Glossary

The following gives the procedures used in class along with their descriptions and their type signatures, that is, the types of their inputs and outputs.

Arithmetic

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

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

(**−** *number*)
number → number
Returns the numbe 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 integer, 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, respecitively.

(**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 number*), etc.
number number → Boolean
Returns true if *number1* is less than, greater than, 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 argument is false, or false if argument is true.

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

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

Pictures

All the following procedures return pictures. Rectangle, ellipses, etc. are particular kinds of pictures.

empty-image
$image$
A blank picture.

(**rectangle** *width height mode color*),
(**ellipse** *width height mode color*)
$number\ number\ string\ color \rightarrow image$
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*)
$number\ string\ color \rightarrow image$
Returns a square or circle of the specified *size* (numbers), *mode* (either "outline" or "solid") and *color*.

(**overlay** *pictures* …)
(**beside** *pictures* …)
(**above** *pictures* …)
$image\ ... \rightarrow image$
Returns a picture comprised of all the pictures passed as arguments.

(**scale** *magnification pictures* …)
$number\ image \rightarrow image$
Returns a composite picture of all the specified pictures and scales (grows) it by the specified magnification factor.

(**iterated-overlay** *procedure count*)
(**iterated-beside** *procedure count*)
(**iterated-above** *procedure count*)
$(number \rightarrow image)\ number \rightarrow image$
Procedure should be a procedure that takes a number as input and returns a picture. Calls procedure *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* …)
$X\ ... \rightarrow (listof\ X)$
Returns a list with all the specified *elements*, in order.

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

(**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: (1 0 0).

(**list-ref** *list position*)
$(listof\ X)\ number \rightarrow (listof\ X)$
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 '(1 2 3)) returns 1. If *list* is the empty list, it throws an exception.

(**rest** *list*)
$(listof\ X) \rightarrow (listof\ X)$
Returns a list containing all but the first element of *list*. Thus (rest '(1 2 3)) returns the 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** *procedure list*)
$(In \rightarrow Out)\ \ (listof\ In) \rightarrow (listof\ Out)$

Calls procedure on each element of list, and returns all the results as a list. So in other words, (map *proc* (list 1 2 3)) behaves like (list (*proc* 1) (*proc* 2) (*proc* 3)).

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

(**foldl** *procedure start list*)
(**foldr** *procedure start list*)
$(X\ X \rightarrow X)\ X\ (listof\ X) \rightarrow X$
Applies *procedure* pairwise to all the elements of *list*. So folding + over a list of numbers 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** *procedure list*)
$procedure\ \ list \rightarrow any$
Calls *procedure* once, passing it as arguments all the elements of *list* (in order). 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 procedure) 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.

(**assoc** *item list-of-lists*)
$X\ \ (listof\ (listof\ X)) \rightarrow (listof\ X)$
Returns the first sublist of *list-of-lists* that begins with *item*, or false if no sublists have *item* as their first element.

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

Strings

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

Colors

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