

Useful Predefined Functions and Operators in Haskell

@ José E. Gallardo

Functions and operators	Description
<code>(!!) :: [a] -> Int -> a</code>	Element at index in list. Index for first element is 0
<code>(&&) :: Bool -> Bool -> Bool</code>	Logical conjunction (and)
<code>(-) :: (Num a) => a -> a -> a</code>	Subtraction
<code>(*) :: (Num a) => a -> a -> a</code>	Multiplication
<code>(**) :: (Floating a) => a -> a -> a</code>	Raising to a real exponent
<code>(/) :: (Fractional a) => a -> a -> a</code>	Real division
<code>(/=) :: (Eq a) => a -> a -> Bool</code>	Inequality
<code>(:) :: a -> [a] -> [a]</code>	Prepends element to a list
<code>(^) :: (Num a, Integral b) => a -> b -> a</code>	Raising to natural exponent
<code>() :: Bool -> Bool -> Bool</code>	Logical disjunction (or)
<code>(+) :: (Num a) => a -> a -> a</code>	Addition
<code>(.) :: (b -> c) -> (a -> b) -> (a -> c)</code>	Function composition
<code>(++) :: [a] -> [a] -> [a]</code>	List concatenation
<code>(<), (<=), (>), (>=) :: (Ord a) => a -> a -> Bool</code>	Order relations
<code>(==) :: (Eq a) => a -> a -> Bool</code>	Equality
<code>abs :: (Num a) => a -> a</code>	Absolute value
<code>all :: (a -> Bool) -> [a] -> Bool</code>	Do all elements in list fulfil a predicate?
<code>and :: [Bool] -> Bool</code>	Logical conjunction (and) over a list of Booleans
<code>any :: (a -> Bool) -> [a] -> Bool</code>	Does any element in list fulfil a predicate?
<code>ceiling :: (RealFrac a, Integral b) => a -> b</code>	Real to Integral conversion (returns least integer not less than argument)
<code>chr :: Int -> Char</code> <code>(import Char)</code>	UNICODE code to character. Inverse to chr
<code>concat :: [[a]] -> [a]</code>	Concatenates a list of lists
<code>const :: a -> b -> a</code>	Constant function (const x is a 1 argument function that always returns x)
<code>cos :: (Floating a) => a -> a</code>	Cosine (uses radians)
<code>curry :: ((a, b) -> c) -> (a -> b -> c)</code>	Turns a function taking a 2-element tuple into a curried one
<code>cycle :: [a] -> [a]</code>	Infinite list repeating forever argument list
<code>div :: (Integral a) => a -> a -> a</code>	Integer division
<code>divMod :: (Integral a) => a -> a -> (a, a)</code>	Integer division and modulo
<code>drop :: Int -> [a] -> [a]</code>	Eliminates first n elements from a list
<code>dropWhile :: (a -> Bool) -> [a] -> [a]</code>	Eliminates longest prefix of elements in a list fulfilling a predicate
<code>elem :: (Eq a) => a -> [a] -> Bool</code>	Is a value in a list?
<code>even :: (Integral a) => a -> Bool</code>	Is an integral value multiple of 2?
<code>exp :: (Floating a) => a -> a</code>	Exponential function (base is e)
<code>filter :: (a -> Bool) -> [a] -> [a]</code>	Retains elements in list fulfilling predicate
<code>flip :: (a -> b -> c) -> (b -> a -> c)</code>	Turns a 2-argument function in another one taking arguments in reverse order
<code>floor :: (RealFrac a, Integral b) => a -> b</code>	Real to integral conversion (returns greatest integer not greater than argument)
<code>foldl :: (b -> a -> b) -> b -> [a] -> b</code>	Reduces elements in a list from left to right using a binary function and an initial value
<code>foldr :: (a -> b -> b) -> b -> [a] -> b</code>	Reduces elements in a list from right to left using a binary function and an initial value
<code>fromIntegral :: (Integral a, Num b) => a -> b</code>	Integral to another numeric type conversion
<code>fst :: (a,b) -> a</code>	First component in a 2-element tuple
<code>gcd :: (Integral a) => a -> a -> a</code>	Greatest common divisor
<code>head :: [a] -> a</code>	First element in a list
<code>id :: a -> a</code>	Identity function (returns its argument)
<code>init :: [a] -> [a]</code>	Eliminates last element in a list
<code>iterate :: (a -> a) -> a -> [a]</code>	Successively iterates a function from an initial value
<code>last :: [a] -> a</code>	Last element in a list
<code>lcm :: (Integral a) => a -> a -> a</code>	Least common multiple

<code>length :: [a] -> Int</code>	Number of elements in a list
<code>log :: (Floating a) => a -> a</code>	Natural logarithm (to the base e)
<code>logBase :: (Floating a) => a -> a -> a</code>	Logarithm to some base (1 st argument)
<code>map :: (a -> b) -> [a] -> [b]</code>	Applies some function to all elements in a list
<code>maximum :: (Ord a) => [a] -> a</code>	Maximum element in a list
<code>min :: (Ord a) => a -> a -> a</code>	Minimum of two values
<code>minimum :: (Ord a) => [a] -> a</code>	Minimum element in a list
<code>mod :: (Integral a) => a -> a -> a</code>	Integer division modulo
<code>not :: Bool -> Bool</code>	Logical negation
<code>null :: [a] -> Bool</code>	Is the list empty?
<code>odd :: (Integral a) => a -> Bool</code>	Is an integral value non-multiple of 2?
<code>or :: [Bool] -> Bool</code>	Logical disjunction (or) over a list of Booleans
<code>ord :: Char -> Int</code> <code>(import Char)</code>	Character to UNICODE CODE. Reverse to ord
<code>pi :: (Floating a) => a</code>	π
<code>read :: (Read a) => String -> a</code>	String to value conversion. Inverse to Show
<code>reverse :: [a] -> [a]</code>	List with same elements in opposite order
<code>repeat :: a -> [a]</code>	Infinite list repeating same value
<code>replicate :: Int -> a -> [a]</code>	List with a value repeated a number of times
<code>round :: (RealFrac a, Integral b) => a -> b</code>	Real to integral rounded conversion
<code>show :: (Show a) => a -> String</code>	Value to String conversion. Inverse to read
<code>sin :: (Floating a) => a -> a</code>	Sine (uses radians)
<code>snd :: (a,b) -> b</code>	Second component in a 2-element tuple
<code>sort :: (Ord a) => [a] -> [a]</code> <code>(import List)</code>	Sorts a list in ascending order
<code>span :: (a -> Bool) -> [a] -> ([a], [a])</code>	Combines results from takeWhile and dropWhile
<code>splitAt :: Int -> [a] -> ([a], [a])</code>	Combines results from take and drop
<code>sqrt :: (Floating a) => a -> a</code>	Square root
<code>tail :: [a] -> [a]</code>	Eliminates first element in list
<code>tan :: (Floating a) => a -> a</code>	Tangent (uses radians)
<code>take :: Int -> [a] -> [a]</code>	First n elements in a list
<code>takeWhile :: (a -> Bool) -> [a] -> [a]</code>	Longest prefix of elements in list fulfilling predicate
<code>truncate :: (RealFrac a, Integral b) => a -> b</code>	returns the integer nearest argument between zero and argument
<code>uncurry :: (a -> b -> c) -> ((a,b) -> c)</code>	Turns a 2-argument curried function into one on tuples
<code>unzip :: [(a, b)] -> ([a], [b])</code>	Transforms a list of pairs into a list of first components and a list of second components. Inverse to zip
<code>unzip3 :: [(a, b, c)] -> ([a], [b], [c])</code>	Takes a list of triples and returns three lists. Inverse to zip3
<code>words :: String -> [String]</code>	Breaks a string into words (delimited by whitespaces)
<code>zip :: [a] -> [b] -> [(a,b)]</code>	Takes two lists and returns a list of corresponding pair
<code>zip3 :: [a] -> [b] -> [c] -> [(a, b, c)]</code>	Takes three lists and returns a list of corresponding triples
<code>zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]</code>	Operates pairwise elements in two lists
<code>zipWith3 :: (a->b->c->d) -> [a] -> [b] -> [c] -> [d]</code>	Operates triplewise elements in three lists

Priority	Associativity	Predefined Operators
10	left	Function application
9	left	!!
9	right	.
8	right	\wedge $\wedge\wedge$ **
7	left	* /
6	left	+ -
5	right	: ++
4	non associative	== /= < <= > >=
3	right	&&
2	right	
1	left	>> >>=
1	right	=<<
0	right	\$ \$!