Arithmetic

(**+** *numbers* …)
number ... → *number*
Returns the sum of the given numbers.

(**−** *number number*)
number number → *number*
Returns the difference of two numbers.

(**−** *number*)
number → *number*
Returns the number times -1.

(***** *number number*), (**/** *number number*)
number number → *number*
Returns the specified product (or quotient) of the specified numbers.

(**quotient** *integer integer*)
number number → *number*
Returns the quotient of the two integers, rounded down to the nearest integer.

(**abs** *number*)
number → *number*
Returns the absolute value of number, i.e. the number with the sign erased.

(**sin** *number*), (**cos** *number*), (**sqrt** *number*)
number → *number*
Returns the sine, cosine, or square root of the number, respectively.

(**max** *numbers* …), (**min** *numbers* …)
number ... → *number*
Returns the maximum/minimum of the *numbers*.

Comparisons

(**string=?** *string1 string2*)
string string → *Boolean*
Returns true if *string1* and *string2* are equivalent.

(**=** *number1 number2*)
number number → *Boolean*
Returns true if numbers are equal.

(**<** *number1 number2*), (**>** *number1 number2*),
(**>=** *number1 number2*), (**<=** *number1 number2*)
number number → *Boolean*
Returns true if *number1* is less than, greater than, greater than or equal to, or less than or equal to, *number2*, respectively.

Other predicates

(**and** *booleans* …), (**or** *booleans* …)
Booleans ... → *Boolean*
Returns true if all/any of the *booleans* are true.

(**not** *boolean*)
Boolean → *Boolean*
Returns true if input is false, or false if input is true.

(**odd?** *number*), (**even?** *number*)
number → *Boolean*
Returns true if number is odd/even, else false.

(**number?** *object*), (**integer?** *object*),
(**string?** *object*), (**list?** *object*)
any → *Boolean*
Returns true if *object* is of that type, otherwise false.

Pictures

empty-image
image
A blank picture.

(**rectangle** *width height mode color*),
(**ellipse** *width height mode color*)
number number string color → *image*
Returns a rectangle or ellipse of the specified *width* and *height* (numbers), *mode* (either "outline" or "solid") and *color*.

(**square** *size mode color*), (**circle** *size mode color*)
number string color → *image*
Returns a square or circle of the specified *size* (numbers), *mode* (either "outline" or "solid") and *color*.

(**overlay** *pictures* …), (**beside** *pictures* …),
(**above** *pictures* …)
image ... → *image*
Returns a picture composed of all the pictures passed as arguments.

(**scale** *magnification pictures* …)
(**rotate** *degrees pictures* …)
number image → *image*
Returns a composite picture of all the specified pictures and scales/rotates it by the specified amounts.

(**iterated-overlay** *function count*), (**iterated-beside** *function count*), (**iterated-above** *function count*)
(*number* → *image*) *number* → *image*
Function should be a function that takes a number as input and returns a picture. Calls function *count* times with arguments starting at 0 and going to *count*-1. Collects all the pictures together and returns one picture that is the composite of all the pictures.

Lists

(**list** *elements* …)
X ... → $(listof\ X)$
Returns a list with all the specified *elements*, in order.

(**append** *lists* …)
$(listof\ X)$ ... → $(listof\ X)$
Returns one long list containing all the elements of all the *lists*, in order. Thus (append (list 1 2) (list 3 4)) returns the list (1 2 3 4).

(**list-ref** *list position*)
$(listof\ X)\ number \rightarrow X$
Returns the element of *list* at the specified *position* (0=first element 1=second, etc.).

(**first** *list*), (**second** *list*), etc.
$(listof\ X) \rightarrow X$
Returns the first (or second, etc.) element of the *list*. Thus (first (list 1 2 3)) returns 1. If *list* is the empty list, it throws an exception.

(**cons** *element list*)
$X\ \ (listof\ X) \rightarrow (listof\ X)$
Returns a new list starting with *element*, and followed by all the elements of *list*, in order. Thus (cons 1 (list 0 0)) returns the list: (list 1 0 0).

(**rest** *list*)
$(listof\ X) \rightarrow (listof\ X)$
Returns a list containing all but the first element of *list*. Thus (rest (list 1 2 3)) returns the list: (list 2 3). If *list* is the empty list, it throws an exception.

(**empty?** *list*)
$list \rightarrow boolean$
Returns true if *list* has no elements, otherwise returns false.

(**length** *list*)
$list \rightarrow number$
Returns the number of items in *list*.

(**map** *function list*)
$(In \rightarrow Out)\ \ (listof\ In) \rightarrow (listof\ Out)$
Calls function on each element of list, and returns all the results as a list. In other words, (map *func* (list 1 2 3)) behaves like (list (*func* 1) (*func* 2) (*func* 3)).

(**for-each** *function list*)
$(In \rightarrow Out)\ \ (listof\ In) \rightarrow void$
Calls function on each element of list, returns nothing.

(**filter** *function list*)
$(X \rightarrow boolean)\ \ (listof\ X) \rightarrow (listof\ X)$
Returns a new list consisting of only those elements of the original *list* for which *function* returns true. If *function* returns a value other than true or false, it will produce an exception.

(**foldl** *function start list*), (**foldr** *function start list*)
$(X\ X \rightarrow X)\ \ X\ (listof\ X) \rightarrow X$
Applies *function* pairwise to all the elements of *list*. So folding + over a list of numbers starting at 0 will return the sum of all the numbers. If *list* is empty, fold will just return *start*. foldl processes the list elements left-to-right, and foldr processes them right-to-left.

(**apply** *function list*)
$function\ \ list \rightarrow any$
Calls *function* with all the elements of *list* (in order) as arguments to the function. In other words, (apply + (list 1 2 3)) behaves like (+ 1 2 3).

(**andmap** *predicate list*), (**ormap** *predicate list*)
$(X \rightarrow boolean)\ \ (listof\ X) \rightarrow boolean$
Calls *predicate* (a function) on successive elements of *list*. Ormap returns true if *predicate* returns true for at least one element of *list*, otherwise it returns false. Andmap only returns true if *predicate* returns true for every element of *list*. If *predicate* returns a value other than true or false, it will produce an exception.

(**member** *item list*)
$X\ (listof\ X) \rightarrow Boolean$
True if and only if *item* is contained in *list* else false.

(**remove-all** *item list*)
$X\ (listof\ X) \rightarrow (listof\ X)$
Returns a copy of *list* with every occurrence of *item* removed.

Strings

(**string-append** *strings* …)
$string$ ... → $string$
Returns a new string containing all the text from *strings*.

(**string-length** *string*)
$string \rightarrow number$
Returns the number of characters in the input.

(**printf** *string things-to-print…*)
$string\ any$ ... → $void$
Displays *things-to-print* according to *string* template.

Colors

(**color** *red green blue*)
$number\ number\ number \rightarrow color$
Returns a color with the specified amounts of red, green, and blue light.