonometry (basic)
`asin / acos / atan`(8)	trigonometry (inverse)
`int_of_float / / floor / ceil`	truncate / round / floor / ceil
`float`	type name (floating point)
`int`	type name (integers)

Remarks

- (1) in Matlab, only for anonymous function
- (2) using a correspondance from the package name to the file name
- (3) adding an end-of-line
- (4) Smalltalk: postfix operator
- (5) in Scheme, not standard, but nearly standard
- (6) prefix
- (7) with mathematical priorities
- (8) Ruby >= 1.7

Pixel

Generated from syntax-across-languages.html.pl
$Id: syntax-across-languages.html.pl 408 2008-08-29 08:32:23Z pixel $