defn

let

=

is

if

fn

def

str

deftest

map

Bind names in a local scope.

Define a function at the top level.

Assert that an expression returns truthy inside of a test.

Return true if all arguments are equal, false otherwise.

Create a function.

Two-way conditional branch.

Convert the arguments to strings and concatenate them.

Define a top level Var.

Create a seq whose elements are the result of applying a function to the elements of another seq.

Define a test.

ns

->

defn-

first

when

testing

or

apply

assoc

count

Thread-first macro.

Define a namespace at the top of a file.

Return the initial element of a seq.

Define a function at the top level that is private to the namespace.

Add context text to enclosed assertions.

One-way conditional branch.

Call a function on a seq of arguments.

Evaluate expressions in turn, returning the first that is truthy or nil otherwise.

Return the number of elements in the given collection.

Add one or more key/value pairs to an associative data structure such as a hashmap or vector.

and

not

nil?

defmethod

+

defmacro

get

*

cond

recur

Return true if the argument is falsey, false otherwise.

Evaluate expressions in turn, returning the first that is not truthy, or the last one otherwise.

Define a method in a multimethod.

Return true if the argument is nil, false otherwise.

Define a new macro.

Add all arguments numerically.

Multiply all arguments numerically.

Return the value associated with the key in an associative data structure.

Explicit tail recursion.

Multi-way conditional branch.

println	do
seq	doseq
->>	throw
atom	reduce
-	name

Execute all expressions and return the value of the last expression.

Print out all arguments and output a new line.

Iterate over a seq, executing the body expressions for each element.

Convert a value to a seq or nil if it's empty.

Raise an exception.

Thread last macro.

Starting with an initial value, apply a function to that value and successive elements of a seq.

Create a Atom.

Return the non-namespace part of a keyword or symbol as a string, or the argument itself if it's a string.

Subtract arguments numerically, or negate a number.

instance?

conj

if-let

inc

swap!

into

range

filter

get-in

merge

Add an element to a collection.

Check if a value is an instance of a class or interface.

Add one to a number.

Two-way conditional branch with binding of the test result.

Add elements of a seq to a collection.

Replace an Atom's value with the result of applying a function to the current value.

Keep elements of a seq that are truthy for a given predicate.

Create a seq of numbers from start to end.

Add key/value pairs from one map into another.

Get the value from a nested associative data structure at a path.

empty?

loop

for

try

list

format

catch

when-not

rest

<

Create a scope for tail recursion.

Returns true if there are no elements in a collection, false otherwise.

Create a point for catching exception.

List comprehension.

Create a string using java.lang.String.format.

Create a list.

Negative one-way conditional branch.

Declare what to do when a particular class of exception is caught.

Mathematical less than.

Return a seq of elements from a collection excluding the first.

vec

partial

concat

reset!

set

when-let

/

int

cons

nth

Take a function and arguments and make a new function that has those arguments already applied.

Convert a collection to a vector.

Set the value of an Atom without regard to the current value.

String seqs together in order.

One-way conditional branch with binding of the test result.

Convert a collection to a set.

Convert a number to an integer.

Numeric division.

Retrieve an element from a seq by numeric index.

Add an element to the beginning of a seq.

assert

defproject

second

are

contains?

update-in

>

doto

defprotocol

.

Define a Leiningen project.

Throw an AssertionError if expression is false.

Assert that multiple expressions are truthy in a test.

Return the second element of a seq.

Modify a value at a path in a nested data structure by applying a function to that value.

Does a key appear in an associative data structure?

Execute forms on first argument, then return it.

Mathematical greater than.

Execute a Java method.

Define a new protocol.

meta

==

keys

next

map?

string?

set!

aget

keyword

if-not

Numeric equality comparison (type-independent).

Return the metadata on a value.

Return a seq of elements from a collection excluding the first, or nil if it's empty.

Return a seq of keys from a hash map.

Return true if the argument is a string, false otherwise.

Return true if the argument is a map, false otherwise.

Return the value of an array at an index.

Set thread-local vars, Java object instance fields, and Java class static fields.

Two-way negative conditional branch.

Create a keyword from a string or a namespace and name.

symbol

binding

dec

dissoc

defrecord

comp

not=

thrown?

float

select

Create thread-local bindings for dynamic vars.

Create a symbol from a string or a namespace and name.

Remove a key/value from an associative data structure.

Subtract one from a number.

Compose two or more functions.

Create a new record type.

A special form in clojure.test/is expressions to check if an exception is thrown.

Return true if the arguments are not equal, false otherwise.

Return a new set keeping only elements that are truthy for a given predicate.

Convert a number to a Java float.