Clojure API Documentation

1.4

huangz1990

11 08, 2017

Contents

Clojure API clojure.github.com/clojure

github.com/huangz1990/clojure_api_cn

CHAPTER 1

clojure.core

1.1 ->

(-> x)
(-> x form)
(-> x form & more)

`x form x (item) form form form [ (-> a-map :key) (:key a-map) :key a-map ]`

```
;

user=> (-> "a b c d" .toUpperCase (.replace "A" "X") (.split " ") first)
"X"

user=> (use '[clojure.walk :only [macroexpand-all]])
nil

user=> (macroexpand-all '(-> "a b c d"
                             .toUpperCase
                             (.replace "A" "X")
                             (.split " ")
                             first))
(first (. (. (. "a b c d" toUpperCase) replace "A" "X") split " "))

;  map

user=> (def language {:clojure {:author {:first-name "Rich" :last-name "Hickey"}}})
#'user/language

user=> (:first-name (:author (:clojure language)))
"Rich"
```

```
user=> (-> language :clojure :author :first-name)
"Rich"
```

1.2 ->>

(->> x form)
(->> x form & more)

x form x (item) form form form [(->> a-map :key) (:key a-map) :key a-map]

```
;  10

user=> (->> (range)
             (map #(* % %))
             (filter even?)
             (take 10)
             (reduce +))
1140

user=> (use '[clojure.walk :only [macroexpand-all]])
nil

user=> (macroexpand-all '(->> (range)
                              (map #(* % %))
                              (filter even?)
                              (take 10)
                              (reduce +)))
(reduce + (take 10 (filter even? (map (fn* [p1__3#] (* p1__3# p1__3#)) (range)))))
```

1.3 ->

(-> x)
(-> x form)
(-> x form & more)

x form x (item) form form form [(-> a-map :key) (:key a-map) :key a-map]

```
;

user=> (-> "a b c d" .toUpperCase (.replace "A" "X") (.split " ") first)
"X"

user=> (use '[clojure.walk :only [macroexpand-all]])
nil
```

```
user=> (macroexpand-all '(-> "a b c d"
                             .toUpperCase
                             (.replace "A" "X")
                             (.split " ")
                             first))
(first (. (. (. "a b c d" toUpperCase) replace "A" "X") split " "))

;  map

user=> (def language {:clojure {:author {:first-name "Rich" :last-name "Hickey"}}})
#'user/language

user=> (:first-name (:author (:clojure language)))
"Rich"

user=> (-> language :clojure :author :first-name)
"Rich"
```

1.4 ->>

(->> x form)
(->> x form & more)

`x form x` (item) `form form form` [`(->> a-map :key) (:key a-map) :key a-map`]

```
;  10

user=> (->> (range)
            (map #(* % %))
            (filter even?)
            (take 10)
            (reduce +))
1140

user=> (use '[clojure.walk :only [macroexpand-all]])
nil

user=> (macroexpand-all '(->> (range)
                              (map #(* % %))
                              (filter even?)
                              (take 10)
                              (reduce +)))
(reduce + (take 10 (filter even? (map (fn* [p1__3#] (* p1__3# p1__3#)) (range)))))
```

1.5 aclone

(aclone array)

Java

```
user=> (def a (int-array [1 2 3 4]))
#'user/a

user=> (def b (aclone a))
#'user/b

user=> (aset b 0 23)
23

user=> (vec b)
[23 2 3 4]

user=> (vec a)
[1 2 3 4]
```

1.6 alength

(alength array)

Java,

```
(def array (into-array Integer/TYPE [1 2 3 4 5]))
#'user/array

(alength array)
5
```

1.7 alter-var-root

(alter-var-root v f & args)

var `v v args f`

```
user=> (def v 10)
#'user/v

user=> (alter-var-root (var v) + 1)      ; (var v)  #'v
11

user=> v
11
```

1.8 assoc

(assoc map key val)
(assoc map key val & kvs)

assoc associate

assoc Map key-val Map Map Map Map assoc key-val

assoc index key val

<= (count vector)

```
user=> (assoc {} :Clojure "Rich")
{:Clojure "Rich"}

user=> (assoc {:Clojure "Rich"} :Clojure "Rich Hickey")      ;  key
{:Clojure "Rich Hickey"}

user=> (assoc [1 2 3] 0 10086)
[10086 2 3]

user=> (assoc [1 2 3] 3 10086)           ; key  (count vector)
[1 2 3 10086]

user=> (assoc [1 2 3] 10086 10086)       ; key  (count vector)
IndexOutOfBoundsException   clojure.lang.PersistentVector.assocN
(PersistentVector.java:136)
```

1.9 bigdec

(bigdec x)

x BigDecimal

```
user=> (bigdec 3.0)
3.0M

user=> (bigdec 5)
5M

user=> (bigdec -1)
-1M

user=> (bigdec -1.0)
-1.0M
```

1.10 bigint

(bigint x)

x BigInt

```
user=> (def x (biginteger 19931029))
#'user/x

user=> (class x)
java.math.BigInteger
```

1.11 biginteger

(biginteger x)

`x BigInteger`

```
user=> (def x (biginteger 19931029))
#'user/x

user=> (class x)
java.math.BigInteger
```

1.12 comment

(comment & body)

`body nil`

```
user=> (comment hello-clojure)
nil

user=> (comment "clojure!")
nil

user=> (defn msg []
         (comment "nothing but a greeting message here")
         (println "hello"))
#'user/msg

user=> (msg)
hello
nil
```

1.13 comp

(comp)
(comp f)
(comp f g)
(comp f g h)
(comp f g h & other-functions)

`comp`

variable number of args

```
user=> ((comp str double +) 3 3 3)  ;
"9.0"
```

```
user=> (str (double (+ 3 3 3)))
"9.0"
```

1.14 compile

(compile lib)

(symbol) `lib` class `lib` (classpath-relative)class `*compile-path*` (classpath)

1.15 complement

(complement f)

`f`

`f` `f`

```
user=> (defn f []
           (println "hello")
           false)
#'user/f

user=> (f)
hello
false

user=> ((complement f))
hello
true
```

1.16 concat

(concat)
(concat x)
(concat x y)
(concat x y & zs)

collection

```
;  collection

user=> (concat)
()

user=> (concat [1])
(1)

user=> (concat [1] [2])
```

```
(1 2)

user=> (concat [1] [2] [3])
(1 2 3)

user=> (concat [1] [2] [3] [4 5 6])
(1 2 3 4 5 6)


;  concat  collection
;  collection  cons  conj

; user=> (concat 1 [2 3])
; IllegalArgumentException Don't know how to create ISeq from: java.lang.Long
↪clojure.lang.RT.seqFrom (RT.java:487)
```

1.17 cond

(cond & clauses)

test/expression test test true cond test expression test

(cond) nil

```
user=> (defn type-of-number [n]
           (cond (> n 0) "positive number"
                 (< n 0) "negative number"
                 :else "zero"))
#'user/type-of-number

user=> (type-of-number 10)
"positive number"

user=> (type-of-number -5)
"negative number"

user=> (type-of-number 0)
"zero"
```

1.18 conj

(conj coll x)
(conj coll x & xs)

conj conjoin collection

coll coll x coll

coll nil (conj nil item) (item)

```
; coll  nil

user=> (conj nil 1)
```

```
(1)


;

user=> (conj [0 1 2] 3)
[0 1 2 3]


;

user=> (conj (list 0 1 2) 3)
(3 0 1 2)


;  conj
;

user=> (conj [0 1 2] 3 4 5)
[0 1 2 3 4 5]

user=> (conj (list 0 1 2) 3 4 5)
(5 4 3 0 1 2)
```

1.19 cons

(cons x seq)

x seq

```
; cons  1

user=> (cons 1 '())
(1)


; cons  1  (2 3)

user=> (cons 1 (list 2 3))
(1 2 3)
```

1.20 constantly

(constantly x)

x

```
user=> (def ten (constantly 10))
#'user/ten

user=> (ten)
10

user=> (ten 1)
```

```
10

user=> (ten 1 2)
10

user=> (ten 1 2 3)
10
```

1.21 contains?

(contains? coll key)

`key coll true false`

index collection Java `contains? key` (range)

`contains?`

`coll` *some*

```
user=> (contains? {:clojure "Rich"} :python)        ;  Map
false

user=> (contains? {:clojure "Rich"} :clojure)
true

user=> (contains? [1 3 5 7 9] 3)                    ;
true

user=> (contains? [1 3 5 7 9] 10086)
false
```

1.22 count

(count coll)

`coll`

`(count nil) 0`

`coll` Java Collection Map

```
user=> (count nil)
0

user=> (count [1 2 3 4])
4

user=> (count (list 1 2 3 4))
4

user=> (count "string")
6
```

```
user=> (count {:clojure "Rich" :python "Guido" :ruby :Matz})
3
```

1.23 counted?

(counted? coll)

`coll count true`

```
; Map  count
;

user=> (counted? [1 2 3])
true

user=> (counted? '(1 2 3))
true

user=> (counted? {:clojure "Rich"})
true

user=> (counted? #{:a :b :c})
true

user=> (counted? "string")
false
```

1.24 declare

(declare & names)

var forward declarations

```
user=> (defn f []
           (g))
;CompilerException java.lang.RuntimeException: Unable to resolve symbol: g in this␣
↪context, compiling:(NO_SOURCE_PATH:2)

user=> (declare g)
#'user/g

user=> (defn f []
            (g))
#'user/f
```

1.25 defn-

(defn- name & decls)

`defn`

```
user=> (defn- msg [] "hello moto")
#'user/msg

user=> (msg)
"hello moto"

user=> (meta #'msg)
{:arglists ([]), :ns #<Namespace user>, :name msg, :private true, :line 1, :file "NO_
↪SOURCE_PATH"}
```

1.26 defonce

(defonce name expr)

`name` root value `expr` `name` root value

`name` root value `expr`

```
user=> number                          ;  root value
;CompilerException java.lang.RuntimeException: Unable to resolve symbol: number in␣
↪this context, compiling:(NO_SOURCE_PATH:0)

user=> (defonce number 10086)          ;  root value
#'user/number

user=> number
10086

user=> (defonce number 123123)         ;  root value
nil

user=> number
10086
```

1.27 defprotocol

(defprotocol name & opts+sigs)

```
(defprotocol IOFactory
  (make-reader [this] "Create a Buffered Reader")
  (make-writer [this] "Create a Buffered Writer")
)
```

1.28 delay

(delay & body)

`body` Delay

`Delay` `force` `deref` `@` `body`

`body` Delay

```
user=> (def d (delay (println "force delay object")
                     (+ 1 1)))
#'user/d

user=> d
#<Delay@15c5bba: :pending>

user=> @d
force delay object      ;
2                       ;

user=> @d
2                       ;  body

user=> d
#<Delay@15c5bba: 2>     ;  body
```

1.29 delay?

(delay? x)

`x delay Delay true`

```
user=> (delay? 2)
false

user=> (delay? (delay))
true
```

1.30 deref

(deref ref)
(deref ref timeout-ms timeout-val)

`deref` (reader macro) `@ref` / `@agent` / `@var` / `@atom` / `@delay` / `@future` / `@promise`

`deref ref` (in-transaction-value) `ref` (most-recently-committed value).

`var agent atom`

`future`

`promise promise deliver`

`timeout` (variant) `future promise` (reference) `timeout timeout-val`

```
;  deref

user=> (def d (delay (+ 1 1)))
```

```
#'user/d

user=> (deref d)
2

user=> @d
2

;  deref

user=> (def p (promise))
#'user/p

user=> (deref p 5000 nil)    ; 5  nil
nil
```

1.31 dissoc

(dissoc map)
(dissoc map key)
(dissoc map key & keys)

dissoc dissociate

`dissoc` Map `key` Map Map Map `key`

```
user=> (dissoc {:clojure "Rich" :python "Guido"})                              ;  key
{:python "Guido", :clojure "Rich"}

user=> (dissoc {:clojure "Rich" :python "Guido"} :python)                      ;  key
{:clojure "Rich"}

user=> (dissoc {:clojure "Rich" :python "Guido"} :ruby)                        ;
↪Map  key
{:python "Guido", :clojure "Rich"}

user=> (dissoc {:clojure "Rich" :python "Guido" :ruby "Matz"} :python :ruby)   ;  key
{:clojure "Rich"}
```

1.32 distinct

(distinct coll)

`coll`

```
;

user=> (distinct [1 2 1 2])
(1 2)


;
```

```
user=> (distinct [1 2 3 4])
(1 2 3 4)
```

1.33 doall

(doall coll)
(doall n coll)

`doall next`

Clojure 1.0

```
user=> (def one-to-three (range 3))
#'user/one-to-three

user=> (doall one-to-three)
(0 1 2)

user=> one-to-three
(0 1 2)
```

1.34 dorun

(dorun coll)
(dorun n coll)

`dorun next`

`nil`

Clojure 1.0

```
user=> (def infinity-hi (repeatedly #(println "hi")))
#'user/infinity-hi

user=> (dorun 5 infinity-hi)
hi
hi
hi
hi
hi
hi
nil

user=> (def one-to-ten (map println (range 10)))
#'user/one-to-ten

user=> (dorun one-to-ten)
0
1
```

```
2
3
4
5
6
7
8
9
nil
```

1.35 doseq

(doseq seq-exprs & body)

`for body`

`nil`

Clojure 1.0

```
user=> (doseq [i (range 10)] (prn i))
0
1
2
3
4
5
6
7
8
9
nil
```

1.36 empty?

(empty? coll)

`coll true (not (seq coll))`

`(seq x) (not (empty? x))`

```
user=> (empty? '())
true

user=> (empty? nil)
true

user=> (empty? [1 2 3])
false
```

1.37 even?

(even? n)

ntruen

```
(even? 2)
true
(even? 1)
false
```

1.38 first

(first coll)

coll

coll seq

coll nil nil

```
user=> (first nil)
nil

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

user=> (first (list 1 2 3))
1

user=> (first {:clojure "Rich" :python "Guido" :ruby "Matz"})
[:python "Guido"]
```

1.39 file-seq

(file-seq dir)

dir

dir java.io.File

```
;  file  File
;  /tmp

user=> (use '[clojure.java.io :only [file]])
nil

user=> (def tmp-folder (file "/tmp"))
#'user/tmp-folder

user=> (file-seq tmp-folder)
(#<File /tmp> #<File /tmp/.esd-1000> #<File /tmp/.esd-1000/socket> #<File /tmp/.Test-
↪unix> #<File /tmp/mongodb-27017.sock> #<File /tmp/at-spi2> #<File /tmp/at-spi2/
↪socket-1179-1131176229> #<File /tmp/at-spi2/socket-1268-1804289383> #<File /tmp/at-
↪spi2/socket-1169-1369485920> #<File /tmp/mongodb-28017.sock> #<File /tmp/.X0-lock> #
↪<File /tmp/.org.chromium.Chromium.NUsJHg> #<File /tmp/.org.chromium.Chromium.NUsJHg/
↪SingletonSocket> #<File /tmp/.org.chromium.Chromium.NUsJHg/SingletonCookie> #<File /
↪tmp/keyring-NwNaja> #<File /tmp/keyring-NwNaja/ssh> #<File /tmp/keyring-NwNaja/gpg>
↪#<File /tmp/keyring-NwNaja/pkcs11> #<File /tmp/keyring-NwNaja/control> #<File /tmp/.
↪ICE-unix> #<File /tmp/.ICE-unix/1303> #<File /tmp/cron.qpBNVU> #<File /tmp/pulse-
↪PKdhtXMmr18n> #<File /tmp/ssh-ElvUhBgb1303> #<File /tmp/ssh-ElvUhBgb1303/agent.1303>
↪ #<File /tmp/.font-unix> #<File /tmp/pulse-397VI5uG1yhc> #<File /tmp/pulse-
```

```
;  doseq  sort  println
;

user=> (doseq [f (sort (file-seq tmp))]
         (println f))
#<File /tmp>
#<File /tmp/.ICE-unix>
#<File /tmp/.ICE-unix/1303>
#<File /tmp/.Test-unix>
#<File /tmp/.X0-lock>
#<File /tmp/.X11-unix>
#<File /tmp/.X11-unix/X0>
#<File /tmp/.XIM-unix>
#<File /tmp/.esd-1000>
#<File /tmp/.esd-1000/socket>
#<File /tmp/.esd-120>
#<File /tmp/.font-unix>
#<File /tmp/.org.chromium.Chromium.NUsJHg>
#<File /tmp/.org.chromium.Chromium.NUsJHg/SingletonCookie>
#<File /tmp/.org.chromium.Chromium.NUsJHg/SingletonSocket>
#<File /tmp/at-spi2>
#<File /tmp/at-spi2/socket-1169-1369485920>
#<File /tmp/at-spi2/socket-1179-1131176229>
#<File /tmp/at-spi2/socket-1268-1804289383>
#<File /tmp/cron.qpBNVU>
#<File /tmp/hsperfdata_huangz>
#<File /tmp/hsperfdata_huangz/9350>
#<File /tmp/keyring-NwNaja>
#<File /tmp/keyring-NwNaja/control>
#<File /tmp/keyring-NwNaja/gpg>
#<File /tmp/keyring-NwNaja/pkcs11>
#<File /tmp/keyring-NwNaja/ssh>
#<File /tmp/mongodb-27017.sock>
#<File /tmp/mongodb-28017.sock>
#<File /tmp/pulse-397VI5uG1yhc>
#<File /tmp/pulse-397VI5uG1yhc/dbus-socket>
#<File /tmp/pulse-397VI5uG1yhc/native>
#<File /tmp/pulse-397VI5uG1yhc/pid>
#<File /tmp/pulse-PKdhtXMmr18n>
#<File /tmp/pulse-T9RwKSB1FebW>
#<File /tmp/ssh-ElvUhBgb1303>
#<File /tmp/ssh-ElvUhBgb1303/agent.1303>
nil
```

1.40 filter

(filter pred coll)

`coll (pred item) true`

`pred`

```
;  0 - 9
```

```
user=> (filter even? (range 10))
(0 2 4 6 8)


;  0 - 9

user=> (filter odd? (range 10))
(1 3 5 7 9)


;  0 - 9  10086

user=> (filter #(> % 10086) (range 10))
()
```

1.41 fn

params => positional-params* positional-params* & next-param
positional-param => binding-form
next-param => binding-form
name => symbol

```
user=> (fn greeting [name]                                          ;
           (str "Hello, " name " ."))
;#<user$eval1$greeting__2 user$eval1$greeting__2@616fde>

user=> ((fn greeting [name]                                         ;
           (str "hello, " name " ."))
        "moto")
"hello, moto ."

user=> ((fn greeting                                                ; arity␣
↪overloading
            ([name]
                (greeting "Hello" name))
            ([msg name]
                (str msg ", " name " ")))
        "moto")
"Hello, moto ."

user=> ((fn greeting                                                ;
            [name & others]
            (if (seq others)
                (str "Hello, " name " and others: " others " .")
                (str "Hello, " name " .")))
        "moto" "nokia" "apple")
"Hello, moto and others: (\"nokia\" \"apple\") ."
```

1.42 future

(future & body)

body future deref @

future body future

body deref

```
;  future

user=> (def f (future (+ 1 1)))
#'user/f

user=> f
#<core$future_call$reify__6110@fae040: 2>

user=> @f
2

;  5  future

user=> @(future (println "start sleep") (Thread/sleep 5000) 10086)
start sleep
10086

;  timeout  deref

user=> (deref (future (Thread/sleep 10000000000000000))
              1000
              "reach block timeout") ;  1
"reach block timeout"
```

1.43 get

(get map key)
(get map key not-found)

map key

key not-found not-found nil

```
user=> (def clojure {:author "Rich Hickey"})
#'user/clojure

user=> (get clojure :author)
"Rich Hickey"

user=> (get clojure :version)       ;  not-found
nil

user=> (get clojure :version 1.4)   ;  not-found
1.4
```

1.44 if-let

(if-let bindings then)
(if-let bindings then else & oldform)

bindings => binding-form test

test binding-form then

test else

```
user=> (defn sum-all-even-number [all-number]
           (if-let [all-even-number (filter even? all-number)]
               (reduce + all-even-number)
               0))
#'user/sum-all-even-number

user=> (sum-all-even-number [1 2 3 4 5 6 7 8 9])
20  ; 2 + 4 + 6 + 8

user=> (sum-all-even-number [1 3 5 7 9])
0
```

1.45 if-not

(if-not test then)
(if-not test then else)

test false then

test true else else nil

```
user=> (if-not false :then-part :else-part)
:then-part

user=> (if-not true :then-part)
nil

user=> (if-not true :then-part :else-part)
:else-part
```

1.46 import

(import & import-symbols-or-lists)

import-list => (package-symbol class-name-symbols*)

class-name-symbols name package.name namespace

ns :import

```
user=> (import java.util.Date)                          ;
java.util.Date

user=> (str (Date.))
"Wed Jun 20 23:18:42 CST 2012"

user=> (import '(java.util Date Calendar)               ;
               '(java.net URI ServerSocket))
java.net.ServerSocket

user=> (ns foo.bar                                      ;  ns
           (:import (java.util Date Calendar)
                    (java.net URI ServerSocket)))
java.net.ServerSocket
```

1.47 into

(into to from)

from-collto-coll

```
user=> (into (sorted-map) [ [:a 1] [:c 3] [:b 2] ] )
{:a 1, :b 2, :c 3}
user=> (into (sorted-map) [ {:a 1} {:c 3} {:b 2} ] )
{:a 1, :b 2, :c 3}
```

1.48 iterate

(iterate f x)

```
x (f x) (f (f x)) (f (f (f x)))
```

f

```
;

user=> (def z (iterate inc 1))
#'user/z


;

user=> (nth z 0)
1

user=> (nth z 1)
2


;
```

```
user=> (take 10 z)
(1 2 3 4 5 6 7 8 9 10)
```

1.49 keep-indexed

(keep-indexed f coll)

`coll item item index (keep-indexed f coll) (f index item) nil`

`keep-indexed f`

```
;  0 - 9 (index)

user=> (keep-indexed #(if (even? %1) %2 nil) (range 10))
(0 2 4 6 8)
```

1.50 keep

(keep f coll)

`coll item (keep f coll) (f item) nil`

`keep f`

```
user=> (keep inc [1 2 3])
(2 3 4)

;  collection  seq  nil
;  keep
;  nil

user=> (seq [])
nil

user=> (keep seq (list [1 2 3] [] [4 5 6]))
((1 2 3) (4 5 6))
```

1.51 key

(key e)

map

:: (map key {:a 1,:b 2}) (:a :b)

1.52 keys

(keys map)

`map` (key)

```
;  Map

user=> (keys {})
nil

;  Map

user=> (keys {:python "Guido" :clojure "Rich" :ruby "Matz"})
(:python :ruby :clojure)
```

1.53 letfn

(letfn fnspecs & body)

fnspecs ==> (fname [params*] exprs) (fname ([params*] exprs)+)

body (specs) fnspecs

```
user=> (letfn [(twice [x]
                  (* x 2))
               (six-times [y]
                  (* 3 (twice y)))]
           (println "Twice 15 = " (twice 15))
           (println "Six times 15 = " (six-times 15)))
Twice 15 = 30
Six times 15 = 90
nil

;;  twice  six-times  letfn

;user=> (twice 15)
;CompilerException java.lang.RuntimeException: Unable to resolve symbol: twice in␣
↪this context, compiling:(NO_SOURCE_PATH:7)

;user=> (six-times 15)
;CompilerException java.lang.RuntimeException: Unable to resolve symbol: six-times in␣
↪this context, compiling:(NO_SOURCE_PATH:8)
```

1.54 line-seq

(line-seq rdr)

rdr

rdr java.io.BufferedReader

```
;  reader  java.io.BufferedReader
;  animal.txt

user=> (use '[clojure.java.io :only [reader]])
nil
```

```
user=> (def animal (reader "animal.txt"))
#'user/animal

user=> (line-seq animal)
("dog" "cat" "monkey" "lion" "tiger" "dolphin")

user=> (.close animal)
nil


;  with-open
;  animal.txt

user=> (with-open [animal (reader "animal.txt")]
         (let [all-animal (line-seq animal)]
            (doseq [a all-animal]
                (println a))))
dog
cat
monkey
lion
tiger
dolphin
nil
```

1.55 map-indexed

(map-indexed f coll)

```
0 coll f  1 coll f  coll
f coll
```

```
user=> (map-indexed (fn [idx item] [idx item]) "foobar")
([0 \f] [1 \o] [2 \o] [3 \b] [4 \a] [5 \r])

user=> (map-indexed vector "foobar")    ;
([0 \f] [1 \o] [2 \o] [3 \b] [4 \a] [5 \r])
```

1.56 map

(map f coll)
(map f c1 c2)
(map f c1 c2 c3)
(map f c1 c2 c3 & colls)

```
collection f  collection f collection
```

collection collection collection

f collection

```
;;  collection

user=> (range 10)
(0 1 2 3 4 5 6 7 8 9)

user=> (map inc (range 10))
(1 2 3 4 5 6 7 8 9 10)


;;  collection

user=> (range 0 10)
(0 1 2 3 4 5 6 7 8 9)

user=> (reverse (range 0 10))
(9 8 7 6 5 4 3 2 1 0)

user=> (map #(+ %1 %2) (range 10) (reverse (range 10)))
(9 9 9 9 9 9 9 9 9 9)


;;  collection

user=> (map #(+ %1 %2) (range 10086) (reverse (range 10)))
(9 9 9 9 9 9 9 9 9 9)
```

1.57 mapcat

(mapcat f & colls)

(concat (map f colls))

```
user=> (mapcat reverse [[3 2 1 0]
                        [6 5 4]
                        [9 8 7]])
(0 1 2 3 4 5 6 7 8 9)
```

1.58 max-key

(max-key f item1)
(max-key f item1 item2)
(max-key f item1 item2 & items)

f (f item) item

(f item)

```
user=> (max-key count "abc"
                      "abcd"
                      "a"
                      "abcdefg"
                      "aa")
"abcdefg"
```

1.59 merge

(merge & maps)

map keymap

:: user=> (merge {:a 1 :b 2 :c 3} {:b 9 :d 4}) {:d 4, :a 1, :b 9, :c 3}

user=> (merge {:a 1 :b 2 :c 3} {:b 9 :d 4} {:b 8 :d 3}) {:d 3, :c 3, :b 8, :a 1}

1.60 min-key

(min-key f item1)
(min-key f item1 item2)
(min-key f item1 item2 & items)

```
f (f item) item
(f item)
```

```
user=> (min-key count "aaaaaaa"
                      "bbbbbb"
                      "ccccccc"
                      "aa"
                      "ddddddddd")
"aa"
```

1.61 next

(next coll)

```
coll
coll seq
coll (first coll) nil
```

```
user=> (next nil)
nil

user=> (next [1])
```

```
nil

user=> (next [1 2])
(2)

user=> (next [1 2 3])
(2 3)
```

1.62 odd?

(odd? n)

ntruen

```
(odd? 1)
true
(odd? 2)
false
(odd? 0)
false
```

1.63 partial

(partial f arg1)
(partial f arg1 arg2)
(partial f arg1 arg2 arg3)
(partial f arg1 arg2 arg3 & more)

`partial f f`

additional args `f`

```
user=> (def three-times (partial * 3))
#'user/three-times

user=> (three-times 10)                    ; (* 3 10)
30

user=> (three-times 20)                    ; (* 3 20)
60

user=> (defn add-x-y-z [x y z]
           (+ x y z))
#'user/add-x-y-z

user=> (def add-y-z (partial add-x-y-z 0))  ; x = 0
#'user/add-y-z

user=> (def add-z (partial add-y-z 1))      ; y = 1
#'user/add-z
```

```
user=> (add-z 2)                                    ; z = 2
3                                                   ; (+ 0 1 2)
```

1.64 pcalls

(pcalls & fns)

fns

```
;  3  3  3
user=> (pcalls
         #(Thread/sleep 3000)
         #(Thread/sleep 3000)
         #(Thread/sleep 3000))
(nil nil nil)
```

: pcalls pvalues

(Thread/sleep 3000)

```
user=> (for [i (pvalues 1 2 3
                        (Thread/sleep 3000)
                        (do (println "eval 4") 4)
                        (do (println "eval 5") 5)
                        (do (println "eval 6") 6))
             ]
           (println i)
       )
(1
2
nil eval 5  ;  println
eval 4      ; 4  5  6
3
nil eval 6
nil         ; sleep  3
nil 4
nil 5
nil 6
nil nil)
```

1.65 pmap

(pmap f coll)
(pmap f coll & colls)

pmap map pmap f

pmap (semi-lazy) (consumption) realize

f pmap

```
;  4  3  12
user=> (time
         (dorun
           (map (fn [x] (Thread/sleep 3000))
                (range 4))))
"Elapsed time: 12000.767484 msecs"
nil

;  4  3  3
user=> (time
         (dorun
           (pmap (fn [x] (Thread/sleep 3000))
                 (range 4))))
"Elapsed time: 3002.602211 msecs"
nil
```

1.66 promise

(promise)

promise deref @ deliver

promise deliver deref @ promise deref

deliver promise deref @ deliver

```
user=> (def p (promise))
#'user/p

user=> p
#<core$promise$reify__6153@bfb588: :pending>

;  promise  deref
;  5

user=> (deref p 5000 "reach timeout")
"reach timeout"

;  promise

user=> (deliver p 10086)
#<core$promise$reify__6153@bfb588: 10086>

user=> @p
10086

user=> p
#<core$promise$reify__6153@bfb588: 10086>

;  deliver
```

```
user=> (deliver p 123123)
nil

user=> @p
10086
```

1.67 pvalues

(pvalues & exprs)

exprs

```
user=> (pvalues
         (Thread/sleep 3000)
         10086
         (Thread/sleep 3000)
         "hello moto")
(nil 10086 nil "hello moto")    ;  3
```

: pcalls pvalues

(Thread/sleep 3000)

```
user=> (for [i (pvalues 1 2 3
                        (Thread/sleep 3000)
                        (do (println "eval 4") 4)
                        (do (println "eval 5") 5)
                        (do (println "eval 6") 6))
             ]
           (println i)
       )
(1
2
nil eval 5  ;  println
eval 4      ; 4  5  6
3
nil eval 6
nil         ; sleep  3
nil 4
nil 5
nil 6
nil nil)
```

1.68 range

(range)
(range end)

(range start end)
(range start end step)

`start end(start <= numbers < end) step`

`start 0 step 1 end`

```
;
; 0, 1, 2, 3 ...

user=> (take 3 (range))
(0 1 2)

user=> (take 10 (range))
(0 1 2 3 4 5 6 7 8 9)


;  end
;  0  end

user=> (range 5)
(0 1 2 3 4)

user=> (range 10)
(0 1 2 3 4 5 6 7 8 9)


;  start  end

user=> (range 5 10)
(5 6 7 8 9)

user=> (range 0 10)
(0 1 2 3 4 5 6 7 8 9)


;  start  end  step
;  2  20
;  1  20

user=> (range 2 20 2)
(2 4 6 8 10 12 14 16 18)

user=> (range 1 20 2)
(1 3 5 7 9 11 13 15 17 19)
```

1.69 re-seq

(re-seq re s)

`s re java.util.regex.Matcher.find() re-groups`

```
;
```

```
user=> (re-seq #"[0-9]+" "abs123def345ghi567")
("123" "345" "567")
```

1.70 realized?

(realized? x)

promise delay future lazy-list true

```
user=> (def d (delay (+ 1 1)))
#'user/d

user=> (realized? d)
false

user=> @d
2

user=> (realized? d)
true
```

1.71 reduce

(reduce f coll)
(reduce f val coll)

reduce

- **val**
  - coll f
  - coll f
  - coll coll f coll f
- **val**
  - coll f val
  - coll val coll f coll f

f coll

```
user=> (reduce + [])            ; coll  +  0
0

user=> (reduce + (range 10))    ; coll
45
```

```
user=> (reduce + 0 (range 10))  ; coll  val
45
```

1.72 reductions

(reductions f coll)
(reductions f init coll)

(reduce f coll)

init

```
user=> (reduce + (range 1 10))
45

user=> (reductions + (range 1 10))
(1 3 6 10 15 21 28 36 45)

user=> (reductions + 0 (range 1 10))    ;  init
(0 1 3 6 10 15 21 28 36 45)

user=> (reductions + 10 (range 1 10))   ;
(10 11 13 16 20 25 31 38 46 55)
```

1.73 remove

(remove pred coll)

coll (pred item) false

pred

```
;  0 - 9

user=> (remove even? (range 10))
(1 3 5 7 9)


;  0 - 9

user=> (remove odd? (range 10))
(0 2 4 6 8)


;  0 - 9  0

user=> (remove #(>= % 0) (range 10))
()
```

1.74 repeat

(repeat x)
(repeat n x)

n x

n x

```
;  10  "hi"

user=> (def ten-hi (repeat 10 "hi"))
#'user/ten-hi

user=> ten-hi
("hi" "hi" "hi" "hi" "hi" "hi" "hi" "hi" "hi" "hi")


;  "hi"

user=> (def infinite-hi (repeat "hi"))
#'user/infinite-hi

user=> (take 10 infinite-hi)
("hi" "hi" "hi" "hi" "hi" "hi" "hi" "hi" "hi" "hi")

user=> (take 20 infinite-hi)
("hi" "hi" "hi" "hi" "hi" "hi" "hi" "hi" "hi" "hi" "hi" "hi" "hi" "hi" "hi" "hi"
↪"hi" "hi" "hi")
```

1.75 repeatedly

(repeatedly f)
(repeatedly n f)

f () f n

n f

```
;

user=> (defn greet []
         "hi!")
#'user/greet


;  10  greet
;  take  5  10  greet

user=> (def ten-greet (repeatedly 10 greet))
#'user/ten-greet

user=> (take 5 ten-greet)
```

```
("hi!" "hi!" "hi!" "hi!" "hi!")

user=> (take 10 ten-greet)
("hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!")


;  10086  ten-greet  10
;  n

user=> (take 10086 ten-greet)
("hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!")


;  greet

user=> (def infinite-greet (repeatedly greet))
#'user/infinite-greet

user=> (take 10 infinite-greet)
("hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!")

user=> (take 100 infinite-greet)
("hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!"
↪"hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!"
↪"hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!"
↪"hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!"
↪"hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!"
↪"hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!"
↪"hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!"
↪"hi!" "hi!")
```

1.76 reverse

(reverse coll)

`coll`

(lazy)

```
(user=>(reverse [1 2 3 4])
(4 3 2 1)
```

1.77 rseq

(rseq rev)

`rev`

`rev` sorted-map

`rev nil`

```
;  sorted-map

user=> (rseq [])
```

```
nil

user=> (rseq (sorted-map))
nil


;

user=> (rseq [1 2 3])
(3 2 1)


;  sorted-map

user=> (def alpha (sorted-map :a "apple" :b "boy" :c "cat"))
#'user/alpha

user=> alpha
{:a "apple", :b "boy", :c "cat"}

user=> (rseq alpha)
([:c "cat"] [:b "boy"] [:a "apple"])


;

user=> (rseq (list 1 2 3))
ClassCastException clojure.lang.PersistentList cannot be cast to clojure.lang.
↪Reversible  clojure.core/rseq (core.clj:1480)

user=> (rseq nil)
NullPointerException   clojure.core/rseq (core.clj:1480)
```

1.78 rsubseq

(rsubseq sc test key)
(rsubseq sc start-test start-key end-test end-key)

subseq

`(rseq (subseq sc test key))` `(rseq (subseq sc start-test start-key end-test end-key))`

```
;

user=> (def numbers (sorted-map 1 "one" 2 "two" 3 "three" 4 "four" 5 "five"))
#'user/numbers

user=> numbers
{1 "one", 2 "two", 3 "three", 4 "four", 5 "five"}


;  1  4 -
```

```
user=> (rsubseq numbers >= 2 <= 4)
([4 "four"] [3 "three"] [2 "two"])


;  2 -

user=> (rsubseq numbers > 2)
([5 "five"] [4 "four"] [3 "three"])
```

1.79 seq

(seq coll)

coll

coll nil

(sql nil) nil

seq Java iterable

```
;  nil

user=> (seq [])
nil

user=> (seq nil)
nil

;  Map

user=> (seq [1 2 3])
(1 2 3)

user=> (seq (list 1 2 3))
(1 2 3)

user=> (seq {:language "clojure" :creator "Rich Hickey"})
([:creator "Rich Hickey"] [:language "clojure"])

user=> (seq "hello world")
(\h \e \l \l \o \space \w \o \r \l \d)
```

1.80 shuffle

(shuffle coll)

coll

```
user=> (shuffle [1 2 3 4])
[4 1 3 2]

user=> (shuffle [1 2 3 4])
[1 3 2 4]
```

1.81 some

(some pred coll)

`coll x (pred x)` ‘‘(pred x)‘‘

`coll x (pred x) nil`

`pred some (some #{:fred} coll) :fred coll :fred nil`

```
user=> (some #{:fred} [:fred :peter :jack])
:fred

user=> (some #{:mary} [:fred :peter :jack])
nil

user=> (some #(>= % 10) [1 3 5 7 9])            ;  >= 10
nil

user=> (some #(>= % 5) [1 3 5 7 9])             ;  >= 5
true
```

1.82 sort

(sort coll)
(sort comparator coll)

`coll`

`comparator compare. comparator java.util.Comparator`

```
user=> (sort [4 2 1 3])
(1 2 3 4)

user=> (sort >= [4 2 1 3])
(4 3 2 1)

user=> (sort <= [4 2 1 3])
(1 2 3 4)
```

1.83 subseq

(subseq sc test key)
(subseq sc start-test start-key end-test end-key)

`sc` (entry) `ek (true? (test (.. sc comparator (compare ek key)) 0))`

`nil`

`sc` sorted collection `test < <= > >=`

```
;

user=> (def numbers (sorted-map 1 "one" 2 "two" 3 "three" 4 "four" 5 "five"))
#'user/numbers

user=> numbers
{1 "one", 2 "two", 3 "three", 4 "four", 5 "five"}


;  3 -

user=> (subseq numbers >= 3)
([3 "three"] [4 "four"] [5 "five"])


;  1  4 -

user=> (subseq numbers >= 2 <= 4)
([2 "two"] [3 "three"] [4 "four"])


;  10 - nil

user=> (subseq numbers >= 10)
nil
```

1.84 time

(time expr)

`expr`

`expr`

```
;  100

user=> (time
          (dotimes [_ 100]
            (= :kw :kw)))
"Elapsed time: 0.23802 msecs"
nil
```

1.85 tree-seq

(tree-seq branch? children root)

`branch?`

`children`

`children branch? true`

`root`

```
;  file-seq
;  tree-seq

(defn file-seq
    [dir]
    (tree-seq
        (fn [^java.io.File f] (. f (isDirectory)))      ;  f
        (fn [^java.io.File d] (seq (. d (listFiles))))  ;  listFiles
        dir))                                           ;
```

1.86 vals

(vals map)

map (value)

```
;  Map

user=> (vals {})
nil

;  Map

user=> (vals {:python "Guido" :clojure "Rich" :ruby "Matz"})
("Guido" "Matz" "Rich")
```

1.87 when-let

(when-let bindings & body)
bindings => binding-form test

test value binding-form body

```
user=> (defn drop-one [coll]
           (when-let [s (seq coll)]
               (rest s)))
#'user/drop-one

user=> (drop-one [1 2 3])
(2 3)
```

1.88 when-not

(when-not test & body)

test false do body

```
user=> (when-not true
           (println "hello")
```

```
            (println "moto"))
nil

user=> (when-not false
            (println "hello")
            (println "moto"))
hello
moto
nil
```

1.89 with-open

(with-open bindings & body)

`name init try body  finally name (.close name)`

```
(ns io
  (:require [clojure.java.io :as io]))

(defn readLineByLine [file-name]
  (with-open [reader (io/reader file-name)]
    (doseq [line (line-seq reader)]
      (println line))))

(defn copyLineByLine [source target]
  (with-open [reader (io/reader source)
              writer (io/writer target)]
    (io/copy reader writer)))
```

1.90 xml-seq

(xml-seq root)

xml

xml `clojure.xml/parse`

```
;  xml

user=> (require 'clojure.xml)
nil

user=> (def content (clojure.xml/parse "http://www.w3schools.com/xml/note.xml"))
#'user/content

;  xml  xml

user=> (def tree (xml-seq content))
#'user/tree

user=> tree
({:tag :note, :attrs nil, :content [{:tag :to, :attrs nil, :content ["Tove"]} {:tag␣
↪:from, :attrs nil, :content ["Jani"]} {:tag :heading, :attrs nil, :content [
↪"Reminder"]} {:tag :body, :attrs nil, :content ["Don't forget me this weekend!"]}]}
↪{:tag :to, :attrs nil, :content ["Tove"]} "Tove" {:tag :from, :attrs nil, :content [
↪"Jani"]} "Jani" {:tag :heading, :attrs nil, :content ["Reminder"]} "Reminder" {:tag␣
↪:body, :attrs nil, :content ["Don't forget me this weekend!"]} "Don't forget me␣
↪this weekend!")
```

```
;

user=> (nth tree 0)
{:tag :note, :attrs nil, :content [{:tag :to, :attrs nil, :content ["Tove"]} {:tag␣
↪:from, :attrs nil, :content ["Jani"]} {:tag :heading, :attrs nil, :content [
↪"Reminder"]} {:tag :body, :attrs nil, :content ["Don't forget me this weekend!"]}]}

user=> (nth tree 1)
{:tag :to, :attrs nil, :content ["Tove"]}

user=> (nth tree 2)
"Tove"

user=> (nth tree 3)
{:tag :from, :attrs nil, :content ["Jani"]}

user=> (nth tree 4)
"Jani"
```

1.91 zipmap

(zipmap keys vals)

map

```
user=> (zipmap [:a :b :c :d :e] [1 2 3 4 5])
{:e 5, :d 4, :c 3, :b 2, :a 1}
```

CHAPTER 2

clojure.core.async

2.1 buffer

(buffer n)

nbufferbufferput

```
user=> (def c (chan (buffer 3)))
#'user/c
```

CHAPTER 3

clojure.data

3.1 diff

(diff a b)

abtuple [a b ab]

: * ab [nil,nil,a] * Map * Set * * (string)

```
(use 'clojure.data)
(def uno {:same "same", :different "one"})
(def dos {:same "same", :different "two", :onlyhere "whatever"})
(diff uno dos)
=> ({:different "one"} {:onlyhere "whatever", :different "two"} {:same "same"})
;;  {uno } {           dos            } {}

(diff {:a 1} {:a 1 :b 2})
=> (nil {:b 2} {:a 1})
;;


(diff [1 2 3] [5 9 3 2 3 7])              ;;=> [[1 2] [5 9 nil 2 3 7] [nil nil 3]]
(diff (set [1 2 3]) (set [5 9 3 2 3 7]))  ;;=> [#{1}  #{7 9 5}        #{3 2}]
```

CHAPTER 4

clojure.pprint

(A Pretty Printer for Clojure)

clojure.pprint clojurepretty printer println pprint the building blocks

pprint clojure source code clojure XML JSON

pprint Common lisp cl-format pprint clformat pprint pprint format function clojure format function

pprint cl-format

Added in Clojure version 1.2

4.1 cl-format

(cl-format writer format-in & args)

Common Lisp cl-format string

Writerjava.io.WriterStringformat-in args

format control string String.format dsl

writer cl-format string

: (let [results [46 38 22]] (cl-format true “There ~[are~;is~:;are~]~:* ~d result~:p: ~{~d~^, ~}~%” (count results) results))

out: There are 3 results: 46, 38, 22

format control string ”Common Lisp the Language, 2nd edition”, Chapter 22 (available online at: http://www.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/html/cltl/clm/node200.html#SECTION00263000000000000000) and in the Common Lisp HyperSpec at http://www.lispworks.com/documentation/HyperSpec/Body/22_c.htm

1 EXAMPLE

```
;; integer.
;;  *out*
user=> (cl-format true "~5d\n" 3)
3
nil

;; nilfalsesting
user=> (cl-format nil "~5d" 3)
"    3"

user=> (cl-format nil "Pad with leading zeros ~5,'0d" 3)
"Pad with leading zeros 00003"

user=> (cl-format nil "Pad with leading asterisks ~5,'*d" 3)
"Pad with leading asterisks ****3"


;;  formatString
;;  formatted  String  <width>


user=> (cl-format nil "~15a" (cl-format nil "~:d" 1234567))
"1,234,567      "

user=> (cl-format nil "Always print the sign ~5@d" 3)
"Always print the sign    +3"

user=> (cl-format nil "Use comma group-separator every 3 digits ~12:d" 1234567)
"Use comma group-separator every 3 digits 1,234,567"

user=> (cl-format nil "decimal ~d  binary ~b octal ~o  hex ~x" 63 63 63 63)
"decimal 63  binary 111111  octal 77  hex 3f"

user=> (cl-format nil "base 7  ~7r  with width and zero pad  ~7,15,'0r" 63 63)
"base 7  120  with width and zero pad 000000000000120"

;;  cl-format   BigInt,
;; BigInteger, or BigDecimal.
user=> (cl-format nil "cl-format handle   BigInts ~15d" 12345678901234567890)
"cl-format handles BigInts
12345678901234567890"

user=> (cl-format nil "Be aware of auto-conversion  ~8,'0d  ~8,'0d" 2.4 -5/4)
"Be aware of auto-conversion  000002.4
0000-5/4"

;; bug Common Lisp HyperSpec
;;  (format "%08d" -2)

user=> (cl-format nil "~8,'0d" -2)
"000000-2"
```

4.2 pp

(macro)

pprint `(pprint *1)`

Clojure 1.2

CHAPTER 5

clojure.string

5.1 blank?

(blank? s)

`s nil "" true`

```
user=> (clojure.string/blank? nil)
true

user=> (clojure.string/blank? "")
true

user=> (clojure.string/blank? "    ")
true

user=> (clojure.string/blank? "hello world")
false
```

5.2 reverse

(reverse s)

`s`

```
user=> (clojure.string/reverse "clojure")
"erujolc"
```

5.3 escape

(escape s cmap)

cmap s ch

- (cmap ch) nil ch
- (cmap ch) nil (str (cmap ch))

```
user=> (clojure.string/escape "I want 1 < 2 as HTML, & other good things." {\< "&lt;"␣
↪\> "&gt;" \& "&amp;"})
"I want 1 &lt; 2 as HTML, &amp; other good things."
```

5.4 replace

(replace s match replacement)

s match instance replacement

match / replacement

1. /
2. /
3. pattern /

```
;  1
;  moto  google

user=> (clojure.string/replace "hello moto" "moto" "google")
"hello google"

;  2
;  o  O

user=> (clojure.string/replace "hello moto" "o" "O")
"hellO mOtO"

;  3
;  #"red"  "blue"

user=> (clojure.string/replace "The color is red" #"red" "blue")
"The color is blue"

;  3
;

user=> (clojure.string/replace "The color is red" #"[aeiou]" clojure.string/upper-
↪case)
"ThE cOlOr Is rEd"
```

5.5 replace-first

(replace-first s match replacement)

replace match

```
;  e

user=> (clojure.string/replace-first "The color is red" #"[aeiou]" clojure.string/
↪upper-case)
"ThE color is red"
```

5.6 capitalize

(capitalize s)

s

```
user=> (clojure.string/capitalize "hello world")
"Hello world"

user=> (clojure.string/capitalize "HELLO WORLD")
"Hello world"
```

5.7 lower-case

(lower-case s)

s

```
user=> (clojure.string/lower-case "hello moto")
"hello moto"

user=> (clojure.string/lower-case "HELLO MOTO")
"hello moto"
```

5.8 upper-case

(upper-case s)

s

```
user=> (clojure.string/upper-case "hello moto")
"HELLO MOTO"

user=> (clojure.string/upper-case "HELLO MOTO")
"HELLO MOTO"
```

5.9 join

(join coll)
(join separator coll)

(seq coll) coll

separator

```
user=> (def fruit ["apple" "banana" "cherry"])
#'user/fruit

;  separator

user=> (clojure.string/join fruit)
"applebananacherry"

;  ", "  separator

user=> (clojure.string/join ", " fruit)
"apple, banana, cherry"
```

5.10 split

(split s re)
(split s re limit)

re s

limit

```
;  #"\r?\n"  clojure.string/split-lines

user=> (clojure.string/split "hello\nmoto\r\nagain\r\n" #"\r?\n")
["hello" "moto" "again"]

;  limit

user=> (clojure.string/split "hello\nmoto\r\nagain\r\n" #"\r?\n" 1)
["hello\nmoto\r\nagain\r\n"]

user=>  (clojure.string/split "hello\nmoto\r\nagain\r\n" #"\r?\n" 2)
["hello" "moto\r\nagain\r\n"]

user=>  (clojure.string/split "hello\nmoto\r\nagain\r\n" #"\r?\n" 3)
["hello" "moto" "again\r\n"]

user=>  (clojure.string/split "hello\nmoto\r\nagain\r\n" #"\r?\n" 4)
["hello" "moto" "again" ""]

user=>  (clojure.string/split "hello\nmoto\r\nagain\r\n" #"\r?\n" 5)
["hello" "moto" "again" ""]
```

5.11 split-lines

(split-lines s)

s \n \r\n

```
user=> (clojure.string/split-lines "hello\nmoto\r\nagain\r\n")
["hello" "moto" "again"]

user=> (clojure.string/split-lines "no-new-lines")
["no-new-lines"]

user=> (clojure.string/split-lines "")
[""]

user=> (clojure.string/split-lines nil)
;NullPointerException   java.util.regex.Matcher.getTextLength (Matcher.java:1234)
```

5.12 trim

```
user=> (clojure.string/trim "clojure")
"clojure"

user=> (clojure.string/trim "    clojure    ")
"clojure"
```

5.13 triml

(triml s)

s

```
user=> (clojure.string/triml "    clojure    ")
"clojure    "
```

5.14 trimr

(trimr s)

s

```
user=> (clojure.string/trimr "    clojure    ")
"    clojure"
```

5.15 trim-newline

(trim-newline s)

\n \r

Perl chomp

```
user=> (clojure.string/trim-newline "test\n")
"test"

user=> (clojure.string/trim-newline "test\r")
"test"

user=> (clojure.string/trim-newline "test\n\r")
"test"

user=> (clojure.string/trim-newline "test\r\n")
"test"

;

user=> (clojure.string/trim-newline "leading newline\n trailing newline\n")
"leading newline\n trailing newline"
```

CHAPTER 6

clojure.set

6.1 difference

(difference s1)
(difference s1 s2)
(difference s1 s2 & sets)

s1

```
user> (use 'clojure.set)
nil

;

user> (difference #{:a :b :c})
#{:a :c :b}

user> (let [s #{:a :b :c}] (identical? s (difference s)))
true

;

user> (difference #{:a :b :c} #{:a :b})
#{:c}

user> (difference #{:a :b :c} #{:a :b :c})
#{}

;
```

```
user> (difference #{:a :b :c} #{:a} #{:b})
#{:c}
```

6.2 index

(index xrel ks)

`index` SQL `group by` `xrel` `ks` `key`

`xrel` `ks` (sequence) `xrel` `map` `ks` `key`

`index` `map` `map` `key` `ks` `key` `xrel` `map` `map` `value` `xrel`

```
user> (use 'clojure.set)
nil

;;  set

user> (def points #{{:x 0 :y 0} {:x 0 :y 1} {:x 1 :y 0}}) ;;
#'user/points

user> (index points [:x])       ;; group by x
{{:x 1} #{{:y 0, :x 1}},
 {:x 0} #{{:y 1, :x 0} {:y 0, :x 0}}}

user> (index points [:y :x])    ;; group by x  y
{{:x 1, :y 0} #{{:y 0, :x 1}},
 {:x 0, :y 0} #{{:y 0, :x 0}},
 {:x 0, :y 1} #{{:y 1, :x 0}}}

user> (index points [:z])       ;; group by  z
{{} #{{:y 1, :x 0} {:y 0, :x 0} {:y 0, :x 1}}}

;;  vector

user> (def points-vec [{:x 0 :y 0} {:x 0 :y 1} {:x 1 :y 0}])
#'user/points-vec

user> (index points-vec [:x])
{{:x 1} #{{:y 0, :x 1}}, {:x 0} #{{:y 1, :x 0} {:y 0, :x 0}}}
```

6.3 intersection

(intersection s1)
(intersection s1 s2)
(intersection s1 s2 & sets)

```
user> (use 'clojure.set)
nil

user> (intersection #{1 2 3} #{3 4 5})
#{3}

user> (intersection #{1 2 3} #{2 3} #{3})
#{3}
```

6.4 join

(join xrel yrel)
(join xrel yrel km)

join SQL join xrel yrel

xrel yrel map map key-value

km km key

```
user> (use 'clojure.set)
nil

;;  set

user> (def students              ;;
        #{{:id 1 :name "Li Lei"}
          {:id 2 :name "Han Meimei"}})
#'user/students

user> (def score                 ;;
        #{{:id 1 :score 60}
          {:id 2 :score 99}})
#'user/score

user> (join students score)      ;;
#{{:score 99, :name "Han Meimei", :id 2}
  {:score 60, :name "Li Lei", :id 1}}

;;  vector

user> (def score-vec
         [{:id 1 :score 60}
          {:id 2 :score 99}])
#'user/score

user> (join students score-vec)
#{{:score 99, :name "Han Meimei", :id 2}
  {:score 60, :name "Li Lei", :id 1}}
```

6.5 map-invert

(map-invert m)

map value map key key map value

: key value map key

```
user> (use 'clojure.set)
nil

user> (map-invert {:a 1 :b 2})
{2 :b, 1 :a}

user> (map-invert {:a 1 :b 2 :c 2}) ;;  2  :c
{2 :b, 1 :a}
```

6.6 project

(project xrel ks)

xrel project key ks

xrel ks

```
user> (use 'clojure.set)
nil

user> (def points           ;;
  #{{:x 1 :y 0 :z 1}
    {:x 0 :y 1 :z 1}
    {:x 1 :y 1 :z 0}})
#'user/points

user> (project points [:x]) ;;  x
#{{:x 0} {:x 1}}            ;;  {:x 1}

user> (project points [:x :y])
#{{:y 1, :x 0} {:y 1, :x 1} {:y 0, :x 1}} ;;  x-y
```

6.7 rename

(rename xrel kmap)

xrel key kmap

```
user> (use 'clojure.set)
nil

user> (def students
  #{{:id 1 :name "Li Lei"}
    {:id 2 :name "Han Meimei"}})
#'user/students

user> (rename students {:id :student-id})
#{{:name "Han Meimei", :student-id 2} {:name "Li Lei", :student-id 1}}
```

6.8 rename-keys

(rename-keys map kmap)

map key kmap

array-map kmap

key key-value

```
user> (rename-keys {:id 1 :name "Li Lei"} {:id :new-id :name :new-name})
{:new-id 1, :new-name "Li Lei"}

;  key  {:b 2}

user> (rename-keys {:a 1 :b 2} {:a :b})
{:b 1}

;  array-map

user> (rename-keys  {:a 1 :b 2 :c 3}  (array-map :a :tmp :b :a :tmp :b))
{:b 1, :a 2, :c 3}
```

6.9 select

(select pred xset)

xset pred

select clojure.core/filter select set

```
user=> (use 'clojure.set)
nil
```

```
user=> (select even? #{1 2 3 4 5})
#{2 4}

user=> (select even? [1 2 3 4 5]) ;; set
ClassCastException clojure.lang.PersistentVector cannot be cast to clojure.lang.
↪IPersistentSet  clojure.core/disj (core.clj:1420)
```

6.10 subset?

(subset? set1 set2)

set1 set2

```
user> (clojure.set/subset? #{:a :b} #{:a :b :c})
true

user> (clojure.set/subset? #{:a :b} #{:a :c :d})
false

user> (clojure.set/subset? #{} #{:a :c :d})
true

user> (clojure.set/subset? #{:a} #{:a})
true
```

6.11 superset?

(superset? set1 set2)

set1 set2

```
user> (clojure.set/superset? #{:a :b :c} #{:a :b})
true

user> (clojure.set/superset? #{:a :c :d} #{:a :b})
false

user> (clojure.set/superset? #{:a :b} #{:a :b})
true

user> (clojure.set/superset? #{:a :b} #{})
true
```

6.12 union

(union)
(union s1)
(union s1 s2)
(union s1 s2 & sets)

```
user> (clojure.set/union)
#{}

user> (clojure.set/union #{:a} #{:b :c})
#{:a :c :b}

user> (clojure.set/union #{:a :b} #{:b :c})
#{:a :c :b}
```

CHAPTER 7

clojure.test

7.1 are

macro

(are argv expr & args)

(assertions) clojure.template/do-template

```
;;
(are [x y] (= x y)
           2 (+ 1 1)
           4 (* 2 2))

;;
(do (is (= 2 (+ 1 1)))
    (is (= 4 (* 2 2))))
```

CHAPTER 8

clojure.java.browse

8.1 browse-url

(browse-url url)

`url`

```
user=> (use 'clojure.java.browse)

user=> (browse-url "http://clojuredocs.org")
```

CHAPTER 9

clojure.java.io

9.1 Coercions

`clojure.java.io` protocol `as-file as-url` clojureprotocol

9.2 copy

(copy input output & opts)

`input output nil IOException`

`input java.io.InputStream java.io.Reader java.io.File byte java.lang.String java.lang.String`

`output java.io.OutputStream java.io.Writer java.io.File`

`opts :buffer-size encoding :buffer-size` 1024

`copy`

```
user> (use 'clojure.java.io)
nil
user> (copy "XXXXXX" (output-stream "/tmp/x"))
nil
user> (slurp "/tmp/x")
"XXXXXX"
user> (copy (file "/tmp/x") (output-stream "/tmp/xx"))
nil
user> (slurp "/tmp/x")
"XXXXXX"
```

9.3 IOFactory

clojure.java.io protocol make-reader make-writer make-input-stream make-out-stream readerwriterstream

API reader writer input-stream output-stream

9.4 input-stream

(input-stream x & opts)

x java.io.BufferedInputStream

x java.io.InputStream java.io.File java.net.URL java.net.URI java.lang.String java.net.Socket byte char

x java.lang.String x java.net.URL java.io.File

opts key :append :encoding

```
user> (use 'clojure.java.io)
nil
user> (input-stream (java.io.File. "/tmp/x"))
;;#<BufferedInputStream java.io.BufferedInputStream@21606a56>
user> (input-stream (java.io.File. "/tmp/x") :encoding "UTF-8")
;;#<BufferedInputStream java.io.BufferedInputStream@3e347b11>
```

9.5 as-file

(as-file x)

x java.io.File x java.lang.String java.io.File java.net.URL java.net.URI

x java.net.URL java.net.URI file

x nil nil

```
user> (use 'clojure.java.io)
nil
user> (.exists (as-file "/tmp"))
true
user> (.exists (as-file (java.io.File. "/tmp")))
true
user> (.exists (as-file (java.net.URL. "file:///tmp")))
true
user> (.exists (as-file (java.net.URL. "http://www.google.com")))
;;IllegalArgumentException Not a file: http://www.google.com  clojure.java.io/fn--
↪8210 (io.clj:67)
```

9.6 as-relative-file

(as-relative-path x)

x x java.lang.String java.io.File java.net.URL java.net.URI as-file

x IllegalArgumentException

```
user> (as-relative-path "./tmp")
"./tmp"
user> (as-relative-path "tmp")
"tmp"
user> (as-relative-path "/tmp")
;;IllegalArgumentException /tmp is not a relative path  clojure.java.io/as-relative-
↪path (io.clj:404)
user> (as-relative-path (java.io.File. "tmp-file"))
"tmp-file"
```

9.7 as-url

(as-url x)

x java.net.URL x java.lang.String java.io.File java.net.URL java.net.URI

x java.lang.String x URL

x nil nil

km km keyjoin

```
user> (use 'clojure.java.io)
nil
user> (as-url "http://baidu.com")
#<URL http://baidu.com>
user> (as-url (java.io.File. "/tmp"))
#<URL file:/tmp/>
user> (as-url (java.net.URI. "http://www.google.com"))
#<URL http://www.google.com>
user> (as-url "baidu.com")
;;MalformedURLException no protocol: baidu.com  java.net.URL.<init> (URL.java:567)
```

9.8 default-streams-impl

IOFactory IllegalArgumentException

9.9 delete-file

(delete-file f & [silently])

f

```
user> (use 'clojure.java.io)
nil
user> (spit "/tmp/x" "123") ;;
nil
user> (slurp "/tmp/x") ;;
"123"
user> (delete-file "/tmp/x") ;;
true
user> (slurp "/tmp/x") ;;
;;FileNotFoundException /tmp/x (No such file or directory)  java.io.FileInputStream.
↪open (FileInputStream.java:-2)
user> (delete-file "/tmp/x") ;;
;;IOException Couldn't delete /tmp/x  clojure.java.io/delete-file (io.clj:425)
user> (delete-file "/tmp/x" true) ;;
true
```

9.10 file

(file arg)
(file parent child)
(file parent child & more)

java.io.File

arg java.io.File

parent

parent,child more java.lang.String java.io.File java.net.URL java.net.URI

```
user> (use 'clojure.java.io)
nil
user> (file "/tmp")
#<File /tmp>
user> (file "/tmp" "a" "b")
#<File /tmp/a/b>
user> (file "/tmp" "a" (java.io.File. "../b"))
#<File /tmp/a/../b>
user> (file (java.net.URL. "file:///tmp") "a" (java.io.File. "../b"))
#<File /tmp/a/../b>
```

9.11 make-input-stream

(make-input-stream x opts)

x java.io.BufferedInputStream

x java.io.BufferedInputStream java.io.InputStream java.io.File java.net.URL java.net.URI java.lang.String java.net.Socket byte

x java.lang.String x java.net.URL java.io.File

opt mapkey :append :encoding

```
user> (use 'clojure.java.io)
nil
user> (make-input-stream "/tmp/x" {})
;;#<BufferedInputStream java.io.BufferedInputStream@3a7aa9f6>
user> (make-input-stream (java.io.File. "/tmp/x") {})
;;#<BufferedInputStream java.io.BufferedInputStream@df077d2>
user> (make-input-stream (java.io.File. "/tmp/NO_SUCH_FILE") {})
;;FileNotFoundException /tmp/NO_SUCH_FILE (No such file or directory)  java.io.
↪FileInputStream.open (FileInputStream.java:-2)
```

9.12 make-output-stream

(make-output-stream x opts)

x java.io.BufferedOutputStream

x java.io.BufferOutputStream java.io.OutputStream java.io.File java.net.URL java.net.URI java.lang.String java.net.Socket

x java.lang.String x java.net.URL java.io.File

x java.net.URL java.net.URI file

opt mapkey :append :encoding

```
user> (use 'clojure.java.io)
nil
user> (make-output-stream "/tmp/x" {})
;;#<BufferedOutputStream java.io.BufferedOutputStream@5440bf04>
user> (make-output-stream (file "/tmp/x") {})
;;#<BufferedOutputStream java.io.BufferedOutputStream@4268d15>
```

9.13 make-parents

(make-parents f & more)

true

f more java.lang.String java.io.File java.net.URL java.net.URI

```
user> (use 'clojure.java.io)
user> (make-parents "/tmp/a/b/c/d") ;;  /tmp/a/b/c/
true
user> (make-parents "/tmp/a/x/" "y" "z/") ;;  /tmp/a/x/y
true
```

9.14 make-reader

(make-reader x opts)

x java.io.BufferedReader x java.io.InputStream java.io.File java.net.URL java.net.URI java.lang.String java.net.Socket byte char

x java.lang.String x java.net.URL java.io.File

opt mapkey :append :encoding

```
user> (use 'clojure.java.io)
nil
user> (make-reader "http://baidu.com" {})
;;#<BufferedReader java.io.BufferedReader@5994a1e9>
```

9.15 make-writer

(make-writer x opts)

x java.io.BufferedWriter

x java.io.BufferOutputStream java.io.OutputStream java.io.File java.net.URL java.net.URI java.lang.String java.net.Socket

x java.lang.String x java.net.URL java.io.File

x java.net.URL java.net.URI file

opt mapkey :append :encoding

```
user> (use 'clojure.java.io)
nil
user> (make-writer (java.io.File. "/tmp/xxx") {})
;;#<BufferedWriter java.io.BufferedWriter@c92fa70>
```

9.16 output-stream

(output-stream x & opts)

x java.io.BufferedOutputStream

x java.io.BufferOutputStream java.io.OutputStream java.io.File java.net.URL java.net.URI java.lang.String java.net.Socket

x java.lang.String x java.net.URL java.io.File

x java.net.URL java.net.URI file

opt key :append :encoding

```
user> (use 'clojure.java.io)
nil
user> (input-stream (as-file "/tmp/x"))
;;#<BufferedInputStream java.io.BufferedInputStream@49160709>
user> (input-stream (as-file "/tmp/x") :append true)
;;#<BufferedInputStream java.io.BufferedInputStream@7f6ce64e>
```

9.17 reader

(reader x & opts)

x java.io.BufferedReader x java.io.InputStream java.io.File java.net.URL java.net.URI java.lang.String java.net.Socket byte char

x java.lang.String x java.net.URL java.io.File

opt mapkey :append :encoding

```
user> (use 'clojure.java.io)
nil
user> (reader "http://baidu.com" :encoding "GB2312")
;;#<BufferedReader java.io.BufferedReader@620a9239>
```

9.18 resource

(resource n)
(resource n loader)

ClassLoader java.net.URL nil

n ClassLoader loader

```
user> (use 'clojure.java.io)
nil
user> (resource "project.clj")
#<URL jar:file:/Users/xiafei/.lein/self-installs/leiningen-2.0.0-preview10-standalone.
↪jar!/project.clj>
```

9.19 writer

(writer x & opts)

x java.io.BufferedWriter

x java.io.BufferOutputStream java.io.OutputStream java.io.File java.net.URL java.net.URI java.lang.String java.net.Socket

x java.lang.String x java.net.URL java.io.File

x java.net.URL java.net.URI file

opt mapkey :append :encoding

```
user> (use 'clojure.java.io)
nil
user> (writer (java.net.URL. "file:///tmp/x") :append true)
;;#<BufferedWriter java.io.BufferedWriter@7274187a>
```

CHAPTER 10

clojure.java.javadoc

10.1 add-local-javadoc

(add-local-javadoc path)

`path` javadoc

10.2 add-remote-javadoc

(add-remote-javadoc package-prefix url)

`url` javadoc `package-prefix` URL javadoc

```
user=> (use 'clojure.java.javadoc)
nil

user=> (add-remote-javadoc "org.apache.commons.csv." "http://commons.apache.org/
↪proper/commons-csv/apidocs/index.html")
{"java." "http://java.sun.com/javase/6/docs/api/", "javax." "http://java.sun.com/
↪javase/6/docs/api/", "org.apache.commons.codec." "http://commons.apache.org/codec/
↪api-release/", "org.apache.commons.csv." "http://commons.apache.org/proper/commons-
↪csv/apidocs/index.html", "org.apache.commons.io." "http://commons.apache.org/io/api-
↪release/", "org.apache.commons.lang." "http://commons.apache.org/lang/api-release/",
↪ "org.ietf.jgss." "http://java.sun.com/javase/6/docs/api/", "org.omg." "http://java.
↪sun.com/javase/6/docs/api/", "org.w3c.dom." "http://java.sun.com/javase/6/docs/api/
↪", "org.xml.sax." "http://java.sun.com/javase/6/docs/api/"}
```

10.3 javadoc

(javadoc class-or-object)

`class-or-object` javadoc

`*local-javadocs*` `*remote-javadoc*`

```
user=> (use 'clojure.java.javadoc)
nil

user=> (javadoc String)
"http://java.sun.com/javase/6/docs/api/java/lang/String.html"

user=> (javadoc (java.util.Date.))
"http://java.sun.com/javase/6/docs/api/java/util/Date.html"
```

CHAPTER 11

clojure.java.shell

11.1 sh

(sh & args)

`Runtime.exec()`

- `:in clojure.java.io/copy, InputStream,Reader,File,byte[] String`, stdin
- `:in-enc UTF-8 ISO-8859-1 :in UTF-8 :in` (byte array),
- `:out-enc :bytes String String, UTF-8 ISO-8859-1 :bytes, UTF-8`
- `:env` map env `String[]`
- `:dir String java.io.File` dir

`with-sh-env with-sh-dir :env :dir`

`sh` map

- `:exit`
- `:out` stdout`byte[] String`
- `:err` stderr `String`

```
user=> (use '[clojure.java.shell :only [sh]])

;; Note: The actual output you see from a command like this will look messier.
;; The output below has had all newline characters replaced with line
;; breaks.  You would see a big long string with \n characters in the middle.
user=> (sh "ls" "-aul")

{:exit 0,
 :out "total 64
drwxr-xr-x  11 zkim  staff    374 Jul  5 13:21 .
```

```
drwxr-xr-x  25 zkim  staff    850 Jul  5 13:02 ..
drwxr-xr-x  12 zkim  staff    408 Jul  5 13:02 .git
-rw-r--r--   1 zkim  staff     13 Jul  5 13:02 .gitignore
-rw-r--r--   1 zkim  staff  12638 Jul  5 13:02 LICENSE.html
-rw-r--r--   1 zkim  staff   4092 Jul  5 13:02 README.md
drwxr-xr-x   2 zkim  staff     68 Jul  5 13:15 classes
drwxr-xr-x   5 zkim  staff    170 Jul  5 13:15 lib
-rw-r--r--@  1 zkim  staff   3396 Jul  5 13:03 pom.xml
-rw-r--r--@  1 zkim  staff    367 Jul  5 13:15 project.clj
drwxr-xr-x   4 zkim  staff    136 Jul  5 13:15 src
", :err ""}


user=> (use '[clojure.java.shell :only [sh]])

user=> (println (:out (sh "cowsay" "Printing a command-line output")))

 ___________________________________
< Printing a command-line output. >
 -----------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||

nil




user=> (use '[clojure.java.shell :only [sh]])
nil

;; note that the options, like :in, have to go at the end of arglist
;; advantage of piping-in thru stdin is less need for quoting/escaping
user=> (println (:out (sh "cat" "-" :in "Printing input from stdin with funny chars␣
↪like ' \" $@ & ")))
Printing input from stdin with funny chars like ' " $@ &
nil
```

11.2 with-sh-dir

(with-sh-dir dir & forms)

sh *sh*

11.3 with-sh-env

(with-sh-env env & forms)

sh *sh*

CHAPTER 12

clojure.xml

12.1 parse

(parse s)

(parse s startparse)

`s` s InputStream , URL

`xml/element` `struct-map` `struct-map` `:tag` `:attrs` `:content` `tag` `attrs` `content`

`startparse` sourceContentHandler

```
(require '[clojure.xml :as xml]
 '[clojure.zip :as zip])

;;convenience function, first sawn at nakkaya.com later in clj.zip src
(defn zip-str [s]
  (zip/xml-zip (xml/parse (java.io.ByteArrayInputStream. (.getBytes s)))))

;;parse from xml-strings to internal xml representation
(zip-str "<a href='nakkaya.com'/>")
=>
[{:tag :a, :attrs {:href "nakkaya.com"}, :content nil} nil]

;;root can be rendered with xml/emit-element
(xml/emit-element (zip/root [{:tag :a, :attrs {:href "nakkaya.com"}, :content nil}␣
↪nil]))
=>
<a href='nakkaya.com'/>
;;printed (to assure it's not lazy and performance), can be catched to string␣
↪variable with with-out-str
```

CHAPTER 13

clojure.repl

13.1 apropos

(apropos str-or-pattern)

(str-or-pattern)

```
user=> (apropos "temp")
()

user=> (require 'clojure.template)
nil

user=> (apropos "temp")
(apply-template do-template)

;;
user=> (apropos #".*-temp*")
(apply-template do-template)
```

13.2 dir

(dir nsname)

Var

```
user=> (require 'clojure.string 'clojure.repl)

user=> (clojure.repl/dir clojure.string)
blank?
```

```
capitalize
escape
join
lower-case
replace
replace-first
reverse
split
split-lines
trim
trim-newline
triml
trimr
upper-case
```

13.3 dir-fn

(dir-fn ns)

(ns)Var

```
user=> (require 'clojure.repl 'clojure.string)

user=> (pprint (clojure.repl/dir-fn 'clojure.string))
(blank?
 capitalize
 escape
 join
 lower-case
 replace
 replace-first
 reverse
 split
 split-lines
 trim
 trim-newline
 triml
 trimr
 upper-case)
nil
```

13.4 doc

(doc name)

Varspecial form

```
=> (doc map)
;; prints in console:
-------------------------
```

```
clojure.core/map
([f coll] [f c1 c2] [f c1 c2 c3] [f c1 c2 c3 & colls])
  Returns a lazy sequence consisting of the result of applying f to the
  set of first items of each coll, followed by applying f to the set
  of second items in each coll, until any one of the colls is
  exhausted.  Any remaining items in other colls are ignored. Function
  f should accept number-of-colls arguments.

=> (doc clojure.core)
-------------------------
clojure.core
  Fundamental library of the Clojure language
```

13.5 find-doc

(find-doc re-string-or-pattern)

VarVar

```
user=> (find-doc "data structure")

-------------------------
clojure.core/eval
([form])
  Evaluates the form data structure (not text!) and returns the result.
-------------------------
clojure.core/ifn?
([x])
  Returns true if x implements IFn. Note that many data structures
  (e.g. sets and maps) implement IFn
    user=> (require 'clojure.string 'clojure.repl)
-------------------------
........
```

13.6 pst

(pst)
(pst e-or-depth)
(pst e depth)

print stack trace

REPL

```
user=> (/ 1 0)
;; ArithmeticException Divide by zero  clojure.lang.Numbers.divide (Numbers.java:156)

user=> (pst)
;; ArithmeticException Divide by zero
;;     clojure.lang.Numbers.divide (Numbers.java:156)
```

```
;;     clojure.lang.Numbers.divide (Numbers.java:3691)
;;     user/eval13 (NO_SOURCE_FILE:7)
;;     clojure.lang.Compiler.eval (Compiler.java:6619)
;;     clojure.lang.Compiler.eval (Compiler.java:6582)
;;     clojure.core/eval (core.clj:2852)
;;     clojure.main/repl/read-eval-print--6588/fn--6591 (main.clj:259)
;;     clojure.main/repl/read-eval-print--6588 (main.clj:259)
;;     clojure.main/repl/fn--6597 (main.clj:277)
;;     clojure.main/repl (main.clj:277)
;;     clojure.main/repl-opt (main.clj:343)
;;     clojure.main/main (main.clj:441)
nil
```

CHAPTER 14

Clojure clojure.org/documentation

Clojure API clojure.github.com/clojure

Clojure Cheat Sheet clojure.org/cheatsheet

ClojureDocs API clojuredocs.org