

### Pervasives

<code>val raise : exn -&gt; 'a</code>	Raise the given exception
<code>val (=) : 'a -&gt; 'a -&gt; bool</code>	tests for structural equality of e1 and e2
<code>val (&lt;) : 'a -&gt; 'a -&gt; bool</code>	Negation of (=).
<code>val (==) : 'a -&gt; 'a -&gt; bool</code>	e1 == e2 tests for physical equality of e1 and e2.
<code>val (!=) : 'a -&gt; 'a -&gt; bool</code>	Negation of (==).
<code>val (&lt;) : 'a -&gt; 'a -&gt; bool</code>	
<code>val (&gt;) : 'a -&gt; 'a -&gt; bool</code>	
<code>val (&lt;=) : 'a -&gt; 'a -&gt; bool</code>	
<code>val (&gt;=) : 'a -&gt; 'a -&gt; bool</code>	
<code>val compare : 'a -&gt; 'a -&gt; int</code>	compare x y returns 0 if x is equal to y, a negative integer if x is less than y, and a positive integer if x is greater than y.
<code>val min : 'a -&gt; 'a -&gt; 'a</code>	Return the smaller of the two arguments.
<code>val max : 'a -&gt; 'a -&gt; 'a</code>	Return the greater of the two arguments.
<code>val not : bool -&gt; bool</code>	The boolean negation.
<code>val (&amp;&amp;) : bool -&gt; bool -&gt; bool</code>	
<code>val (  ) : bool -&gt; bool -&gt; bool</code>	
<code>val ( &gt;) : 'a -&gt; ('a -&gt; 'b) -&gt; 'b</code>	Reverse-application operator: $x  > f  > g$ is exactly equivalent to $g (f (x))$ .
<code>val (~-) : int -&gt; int</code>	Unary negation.
<code>val (mod) : int -&gt; int -&gt; int</code>	Integer remainder.
<code>val abs : int -&gt; int</code>	Return the absolute value of the argument.
<code>val (**) : float -&gt; float -&gt; float</code>	Exponentiation.
<code>val sqrt : float -&gt; float</code>	Square root.
<code>val log : float -&gt; float</code>	Natural logarithm.
<code>val log10 : float -&gt; float</code>	Base 10 logarithm.
<code>val ceil : float -&gt; float</code>	Round above to an integer value.
<code>val floor : float -&gt; float</code>	Round below to an integer value.
<code>val abs_float : float -&gt; float</code>	abs_float f returns the absolute value of f.
<code>val float_of_int : int -&gt; float</code>	Convert an integer to floating-point.
<code>val int_of_float : float -&gt; int</code>	
<code>val (^) : string -&gt; string -&gt; string</code>	String concatenation.
<code>val int_of_char : char -&gt; int</code>	
<code>val char_of_int : int -&gt; char</code>	
<code>val fst : 'a * 'b -&gt; 'a</code>	Return the first component of a pair.



By [benjenkinsv95](#)

Published 9th March, 2017.  
Last updated 9th March, 2017.  
Page 1 of 5.

Sponsored by [Readability-Score.com](#)

Measure your website readability!  
<https://readability-score.com>

### Pervasives (cont)

<code>val snd : 'a * 'b -&gt; 'b</code>	Return the second component of a pair.
<code>val (@) : 'a list -&gt; 'a list -&gt; 'a list</code>	List concatenation.

### Types and Type Inference

Function	Type
<code>fun x y -&gt; x + y</code>	<code>int -&gt; int -&gt; int</code>

Include variables that are passed in inside of the type.

### List basics

<code>val length : 'a list -&gt; int</code>	Return the length
<code>val nth : 'a list -&gt; int -&gt; 'a</code>	Return the n-th element of the given list.
<code>val rev : 'a list -&gt; 'a list</code>	List reversal.
<code>val concat : 'a list list -&gt; 'a list</code>	Concatenate a list of lists. The elements of the argument are all concatenated together (in the same order) to give the result.

Prefix with `List`.

### List Iterators

<code>val map : ('a -&gt; 'b) -&gt; 'a list -&gt; 'b list</code>	
<code>val mapi : (int -&gt; 'a -&gt; 'b) -&gt; 'a list -&gt; 'b list</code>	Same as <code>List.map</code> , but the function is applied to the index of the element as first argument (counting from 0), and the element itself as second argument.
<code>val fold_left : ('a -&gt; 'b -&gt; 'a) -&gt; 'a -&gt; 'b list -&gt; 'a</code>	
<code>val fold_right : ('a -&gt; 'b -&gt; 'b) -&gt; 'a list -&gt; 'b -&gt; 'b</code>	
<code>val map2 : ('a -&gt; 'b -&gt; 'c) -&gt; 'a list -&gt; 'b list -&gt; 'c list</code>	
<code>val fold_left2 : ('a -&gt; 'b -&gt; 'c -&gt; 'a) -&gt; 'a -&gt; 'b list -&gt; 'c list -&gt; 'a</code>	
<code>val fold_right2 : ('a -&gt; 'b -&gt; 'c -&gt; 'c) -&gt; 'a list -&gt; 'b list -&gt; 'c -&gt; 'c</code>	

Prefix with `List`.

### List Scanning

<code>val for_all : ('a -&gt; bool) -&gt; 'a list -&gt; bool</code>	checks if all elements of the list satisfy the predicate p
<code>val exists : ('a -&gt; bool) -&gt; 'a list -&gt; bool</code>	checks if at least one element of the list satisfies the predicate p
<code>val for_all2 : ('a -&gt; 'b -&gt; bool) -&gt; 'a list -&gt; 'b list -&gt; bool</code>	
<code>val exists2 : ('a -&gt; 'b -&gt; bool) -&gt; 'a list -&gt; 'b list -&gt; bool</code>	



By [benjenkinsv95](#)

Published 9th March, 2017.  
Last updated 9th March, 2017.  
Page 2 of 5.

Sponsored by [Readability-Score.com](#)

Measure your website readability!

<https://readability-score.com>

### List Scanning (cont)

`val mem : 'a -> 'a list -> bool` mem a l is true if and only if a is equal to an element of l.

`val memq : 'a -> 'a list -> bool` Same as List.mem, but uses physical equality

Prefix with `List.`

### List Searching

`val find : ('a -> bool) -> 'a list -> 'a` returns the first element of the list l that satisfies the predicate p

`val filter : ('a -> bool) -> 'a list -> 'a list` returns all the elements of the list l that satisfy the predicate p

`val partition : ('a -> bool) -> 'a list -> 'a list * 'a list` returns a pair of lists (l1, l2), where l1 is the list of all the elements of l that satisfy the predicate p, and l2 is the list of all the elements of l that do not satisfy p

Prefix with `List.`

### Lists of pairs

`val split : ('a 'b) list -> 'a list * 'b list` Transform a list of pairs into a pair of lists

`val combine : 'a list -> 'b list -> ('a * 'b) list` Transform a pair of lists into a list of pairs

Prefix with `List.`

### List Sorting

`val sort : ('a -> 'a -> int) -> 'a list -> 'a list`

`val merge : ('a -> 'a -> int) -> 'a list -> 'a list -> 'a list` Merge two lists: Assuming that l1 and l2 are sorted according to the comparison function cmp, merge cmp l1 l2 will return a sorted list containing all the elements of l1 and l2.

Prefix with `List.`

### String basics

`val length : string -> int`

`val get : string -> int -> char` returns the character at index n

`val make : int -> char -> string` String.make n c returns a fresh string of length n, filled with the character c.

`val sub : string -> int -> int -> string` returns a fresh string of length len, containing the substring of s that starts at position start and has length len.



By [benjenkinsv95](#)

Published 9th March, 2017.  
Last updated 9th March, 2017.  
Page 3 of 5.

Sponsored by [Readability-Score.com](#)  
Measure your website readability!  
<https://readability-score.com>

[cheatography.com/benjenkinsv95/](https://cheatography.com/benjenkinsv95/)

### String basics (cont)

<code>val concat : string -&gt; string list -&gt; string</code>	<code>String.concat</code> sep sl concatenates the list of strings sl, inserting the separator string sep between each.
<code>val map : (char -&gt; char) -&gt; string -&gt; string</code>	<code>String.map</code> f s applies function f in turn to all the characters of s (in increasing index order) and stores the results in a new string that is returned.
<code>val trim : string -&gt; string</code>	
<code>val index : string -&gt; char -&gt; int</code>	returns the index of the first occurrence of character c in string s
<code>val rindex : string -&gt; char -&gt; int</code>	returns the index of the last occurrence of character c in string s
<code>val contains : string -&gt; char -&gt; bool</code>	
<code>val uppercase_ascii : string -&gt; string</code>	all lowercase letters translated to uppercase
<code>val lowercase_ascii : string -&gt; string</code>	all uppercase letters translated to lowercase
<code>val capitalize_ascii : string -&gt; string</code>	first character set to uppercase
<code>val uncapitalize_ascii : string -&gt; string</code>	first character set to lowercase

Prefix with `String.`

### Map

```
let rec map ( f : 'a -> 'b) (lst : 'a list) : 'b list =
  match lst with
  | [] -> []
  | hd :: tl -> f hd :: map f tl ;;
```

### fold\_left

```
let rec foldleft ( f : 'a -> 'b -> 'a)
  (acc : 'a)
  (lst : 'b list)
  : 'a =
  match lst with
  | [] -> acc
  | hd :: tl -> foldleft f (f acc hd) tl ;;
```

### Different Data Types

list	<code>&lt;type&gt; list</code>	Group together data of the same type
tuple	<code>(&lt;type1&gt; * &lt;type2&gt;)</code>	Group together data of different types
Record	<code>{field one: &lt;type1&gt;; field two: &lt;type2&gt;;...}</code>	Group together data of different types, but it is easier to access data.
Algebraic Data type	<code>Name_one of &lt;type1&gt;   Name_two of &lt;type2&gt;</code>	One piece of data can have many forms



By [benjenkinsv95](#)

Published 9th March, 2017.  
Last updated 9th March, 2017.  
Page 4 of 5.

Sponsored by [Readability-Score.com](#)

Measure your website readability!  
<https://readability-score.com>

### Module

```
module type Integer =
sig
val value : int
end

module ThreeSig : Integer =
struct
let value = 3
let is_positive : bool = true
end
```

Applying the signature Integer, via ": Integer" restricts ThreeSig makes only "value" public.

### Functor

Takes in a module and returns a new module. Ex.

```
module Increment ( I : Integer ) : Integer =
struct
let value = I.value + 1
end
```

Functor can take in multiple modules.

### Abstraction Barrier

If a signature has a type

```
module type OPERATORS =
sig
type t
end ;;
```

Then it can be overridden in the implementation using the "with" keyword

```
module IntOps : OPERATORS with type t = int =
struct
type t = int
let times = ( * )
let divide = ( / )
let plus = ( + )
let int_to_t x = x
end ; ;
```



By [benjenkinsv95](#)

Published 9th March, 2017.  
Last updated 9th March, 2017.  
Page 5 of 5.

Sponsored by [Readability-Score.com](#)  
Measure your website readability!  
<https://readability-score.com>