Function	Parameters	Description	Examples
if	any number of pairs boolean and value (condition, value for true), and last value (when all conditions false)	evaluates condition; if true, then evaluates following expression; if false then skips expression, and evaluates following condition; if all conditions fail then evaluates last expression	(if (yes) +1.0 +2.0) = +1.0, (if (no) +1.0 +2.0) = +2.0, (if (no) +1.0 (yes) +2.0 (no) +3.0 +4.0) = +2.0, (if (no) +1.0 (no) +2.0 (yes) +3.0 +4.0) = +3.0, (if (no) +1.0 (no) +2.0 (no) +3.0 +4.0) = +4.0, (if +12.0) = +12.0
as	any number of pairs name and value, and expression to compute	computes values and assigns them to names; then computes expression which can use these names as precomputed values	(as a +2.0 (mul (a) (mul (a) (a)))) = +8.0, (as a +2.0 b +3.0 (min (mul (a) (b)) (div (a) (b)))) = +0.66666666666666667, (as +4.0) = +4.0
do	any types, any number, at least one	value of last parameter	(do +1.0 +2.0) = +2.0
df	one name	tell if function is defined	(df add) = (yes), (df foobar) = (no)
is	name and value of any type	tell if value has a type described by name	(is numb +2.0) = (yes), (is numb "alfa beta") = (no)
inc	one number	number plus one	(inc +2.0) = 3.0, (inc -2.0) = -1.0
dec	one number	number minus one	(dec +2.0) = +1.0, (dec -2.0) = -3.0
add	two numbers	sum of these numbers	(add +12.0 -3.0) = +9.0
sub	two numbers	difference between numbers	(sub +19.0 +4.0) = +15.0, (sub +19.0 -2.0) = +21.0
mul	two numbers	multiplication of these numbers	(mul +2.0 +3.0) = +6.0
div	two numbers, second not zero	division of numbers	(div +3.0 +2.0) = +1.5
rnd	one number	number rounded to integer	(rnd +2.1) = +2.0, (rnd -2.9) = -3.0
flo	one number	biggest int not bigger than number	(flo +2.9) = +2.0, (flo -2.1) = -3.0
cil	one number	smallest integer not bigger than number	(cil +2.1) = +3.0, (cil -2.2) = -1.0
min	none or two numbers	mininimum representable positive number for no parameters or smaller number out of two	(min) = +0.00000000000000001, (min +1.0 -2.9) = -2.9
max	none or two numbers	maximum representable number or bigger number out of two	(max) = +99999999999999999.9, (max +1.0 -2.0) = +1.0
ops	one number	number with opposite sign	(ops -2.0) = +2.0, (ops 0.0) = 0.0
abs	one number	absolute value of number	(abs +1.0) = +1.0, (abs -2.9) = +2.9
zer	none or one number	0.0 for no parameters or if number is zero	(zer) = 0.0, (zer +2.0) = (no), (zer 0.0) = (yes)
pos	one number	is number positive?	(pos +2.5) = (yes), (pos 0.0) = (no), (pos -2.0) = (no)
neg	one number	is number negative?	(neg +2.0) = (no), (neg 0.0) = (no), (neg -2.0) = (yes)
equ	two numbers	are numbers equal?	(equ +2.0 +1.0) = (no), (equ +2.0 +2.0) = (yes)
gre	two numbers	is first number greater than the second one?	(gre +2.0 +1.0) = (yes), (gre +1.0 +2.0) = (no)
les	two numbers	is first number less than the second one?	(les +2.0 +1.0) = (no), (les +1.0 +2.0) = (yes)
yes	none	boolean value "true", "yes"	(yes) = (yes)
no	none	boolean value "false", "no"	(no) = (no)
not	one boolean	negation of boolean value	(not (yes)) = (no), (not (no)) = (yes)

and	two booleans	logical "and" of these	(and (yes) (yes)) = (yes), (and (no) (yes)) = (no)
or	two booleans	logical "or" or these	(or (yes) (no)) = (yes), (or (no) (no)) = (no)
numb	one text	get number from text, parse number from text	(numb "0.0") = 0.0, (numb "-2.0") = -2.0
name	one text or one node of tree	convert text to name or get name from node of tree	(name "alfa") = alfa, (name (root (ins (new) alfa +3.0))) = alfa
text	one number or one text	text representation of number or name	(text +2.34) = "+2.34", (text alfa) = "alfa"
leng	one name or one text	length of name or text, number of characters	(leng alfa) = +4.0, (leng "alfa beta") = +7.0
same	two names or two texts	does these names or text look the same?	(same alfa alfa) = (yes), (same "alfa" "beta") = (no)
bfor	two names or two texts	does the first one sort before the second one?	(bfor alfa beta) = (yes), (bfor alfa aaaa) = (no)
aftr	two names or two texts	does the first one sort after the second one	(aftr alfa aaaa) = (yes), (aftr alfa beta) = (no)
escp	one text	escape special chars in text	(escp "a\"b\\c") = "a\\\"b\\\\c"
unes	one text	unescape special chars in text	(unes "a\\\"b\\\\c") = "a\"b\\c"
uppr	one name or one text	convert letters to uppercase	(uppr Alfa) = ALFA
lowr	one name or one text	convert letters to lowercase	(lowr "ALFA beta") = "alfa beta"
head	one text or one name, and one number	beginning of text or name	(head alfa +2.0) = al, (head "alfa beta" +6.0) = "alfa b"
tail	one text or one name, and one number	get ending of text or name	(tail alfa +2.0) = fa, (tail "alfa beta" +6.0) = "a beta"
join	two names or two texts	join names with underscore, or concatenate texts	(join alfa beta) = alfa_beta, (join "alfa" "beta") = "alfabeta"
part	two names or two texts	does first one have a part as the second one	(part alfa lf) = (yes), (part alfa aa) = (no)
indx	two names or two texts	get index of part	(indx alfa fa) = +2.0, (index alfa a) = 0.0
ptrn	one text	is text valid pattern (regular expression)?	(ptrn "ab*c") = (yes), (ptrn "ab*[") = (no)
subs	one text	number of subespressions in pattern	(subs "a(lf)a") = +1.0, (subs "a(l(f(a)))") = +3.0
mtch	one name or text, one text, optional number	matches regular expression against text or name; number tells what subexpression to look at (if not present looks at whole pattern)	(mtch alfa "a.*a") = (yes), (mtch alfa "a(lf|xy)a" +1.0) = (yes)
msta	same as mtch	starting index of matched text or name	(msta alfa "a.*a") = 0.0, (msta alfa "a(lf|xy)a" +1.0) = +1.0
mend	same as mtch	ending index of matched text or name	(mend alfa "a.*a") = +4.0, (mend alfa "a(lf|xy)a" +1.0) = +3.0
new	none	new, empty tree (object)	(new) = (tree ...)
ref	two trees or two nodes of tree	does both reference the same place in memory?	(ref (new) (new)) = (no), (as t (new) (ref (t) (t))) = (yes)
any	one tree	are there any nodes in tree?	(any (new)) = (no), (any (ins (new) alfa +1.0)) = (yes)
qty	one tree	number of nodes in tree	(qty (new)) = 0.0, (qty (ins (new) alfa +1.0)) = +1.0

def	one tree and one name	is node of given name defined in tree?	(def (new) alfa) = (no), (def (ins (new) alfa +1.0) alfa) = (yes)
ins	one tree, one name, and one of any type	inserts into tree new node with given name and value; returns tree; error if name already defined	(ins (ins (new) alfa +1.0) beta +2.0) = (tree alfa +1.0 beta +2.0)
del	one tree, one name	deletes from tree node of given name; error if node not defined	(del (ins (ins (new) alfa +1.0) beta +2.0) alfa) = (tree beta +2.0)
set	one tree, one name, one value of any type; or one node and one value of any type	in tree set node of given name to given value; error if node of that name not defined; or set value of given node	(set (ins (new) alfa +2.0) alfa -99.0) = (tree alfa -99.0), (as t (ins (new) alfa +2.0) (do (set (root (t)) +99.0) (t))) = (tree alfa +99.0)
get	one tree and one name; or one node of tree	get value from named node in given tree; or get value of node	(as t (ins (new) alfa +2.0) (get (t) alfa)) = +2.0, (as t (ins (new) alfa +2.0) (get (root (t))))
root	one tree	root node of tree (top of AVL tree)	(as t (new) (do (ins (t) a +1.0) (ins (t) b +2.0) (ins (t) c +3.0) (root (t)))) = (node b +2.0)
frst	one tree	frst (leftmost) node of AVL tree	(as t (new) (do (ins (t) a +1.0) (ins (t) b +2.0) (ins (t) c +3.0) (frst (t)))) = (node a +1.0)
last	one tree	last (rightmost) node of AVL tree	(as t (new) (do (ins (t) a +1.0) (ins (t) b +2.0) (ins (t) c +3.0) (last (t)))) = (node c +3.0)
left	one node of tree	left child of node or (void) if not defined	(as t (new) (do (ins (t) a +1.0) (ins (t) b +2.0) (ins (t) c +3.0) (left (root (t))))) = (node a +1.0)
rght	one node of tree	right child of node or (void) if not defined	(as t (new) (do (ins (t) a +1.0) (ins (t) b +2.0) (ins (t) c +3.0) (rght (root (t))))) = (node c +3.0)
next	one node of tree	next node in tree after given node or (void) if current node is last one	(as t (new) (do (ins (t) a +1.0) (ins (t) b +2.0) (ins (t) c +3.0) (next (root (t))))) = (node c +3.0)
prev	one node of tree	previous node in tree before given node or (void) if current node is first one	(as t (new) (do (ins (t) a +1.0) (ins (t) b +2.0) (ins (t) c +3.0) (prev (root (t))))) = (node a +1.0)
vlue	one node of tree	value of given node	(as t (new) (do (ins (t) a +1.0) (ins (t) b +2.0) (ins (t) c +3.0) (vlue (root (t))))) = +2.0
tree	one node of tree	tree the node belongs to	(as t (new) (do (ins (t) a +1.0) (ins (t) b +2.0) (ins (t) c +3.0) (tree (root (t))))) = (tree a +1.0 b +2.0 c +3.0)
main	none	a tree that is defined in treep at the beginning of computation; one can use this tree to store information throughout all computation; this is to hold data that must be accessible all the time	(do (ins (main) a +1.0) (ins (main) b +2.0) (ins (main) c +3.0) (main)) = (tree a +1.0 b +2.0 c +3.0)
read	none	read one piece of data from stdin; data can be number, name, text, left or right parenthesis, space, tabulator, new line char, or void on the end of file	(read)+2.0 = +2.0, (read) +1.0 = (spac), (do (read) (read)) "beta" = "beta"
back	one value of any type that read returns	cancels last "read" call; treep can remember only one such value	(as r (read) (do (read) (back (r)) (read)))+1.0+2.0 = +1.0

wrte	one value of any type that read returns	writes data to stdout; outputs nothing for (void)	(wrte "Hello World!") = "Hello World!"
dump	one text	writes to stdout contents of text (without quotes, unescaped)	(dump "Hello World!") = Hello World!
time	none	number of seconds since epoch	(time) = +1318432558.0
repr	one number	ISO representation of time given as number of seconds	(repr (time)) = "2011-10-12 17:17:43"
pars	one text	parse text as ISO date format and return number of seconds	(pars "1999-01-01 12:34:56") = +915190496.0
rand	none	random number in range from 0.0 to +1.0	(rand) = +0.32514742745934555
iden	one name	uniqe identifier starting with given name	(iden My_Id) = My_Id_20111012_172117_957613416
vers	none	version of treep	(vers) = "2011-10-12"
void	none	value of type "void"	(void) = (void)
spac	none	value of type "spac"; this is space	(spac) = (spac)
tabu	none	vlue of type "tabu", this is tabulator	(tabu) = (tabu)
line	none	vlue of type "line", this is new line character	(line) = (line)
lpar	none	this is value of type "lpar", this is left parenthesis	(lpar) = (lpar), (wrte (lpar)) = (
rpar	none	value of type "rapr", this is right parenthesis	(rpar) = (rpar), (wrte (rpar)) =)
info	any number of values of any types	write information about values on stderr; texts are output without quotes and unescaped	(info +2.4 alfa "a\"b\\c") = +2.4alfaa"b\c
exit	one boolean	exits computation with success for (yes) and failure for (no)	(exit (no)) = failure, (exit (yes)) = success