# CheckerLang 2.0.9 function reference

Inhalt

[abs 5](#_Toc60992867)

[acos 5](#_Toc60992868)

[add 5](#_Toc60992869)

[all 5](#_Toc60992870)

[any 6](#_Toc60992871)

[append 6](#_Toc60992872)

[asin 6](#_Toc60992873)

[atan 6](#_Toc60992874)

[body 6](#_Toc60992875)

[boolean 7](#_Toc60992876)

[ceiling 7](#_Toc60992877)

[checkerlang\_version 7](#_Toc60992878)

[close 7](#_Toc60992879)

[compare 7](#_Toc60992880)

[const 7](#_Toc60992881)

[cos 8](#_Toc60992882)

[date 8](#_Toc60992883)

[date\_day 8](#_Toc60992884)

[date\_hour 8](#_Toc60992885)

[date\_minute 9](#_Toc60992886)

[date\_month 9](#_Toc60992887)

[date\_second 9](#_Toc60992888)

[date\_year 9](#_Toc60992889)

[decimal 9](#_Toc60992890)

[div 10](#_Toc60992891)

[div0 10](#_Toc60992892)

[E 10](#_Toc60992893)

[equals 10](#_Toc60992894)

[esc 10](#_Toc60992895)

[escape\_pattern 11](#_Toc60992896)

[eval 11](#_Toc60992897)

[exp 11](#_Toc60992898)

[file\_input 11](#_Toc60992899)

[file\_output 11](#_Toc60992900)

[find 12](#_Toc60992901)

[first 12](#_Toc60992902)

[floor 12](#_Toc60992903)

[format\_date 12](#_Toc60992904)

[greater 13](#_Toc60992905)

[greater\_equals 13](#_Toc60992906)

[grep 13](#_Toc60992907)

[identity 13](#_Toc60992908)

[if\_empty 13](#_Toc60992909)

[if\_null 14](#_Toc60992910)

[if\_null\_or\_empty 14](#_Toc60992911)

[info 14](#_Toc60992912)

[int 14](#_Toc60992913)

[interval 14](#_Toc60992914)

[is\_alphanumerical 15](#_Toc60992915)

[is\_boolean 15](#_Toc60992916)

[is\_decimal 15](#_Toc60992917)

[is\_empty 15](#_Toc60992918)

[is\_func 15](#_Toc60992919)

[is\_int 16](#_Toc60992920)

[is\_list 16](#_Toc60992921)

[is\_map 16](#_Toc60992922)

[is\_negative 16](#_Toc60992923)

[is\_not\_empty 16](#_Toc60992924)

[is\_not\_null 17](#_Toc60992925)

[is\_null 17](#_Toc60992926)

[is\_numeric 17](#_Toc60992927)

[is\_numerical 17](#_Toc60992928)

[is\_positive 18](#_Toc60992929)

[is\_set 18](#_Toc60992930)

[is\_string 18](#_Toc60992931)

[is\_valid\_date 18](#_Toc60992932)

[is\_valid\_time 18](#_Toc60992933)

[is\_zero 19](#_Toc60992934)

[join 19](#_Toc60992935)

[label\_data 19](#_Toc60992936)

[length 19](#_Toc60992937)

[less 19](#_Toc60992938)

[less\_equals 20](#_Toc60992939)

[list 20](#_Toc60992940)

[log 20](#_Toc60992941)

[log10 20](#_Toc60992942)

[log2 20](#_Toc60992943)

[lower 21](#_Toc60992944)

[ls 21](#_Toc60992945)

[map 21](#_Toc60992946)

[map\_get 21](#_Toc60992947)

[map\_get\_pattern 21](#_Toc60992948)

[map\_list 22](#_Toc60992949)

[max 22](#_Toc60992950)

[MAXDECIMAL 22](#_Toc60992951)

[MAXINT 22](#_Toc60992952)

[min 22](#_Toc60992953)

[MINDECIMAL 23](#_Toc60992954)

[MININT 23](#_Toc60992955)

[mod 23](#_Toc60992956)

[mul 23](#_Toc60992957)

[non\_empty 23](#_Toc60992958)

[non\_zero 24](#_Toc60992959)

[not\_equals 24](#_Toc60992960)

[now 24](#_Toc60992961)

[pairs 24](#_Toc60992962)

[parse 24](#_Toc60992963)

[parse\_date 25](#_Toc60992964)

[pattern 25](#_Toc60992965)

[PI 25](#_Toc60992966)

[pow 25](#_Toc60992967)

[print 26](#_Toc60992968)

[println 26](#_Toc60992969)

[prod 26](#_Toc60992970)

[put 26](#_Toc60992971)

[q 26](#_Toc60992972)

[random 27](#_Toc60992973)

[range 27](#_Toc60992974)

[read 27](#_Toc60992975)

[read\_all 27](#_Toc60992976)

[readln 28](#_Toc60992977)

[reduce 28](#_Toc60992978)

[replace 28](#_Toc60992979)

[rest 28](#_Toc60992980)

[reverse 28](#_Toc60992981)

[reverse\_list 29](#_Toc60992982)

[reverse\_string 29](#_Toc60992983)

[round 29](#_Toc60992984)

[run 29](#_Toc60992985)

[s 29](#_Toc60992986)

[set 30](#_Toc60992987)

[set\_seed 30](#_Toc60992988)

[sign 30](#_Toc60992989)

[sin 30](#_Toc60992990)

[sorted 30](#_Toc60992991)

[split 31](#_Toc60992992)

[split2 31](#_Toc60992993)

[sqrt 31](#_Toc60992994)

[str\_contains 31](#_Toc60992995)

[str\_ends\_with 32](#_Toc60992996)

[str\_find 32](#_Toc60992997)

[str\_input 32](#_Toc60992998)

[str\_matches 32](#_Toc60992999)

[str\_starts\_with 32](#_Toc60993000)

[str\_trim 33](#_Toc60993001)

[string 33](#_Toc60993002)

[sub 33](#_Toc60993003)

[sublist 33](#_Toc60993004)

[substitute 33](#_Toc60993005)

[substr 34](#_Toc60993006)

[sum 34](#_Toc60993007)

[tan 34](#_Toc60993008)

[timestamp 35](#_Toc60993009)

[type 35](#_Toc60993010)

[upper 35](#_Toc60993011)

[zip 35](#_Toc60993012)

[zip\_map 35](#_Toc60993013)

## abs

abs(n)

Returns the absolute value of n.

Examples:

abs(2) => 2

abs(-3) => 3

## acos

acos(x)

Returns the arcus cosinus of x.

Examples:

acos(1) => 0

## add

add(a, b)

Returns the sum of a and b. For numerical values this uses usual arithmetic. For lists and strings it concatenates. For sets it uses union.

Examples:

add(1, 2) => 3

## all

all(lst, predicate) all(lst)

Returns TRUE, if the predicate function returns TRUE for all elements of the list.

If no predicate function is passed, the list must contain boolean values.

Examples:

all([1, 2, 3], fn(n) n <= 3) => TRUE

all([1, 2, 3], fn(n) n < 3) => FALSE

all([TRUE, TRUE, TRUE]) => TRUE

all([TRUE, FALSE, TRUE]) => FALSE

all([e >= 2 for e in [2, 3, 4]]) => TRUE

all([e >= 2 for e in [1, 3, 4]]) => FALSE

## any

any(lst, predicate) any(lst)

Returns TRUE, if the predicate function returns TRUE for any element of the list.

If no predicate function is passed, the list must contain boolean values.

Examples:

any([1, 2, 3], fn(n) n == 3) => TRUE

any([1, 2, 3], fn(n) n == 4) => FALSE

any([TRUE, TRUE, TRUE]) => TRUE

any([TRUE, FALSE, TRUE]) => TRUE

any([e >= 2 for e in [2, 3, 4]]) => TRUE

any([e >= 2 for e in [1, 3, 4]]) => TRUE

## append

append(lst, element)

Appends the element to the list lst. The lst may also be a set. Returns the changed list.

Examples:

append([1, 2], 3) => [1, 2, 3]

append(set([1, 2]), 3) => set([1, 2, 3])

## asin

asin(x)

Returns the arcus sinus of x.

Examples:

asin(0) => 0

## atan

atan(x)

Returns the arcus tangens of x.

Examples:

atan(0) => 0

## body

body(f)

Returns the body of the lambda f.

Examples:

body(fn(x) 2 \* x) => '(mul 2, x)'

## boolean

boolean(obj)

Converts the obj to a boolean, if possible.

Examples:

boolean(1) => TRUE

## ceiling

ceiling(x)

Returns the integral decimal value that is equal to or next higher than x.

Examples:

ceiling(1.3) => 2

## checkerlang\_version

The checkerlang version

## close

close(conn)

Closes the input or output connection and releases system resources.

## compare

compare(a, b)

Returns -1 if a is less than b, 0 if a is equal to b, and 1 if a is greater than b.

Examples:

compare(1, 2) => -1

## const

const(val)

Returns a function that returns a constant value, regardless of the argument used.

Examples:

def f = const(2); f(1) => 2

def f = const(2); f('x') => 2

## cos

cos(x)

Returns the cosinus of x.

Examples:

cos(PI) => -1

## date

date(obj)

Converts the obj to a date, if possible. If obj is a string, the format YYYYmmdd is assumed. If this fails, the fallback YYYYmmddHH is tried.

See parse\_date for handling other formats.

Examples:

string(date('20170102')) => '20170102'

## date\_day

date\_day(value)

Extracts the day part from the given date value and returns it as an integer. The value will be converted to a date value using the date function.

Examples:

date\_day('20190102') => 02

## date\_hour

date\_hour(value)

Extracts the hour part from the given date value and returns it as an integer. The value will be converted to a date value using the date function.

Examples:

date\_hour('2019010212') => 12

## date\_minute

date\_minute(value)

Extracts the hour part from the given date value and returns it as an integer.

Examples:

date\_minute(parse\_date('201901021223', fmt='yyyyMMddHHmm')) => 23

## date\_month

date\_month(value)

Extracts the month part from the given date value and returns it as an integer. The value will be converted to a date value using the date function.

Examples:

date\_month('20190102') => 01

## date\_second

date\_second(value)

Extracts the second part from the given date value and returns it as an integer.

Examples:

date\_second(parse\_date('20190102122345', fmt='yyyyMMddHHmmss')) => 45

## date\_year

date\_year(value)

Extracts the year part from the given date value and returns it as an integer. The value will be converted to a date value using the date function.

Examples:

date\_year('20190102') => 2019

## decimal

decimal(obj)

Converts the obj to a decimal, if possible.

Examples:

decimal('1.2') => 1.2

## div

div(a, b)

Returns the value of a divided by b. If both values are ints, then the result is also an int. Otherwise, it is a decimal.

Examples:

div(6, 2) => 3

## div0

div0(a, b, div\_0\_value = MAXINT)

If b is not zero, the result of a / b is returned. If b is zero, the value div\_0\_value is returned.

Examples:

div0(12, 3) => 4

div0(12, 5) => 2

div0(12.0, 5) => 2.40

div0(12.5, 2) => 6.25

div0(12, 0) => MAXINT

div0(12, 0, 0) => 0

div0(12, 0.0, 0) => 0

## E

The mathematical constant e.

## equals

equals(a, b)

Returns TRUE if a is equals to b.

Integer values are propagated to decimal values, if required.

Examples:

equals(1, 2) => FALSE

equals(1, 1) => TRUE

equals(1, 1.0) => TRUE

equals('a', 'b') => FALSE

## esc

esc(str)

Escapes the characters <, > and & by their HTML entities.

Examples:

esc('a<b') => 'a&lt;b'

esc('<code>') => '&lt;code&gt;'

## escape\_pattern

escape\_pattern(s)

Escapes special characters in the string s, so that the result can be used in pattern matching to match the literal string.

Currently, the | and . characters are escaped.

Examples:

escape\_pattern('|') => '\|'

escape\_pattern('|.|') => '\|\.\|'

## eval

eval(s)

Evaluates the string or node s.

Examples:

eval('1+1') => 2

## exp

exp(x)

Returns the power e ^ x.

Examples:

exp(0) => 1

## file\_input

file\_input(filename, encoding = 'UTF-8')

Returns an input object, that reads the characters from the given file.

## file\_output

file\_output(filename, encoding = 'UTF-8', append = FALSE)

Returns an output object, that writes to the given file. If the file exists it is overwritten, unless append is TRUE.

## find

find(obj, value, key = identity, start = 0)

Returns the index of the value in the list or string obj. The search process normally starts at index 0, but this can be changed using the parameter start. If the value is not found, -1 is returned. The optional parameter key is a function used to extract the comparison value from the list elements.

Examples:

find([1, 2, 3, 4], 3) => 2

find(['abc', 'def'], 'e', key = fn(x) x[1]) => 1

find('abc|def|ghi', '|') => 3

find('abc|def|ghi', '|', start = 4) => 7

## first

first(lst)

Returns the first element of a list.

Examples:

first([1, 2, 3]) => 1

first(NULL) => NULL

## floor

floor(x)

Returns the integral decimal value that is equal to or next lower than x.

Examples:

floor(1.3) => 1

## format\_date

format\_date(date, fmt = 'yyyy-MM-dd HH:mm:ss')

Formats the date value according to fmt and returns a string value.

Examples:

format\_date(date('20170102')) => '2017-01-02 00:00:00'

format\_date(NULL) => NULL

format\_date(date('2017010212'), fmt = 'HH') => '12'

## greater

greater(a, b)

Returns TRUE if a is greater than b.

Examples:

greater(1, 2) => FALSE

greater(1, 1) => FALSE

greater(2, 1) => TRUE

## greater\_equals

greater\_equals(a, b)

Returns TRUE if a is greater than or equals to b.

Examples:

greater\_equals(1, 2) => FALSE

greater\_equals(1, 1) => TRUE

greater\_equals(2, 1) => TRUE

## grep

grep(lst, pat)

Returns the sublist of list lst, that contains only entries, that match the regular expression pattern pat.

If pat does not contain ^ and $, then the prefix ^.\* and the postfix .\*$ are added to the pattern, so that the pattern matches partially.

Examples:

grep(['one', 'two', 'three'], //e//) => ['one', 'three']

grep(['one', 'two', 'three'], //^one$//) => ['one']

## identity

identity(a)

Identity function. It returns the input value as return value.

Examples:

identity(1) => 1

## if\_empty

if\_empty(a, b)

Returns b if a is an empty string otherwise returns a.

Examples:

if\_empty(1, 2) => 1

if\_empty('', 2) => 2

## if\_null

if\_null(a, b)

Returns b if a is NULL otherwise returns a.

Examples:

if\_null(1, 2) => 1

if\_null(NULL, 2) => 2

## if\_null\_or\_empty

if\_null\_or\_empty(a, b)

Returns b if a is null or an empty string otherwise returns a.

Examples:

if\_null\_or\_empty(1, 2) => 1

if\_null\_or\_empty(NULL, 2) => 2

if\_null\_or\_empty('', 2) => 2

## info

info(obj)

Returns the info associated with an object.

## int

int(obj)

Converts the obj to an int, if possible.

Examples:

int('1') => 1

## interval

interval(a, b)

Returns the interval of integers between a and b, inclusive.

Examples:

interval(1, 10) => [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

interval(1, 1) => [1]

## is\_alphanumerical

is\_alphanumerical(str, min = 0, max = 99999)

Returns TRUE if the string is alphanumerical, i.e. contains only a-z, A-Z and 0-9. It is possible to specify minimal and maximal length using the min and max optional parameters.

Examples:

is\_alphanumerical('Ab12') => TRUE

## is\_boolean

is\_boolean(obj)

Returns TRUE if the object is of type boolean.

Examples:

is\_boolean(1 == 2) => TRUE

## is\_decimal

is\_decimal(obj)

Returns TRUE if the object is of type decimal.

Examples:

is\_decimal(123.45) => TRUE

## is\_empty

is\_empty(obj)

Returns TRUE, if the obj is empty. Lists, sets and maps are empty, if they do not contain elements. Strings are empty, if the contain no characters. NULL is always empty.

Examples:

is\_empty([]) => TRUE

is\_empty(set([1, 2])) => FALSE

is\_empty('') => TRUE

## is\_func

is\_func(obj)

Returns TRUE if the object is of type func.

Examples:

is\_func(fn(x) 2 \* x) => TRUE

is\_func(sum) => TRUE

## is\_int

is\_int(obj)

Returns TRUE if the object is of type int.

Examples:

is\_int(123) => TRUE

## is\_list

is\_list(obj)

Returns TRUE if the object is of type list.

Examples:

is\_list([1, 2, 3]) => TRUE

## is\_map

is\_map(obj)

Returns TRUE if the object is of type map.

Examples:

is\_map(map([['a', 1], ['b', 2]])) => TRUE

## is\_negative

is\_negative(obj)

Returns TRUE if the obj is negative.

Examples:

is\_negative(-1) => TRUE

## is\_not\_empty

is\_not\_empty(obj)

Returns TRUE, if the obj is not empty. Lists, sets and maps are empty, if they do not contain elements. Strings are empty, if the contain no characters. NULL is always empty.

Examples:

is\_not\_empty([]) => FALSE

is\_not\_empty(set([1, 2])) => TRUE

is\_not\_empty('a') => TRUE

## is\_not\_null

is\_not\_null(obj)

Returns TRUE, if the obj is not NULL.

Examples:

is\_not\_null('') => TRUE

is\_not\_null(1) => TRUE

is\_not\_null(NULL) => FALSE

## is\_null

is\_null(obj)

Returns TRUE, if the obj is NULL.

Examples:

is\_null('') => FALSE

is\_null(1) => FALSE

is\_null(NULL) => TRUE

## is\_numeric

is\_numeric(obj)

Returns TRUE if the object is of type numeric. Numeric types are int and decimal.

Examples:

is\_numeric(123) => TRUE

is\_numeric(123.45) => TRUE

## is\_numerical

is\_numerical(str, min = 0, max = 99999)

Returns TRUE if the string is numerical, i.e. contains only 0-9. It is possible to specify minimal and maximal length using the min and max optional parameters.

Examples:

is\_numerical('123') => TRUE

## is\_positive

is\_positive(obj)

Returns TRUE if the obj is positive.

Examples:

is\_positive(1) => TRUE

## is\_set

is\_set(obj)

Returns TRUE if the object is of type set.

Examples:

is\_set(set([1, 2, 3])) => TRUE

## is\_string

is\_string(obj)

Returns TRUE if the object is of type string.

Examples:

is\_string('abc') => TRUE

## is\_valid\_date

is\_valid\_date(str, fmt='yyyyMMdd')

Returns TRUE if the string represents a valid date. The default format is yyyyMMdd. It is possible to specify different formats using the fmt optional parameter.

Examples:

is\_valid\_date('20170304') => TRUE

## is\_valid\_time

is\_valid\_time(str, fmt='HHmm')

Returns TRUE if the string represents a valid time. The default format is HHmm. It is possible to specify different formats using the fmt optional parameter.

Examples:

is\_valid\_time('1245') => TRUE

## is\_zero

is\_zero(obj)

Returns TRUE if the obj is zero.

Examples:

is\_zero(0) => TRUE

## join

join(sep, lst)

Returns a string containing all elements of the list lst separated by the string sep.

Examples:

join('|', [1, 2, 3]) => '1|2|3'

join('--', ['one', 'world']) => 'one--world'

join('|', []) => ''

join('|', [1]) => '1'

## label\_data

label\_data(labels, data)

Creates a map that labels the data with the given labels.

Labels and data must be two lists of equal length. Labels must be unique.

Examples:

label\_data(['a', 'b', 'c'], [1, 2, 3]) => <map 'a': 1, 'b': 2, 'c': 3>

## length

length(obj)

Returns the length of obj. This only works for strings, lists, sets and maps.

Examples:

length('123') => 3

length([1, 2, 3]) => 3

## less

less(a, b)

Returns TRUE if a is less than b.

Examples:

less(1, 2) => TRUE

## less\_equals

less\_equals(a, b)

Returns TRUE if a is less than or equals to b.

Examples:

less\_equals(1, 2) => TRUE

less\_equals(2, 1) => FALSE

less\_equals(1, 1) => TRUE

## list

list(obj)

Converts the obj to a list.

Examples:

list(1) => [1]

## log

log(x)

Returns the natural logarithm of x.

Examples:

round(log(E), digits = 8) => 1

## log10

log10(x)

Returns the logarithm of x to base 10.

Examples:

log10(1000) => 3

## log2

log2(x)

Returns the logarithm of x to base 2.

Examples:

log2(1024) => 10

## lower

lower(str)

Converts str to lower case letters.

Examples:

lower('Hello') => 'hello'

## ls

ls()

Returns a list of all defines symbols (functions and constants).

## map

map(obj)

Converts the obj to a map, if possible.

Examples:

map([[1, 2], [3, 4]]) => <map 1: 2, 3: 4>

## map\_get

map\_get(m, k, default\_value=NULL)

If the map m contains the key k, then the corresponding value is returned. Otherwise, the default\_value is returned.

Examples:

map\_get(map([['a', 1], ['b', 2]]), 'a') => 1

map\_get(map([['a', 1], ['b', 2]]), 'b') => 2

map\_get(map([['a', 1], ['b', 2]]), 'c') => NULL

map\_get(map([['a', 1], ['b', 2]]), 'c', default\_value = 9) => 9

## map\_get\_pattern

map\_get\_pattern(m, k, default\_value=NULL)

The map m is assumed to contain regex patterns as keys. If the key k matches one of the regex patterns, then the corresponding value is returned. Otherwise, the default\_value is returned.

If more than one pattern matches the key k, then it is undefined, which pattern is selected for retrieving its value.

Examples:

map\_get\_pattern(map([[//[ab]//, 1], [//[cd]//, 2]]), 'a') => 1

map\_get\_pattern(map([[//[ab]//, 1], [//[cd]//, 2]]), 'b') => 1

map\_get\_pattern(map([[//[ab]//, 1], [//[cd]//, 2]]), 'c') => 2

map\_get\_pattern(map([[//[ab]//, 1], [//[cd]//, 2]]), 'd') => 2

map\_get\_pattern(map([[//[ab]//, 1], [//[cd]//, 2]]), 'e') => NULL

## map\_list

map\_list(func, lst)

Returns a list where each element is the corresponding element of lst with func applied. Thus, the elements of the list are mapped using the function func to new values.

Examples:

map\_list(fn(x) 2 \* x, [1, 2, 3]) => [2, 4, 6]

map\_list(fn(x) '\*' + x + '\*', ['one', 'two', 'three']) => ['\*one\*', '\*two\*', '\*three\*']

## max

max(a, b, key = identity)

Returns the maximum of the values a, b. The optional key parameter takes a function with one parameter, which is used to get the value from a and b that is used for the comparison. Default key is the identity function.

Examples:

max(1, 2) => 2

max([1, 'z'], [2, 'a'], key = fn(x) x[1]) => [1, 'z']

## MAXDECIMAL

The maximal decimal value

## MAXINT

The maximal int value

## min

min(a, b, key = identity)

Returns the minimum of the values a, b. The optional key parameter takes a function with one parameter, which is used to get the value from a and b that is used for the comparison. Default key is the identity function.

Examples:

min(1, 2) => 1

min([1, 'z'], [2, 'a'], key = fn(x) x[1]) => [2, 'a']

## MINDECIMAL

The minimal decimal value

## MININT

The minimal int value

## mod

mod(a, b)

Returns the modulus of a modulo b.

Examples:

mod(7, 2) => 1

## mul

mul(a, b)

Returns the product of a and b. For numerical values this uses the usual arithmetic. If a is a string and b is an int, then the string a is repeated b times. If a is a list and b is an int, then the list is repeated b times.

Examples:

mul(2, 3) => 6

mul('2', 3) => '222'

mul([1, 2], 3) => [1, 2, 1, 2, 1, 2]

## non\_empty

non\_empty(a, b)

Returns the value a, if a is a non-empty string, otherwise returns b.

Examples:

non\_empty('a', 'b') => 'a'

non\_empty('', 'b') => 'b'

## non\_zero

non\_zero(a, b)

Returns the value a, if a is a non-zero integer, otherwise returns b.

Examples:

non\_zero(1, 2) => 1

non\_zero(0, 2) => 2

## not\_equals

not\_equals(a, b)

Returns TRUE if a is not equals to b.

Integer values are propagated to decimal values, if required.

Examples:

not\_equals(1, 2) => TRUE

not\_equals(1, 1) => FALSE

not\_equals(1, 1.0) => FALSE

not\_equals('a', 'b') => TRUE

## now

now()

Returns the current date and time.

## pairs

pairs(lst)

Returns a list where each entry consists of a pair of elements of lst.

Examples:

pairs(NULL) => NULL

pairs([]) => []

pairs([1]) => []

pairs([1, 2]) => [[1, 2]]

pairs([1, 2, 3]) => [[1, 2], [2, 3]]

pairs([1, 2, 3, 4]) => [[1, 2], [2, 3], [3, 4]]

## parse

parse(s)

Parses the string s.

Examples:

parse('2+3') => '(add 2, 3)'

## parse\_date

parse\_date(str, fmt = 'yyyyMMdd')

Parses the string str according to fmt and returns a datetime value. If the format does not match or if the date is invalid, the NULL value is returned.

It is possible to pass a list of formats to the fmt parameter. The function sequentially tries to convert the str and if it works, returns the value.

Examples:

parse\_date('20170102') => '20170102'

parse\_date('20170102', fmt = 'yyyyMMdd') => '20170102'

parse\_date('20170102', fmt = ['yyyyMMdd']) => '20170102'

parse\_date('201701022015', fmt = ['yyyyMMdd', 'yyyyMMddHH', 'yyyyMMddHHmm']) => '20170102'

parse\_date('20170112', fmt = ['yyyy', 'yyyyMM']) => NULL

parse\_date('20170144') => NULL

## pattern

pattern(obj)

Converts the obj to a regexp pattern, if possible.

Examples:

pattern('xy[1-9]{3}') => //xy[1-9]{3}//

## PI

The mathematical constant pi.

## pow

pow(x, y)

Returns the power x ^ y.

Examples:

pow(2, 3) => 8

pow(2.5, 2) => 6.25

pow(4, 2) => 16

pow(4.0, 2.0) => 16.0

round(pow(2, 1.5), digits = 3) => 2.828

## print

print(obj, out = stdout)

Prints the obj to the output out. Default output is stdout which may be connected to the console (e.g. in case of REPL) or a file or be silently ignored.

Examples:

print('hello') => NULL

## println

println(obj, out = stdout)

Prints the obj to the output out and terminates the line. Default output is stdout which may be connected to the console (e.g. in case of REPL) or a file or be silently ignored.

Examples:

println('hello') => NULL

## prod

prod(list)

Returns the product of a list of numbers.

Examples:

prod([1, 2, 3]) => 6

prod(range(1, 10)) => 362880

## put

put(m, key, value)

Puts the value into the map m at the given key.

Examples:

def m = map([[1, 2], [3, 4]]); put(m, 1, 9) => <map 1: 9, 3: 4>

## q

q(lst)

Returns a string containing all elements of the list lst separated by a pipe character.

Examples:

q([1, 2, 3]) => '1|2|3'

q([]) => ''

## random

random() random(a) random(a, b)

Returns a random number. If no argument is provided, a decimal value in the range [0, 1) is returned. If only a is provided, then an int value in the range [0, a) is returned. If both a and b are provided, then an int value in the range [a, b) is returned.

Examples:

set\_seed(1); random(5) => 1

## range

range(a) range(a, b) range(a, b, step)

Returns a list containing int values in the range. If only a is provided, the range is [0, a). If both a and b are provided, the range is [a, b). If step is given, then only every step element is included in the list.

Examples:

range(4) => [0, 1, 2, 3]

range(3, 6) => [3, 4, 5]

range(10, step = 3) => [0, 3, 6, 9]

range(10, 0, step = -2) => [10, 8, 6, 4, 2]

## read

read(input = stdin)

Read a character from the input. If end of input is reached, an empty string is returned.

Examples:

def s = str\_input('hello'); read(s) => 'h'

## read\_all

read\_all(input = stdin)

Read the whole input. If end of input is reached, an empty string is returned.

Examples:

def s = str\_input('hello'); read\_all(s) => 'hello'

## readln

readln(input = stdin)

Read one line from the input. If end of input is reached, an empty string is returned.

Examples:

def s = str\_input('hello'); readln(s) => 'hello'

## reduce

reduce(list, func)

Reduces a list by successively applying the binary function func to partial results and list elements.

Examples:

reduce([1, 2, 3, 4], add) => 10

## replace

replace(s, a, b, start = 0)

Replaces all occurences of a in the string s with b. The optional parameter start specifies the start index.

Examples:

replace('abc', 'b', 'x') => 'axc'

replace('abcbcbca', 'b', 'x') => 'axcxcxca'

replace('abc', 'b', 'xy') => 'axyc'

replace('abcdef', 'bcd', 'xy') => 'axyef'

replace('abcabcabc', 'abc', 'xy', start = 3) => 'abcxyxy'

## rest

rest(lst)

Returns the rest of a list, i.e. everything but the first element.

Examples:

rest([1, 2, 3]) => [2, 3]

rest([1]) => []

rest([]) => []

rest(NULL) => NULL

## reverse

reverse(obj)

Returns a reversed copy of a string or a list.

Examples:

reverse([1, 2, 3]) => [3, 2, 1]

reverse('abc') => 'cba'

## reverse\_list

reverse\_list(list)

Returns a reversed copy of a list.

Examples:

reverse\_list([1, 2, 3]) => [3, 2, 1]

## reverse\_string

reverse\_string(str)

Returns a reversed copy of a string.

Examples:

reverse\_string('abc') => 'cba'

## round

round(x, digits = 0)

Returns the decimal value x rounded to the specified number of digits. Default for digits is 0.

Examples:

round(1.345, digits = 1) => 1.3

## run

run(file)

Loads and interprets the file.

## s

s(str, start = 0)

Returns a string, where all placeholders are replaced with their appropriate values. Placeholder have the form '{var}' and result in the value of the variable var inserted at this location.

Examples:

def name = 'damian'; s('hello {name}') => 'hello damian'

def foo = '{bar}'; def bar = 'baz'; s('{foo}{bar}') => '{bar}baz'

s('{PI} is cool') => '3.14159265358979 is cool'

## set

set(obj)

Converts the obj to a set, if possible.

Examples:

set([1, 2, 3]) => <set 1, 2, 3>

## set\_seed

set\_seed(n)

Sets the seed of the random number generator to n.

Examples:

set\_seed(1) => 1

## sign

sign(n)

Returns the signum of n

Examples:

sign(2) => 1

sign(-3) => -1

## sin

sin(x)

Returns the sinus of x.

Examples:

sin(0) => 0

## sorted

sorted(lst, cmp=compare, key=identity)

Returns a sorted copy of the list. This is sorted according to the value returned by the key function for each element of the list. The values are compared using the compare function cmp.

Examples:

sorted([3, 2, 1]) => [1, 2, 3]

sorted([6, 2, 5, 3, 1, 4]) => [1, 2, 3, 4, 5, 6]

## split

split(str, delim = '[ \t]+')

Splits the string str into parts and returns a list of strings. The delim is a regular expression. Default is spaces or tabs.

Examples:

split('a,b,c', //,//) => ['a', 'b', 'c']

split('', '-') => []

## split2

split2(str, sep1, sep2)

Performs a two-stage split of the string data. This results in a list of list of strings.

Examples:

split2('a:b:c|d:e:f', escape\_pattern('|'), escape\_pattern(':')) => [['a', 'b', 'c'], ['d', 'e', 'f']]

split2('', '\|', ':') => []

## sqrt

sqrt(x)

Returns the square root of num as a decimal value.

Examples:

sqrt(4) => 2

## str\_contains

str\_contains(str, part)

Returns TRUE if the string str contains part.

Examples:

str\_contains('abcdef', 'cde') => TRUE

## str\_ends\_with

str\_ends\_with(str, part)

Returns TRUE if the string str ends with part.

Examples:

str\_ends\_with('abcdef', 'def') => TRUE

## str\_find

str\_find(str, part, start = 0)

Returns the index of the first occurence of part in the string str. If part is not contained in str, then -1 is returned. Start specifies the search start index. It defaults to 0.

Examples:

str\_find('abcdefg', 'cde') => 2

str\_find('abc|def|ghi', '|', start = 4) => 7

str\_find('abcxyabc', 'abc', start = 5) => 5

## str\_input

str\_input(str)

Returns an input object, that reads the charactes of the given string str.

Examples:

str\_input('abc') => <!input-stream>

## str\_matches

str\_matches(str, pattern)

Returns TRUE, if str matches the regular expression pattern.

Examples:

str\_matches('abc12', //[a-c]+[1-9]+//) => TRUE

## str\_starts\_with

str\_starts\_with(str, part)

Returns TRUE if the string str starts with part.

Examples:

str\_starts\_with('abcdef', 'abc') => TRUE

## str\_trim

str\_trim(str)

Trims leading and trailing whitespace.

Examples:

str\_trim(' abc 12 ') => 'abc 12'

## string

string(obj)

Converts the obj to a string, if possible.

Examples:

string(123) => '123'

## sub

sub(a, b)

Returns the subtraction of b from a. For numerical values this uses usual arithmetic. For lists and sets, this returns lists and sets minus the element b. If a is a datetime value and b is datetime value, then the date difference is returned. If a is a datetime value and b is a numeric value, then b is interpreted as number of days and the corresponding datetime after subtracting these number of days is returned.

Examples:

sub(1, 2) => -1

sub([1, 2, 3], 2) => [1, 3]

sub(date('20170405'), date('20170402')) => 3

sub(date('20170405'), 3.5) => '20170401'

sub(set([3, 1, 2]), 2) => <set 1, 3>

## sublist

sublist(lst, startidx) sublist(lst, startidx, endidx)

Returns the sublist starting with startidx. If endidx is provided, this marks the end of the sublist. Endidx is not included.

Examples:

sublist([1, 2, 3, 4], 2) => [3, 4]

## substitute

substitute(obj, idx, value)

If obj is a list or string, returns a list or string with the element at index idx replaced by value.

Examples:

substitute('abcd', 2, 'x') => 'abxd'

substitute([1, 2, 3, 4], 2, 'x') => [1, 2, 'x', 4]

## substr

substr(str, startidx) substr(str, startidx, endidx)

Returns the substring starting with startidx. If endidx is provided, this marks the end of the substring. Endidx is not included.

Examples:

substr('abcd', 2) => 'cd'

## sum

sum(list, ignore = [])

Returns the sum of a list of numbers. Values contained in the optional list ignore are counted as 0.

Examples:

sum([1, 2, 3]) => 6

sum([1, 2.5, 3]) => 6.5

sum([1, 2.5, 1.5, 3]) => 8.0

sum([1.0, 2.0, 3.0]) => 6.0

sum([1.0, 2, -3.0]) => 0.0

sum([1, 2, -3]) => 0

sum([1, '1', 1], ignore = ['1']) => 2

sum(range(101)) => 5050

sum([]) => 0

sum([NULL], ignore = [NULL]) => 0

sum([1, NULL, 3], ignore = [NULL]) => 4

sum([1, NULL, '', 3], ignore = [NULL, '']) => 4

## tan

tan(x)

Returns the tangens of x.

Examples:

tan(0) => 0

## timestamp

timestamp(x)

Returns current system timestamp.

## type

type(obj)

Returns the name of the type of obj as a string.

Examples:

type('Hello') => 'string'

## upper

upper(str)

Converts str to upper case letters.

Examples:

upper('Hello') => 'HELLO'

## zip

zip(a, b)

Returns a list where each element is a list of two items. The first of the two items is taken from the first list, the second from the second list. The resulting list has the same length as the shorter of the two input lists.

Examples:

zip([1, 2, 3], [4, 5, 6, 7]) => [[1, 4], [2, 5], [3, 6]]

## zip\_map

zip\_map(a, b)

Returns a map where the key of each entry is taken from a, and where the value of each entry is taken from b, where a and b are lists of identical length.

Examples:

zip\_map(['a', 'b', 'c'], [1, 2, 3]) => <map 'a': 1, 'b': 2, 'c': 3>