Documentation

clojure.repl/ doc find-doc apropos dir source pst javadoc (foo.bar/ is

namespace for later syms)

Primitives

Numbers Literals

Long: 7, hex Oxff, oct 017, base 2 2r1011, base 36 36rCRAZY BigInt: 7N Ratio: -22/7 Double: 2.78 -1.2e-5 BigDecimal:

4.2M

+ - * / quot rem mod inc dec max min +' -' *' inc' dec' Arithmetic

== < > <= >= compare Compare

Ritwise bit-and bit-or bit-xor bit-not bit-flip bit-set

bit-shift-right bit-shift-left bit-and-not bit-clear bit-test unsigned-bit-shift-right (see BigInteger for integers larger than Long)

Cast byte short int long float double bigdec bigint num rationalize biginteger

zero? pos? neg? even? odd? number? rational? integer? ratio? decimal? float? (1.9) double? int? nat-int? Test

neg-int? pos-int?
rand rand-int

Random BigDecimal with-precision

Unchecked *unchecked-math* unchecked-add unchecked-dec unchecked-inc

unchecked-multiply unchecked-negate unchecked-subtract

Strings

Create str format "a string" "escapes \b\f\n\t\r\" octal \377 hex \ucafe"

See also section IO/to string

Use

count get subs compare (clojure.string/) join escape split split-lines replace replace-first reverse (1.8) index-of last-index-of

#"pattern" re-find re-seq re-matches re-pattern re-matcher re-groups
(clojure.string/) replace replace-first re-quote-replacement Note: \ Regex

in #"" is not escape char. (re-pattern "\\s*\\d+") can be written #"\s*\d+"

Letters (clojure.string/) capitalize lower-case upper-case

Trim

(clojure.string/) trim trim-newline trim1 trimr string? (clojure.string/) blank? (1.8) starts-with? ends-with? Test

includes?

Other

char char? char-name-string char-escape-string literals: \a Characters

\newline (more at link)

Keywords keyword keyword? find-keyword literals: :kw :my.name.space/kw

::in-cur-namespace ::namespace-alias/kw symbol symbol? gensym literals: my-sym my.ns/foo literals: true false nil Symbols

Misc

Collections

Collections

count empty not-empty into conj (clojure.walk/) walk prewalk Generic ops

prewalk-demo prewalk-replace postwalk postwalk-demo

postwalk-replace (1.9) bounded-count

distinct? empty? every? not-every? some not-any? sequential? associative? sorted? counted? reversible? coll? list? vector? set? map? seq? record? (1.8) Content tests Capabilities Type tests

map-entry?

Lists (conj, pop, & peek at beginning)

Create () list list*

first nth peek .indexOf .lastIndexOf cons conj rest pop Examine

'Change

Vectors (conj, pop, & peek at end)

Create [] vector vec vector-of mapv filterv (clojure.core.rrb-vector/) vector

vec vector-of

(my-vec idx) \rightarrow (nth my-vec idx) get peek .indexOf .lastIndexOf Examine

'Change assoc assoc-in pop subvec replace conj rseq update update-in

Ops reduce-kv

Create unsorted

Sets

Create sorted

sorted-set sorted-set-by (clojure.data.avl/) sorted-set sorted-set-by (flatland.ordered.set/) ordered-set (clojure.data.int-

map/) int-set dense-int-set

Examine $(my\text{-set item}) \rightarrow (get my\text{-set item}) contains?$

Change' conj disj Set ops

(clojure.set/) union difference intersection select See also sec-

tion Relations

(clojure.set/) subset? superset? Test

#{} set hash-set

Sorted sets rseq subseq rsubseq

Create unsorted {} hash-map array-map zipmap bean frequencies group-by (clo-

jure.set/) index

sorted-map sorted-map-by (clojure.data.avl/) sorted-map Create sorted

sorted-map-by (flatland.ordered.map/) ordered-map (clojure.data.priority-map/) priority-map (flatland.useful.map/)

ordering-map (clojure.data.int-map/) int-map

(my-map k) \rightarrow (get my-map k) also (:key my-map) \rightarrow (get my-map :key) get-in contains? find keys vals Examine

'Change assoc assoc-in dissoc merge merge-with select-keys update update-in (clojure.set/) rename-keys map-invert GitHub: Medley

reduce-kv Ops Entry kev val

Sorted maps rseq subseq rsubseq

Queues (coni at end. peek & pop from beginning)

Create clojure.lang.PersistentQueue/EMPTY (no literal syntax or constructor fn)

Examine 'Change conj pop

Relations (set of maps, each with same keys, aka rels)

Rel algebra $({\it clojure.set}/) \ {\it join select project union difference intersection}$

index rename

Transients (clojure.org/reference/transients)

transient persistent! Create

conj! pop! assoc! dissoc! disj! Note: always use return value for

later changes, never original!

Misc

Change

= identical? not= not compare clojure.data/diff
true? false? instance? nil? some? Compare

Test

Sequences

Creating a Lazy Seq

From collection seq vals keys rseq subseq rsubseq sequence

From producer fn lazy-seq repeatedly iterate

From constant repeat range

file-seq line-seq resultset-seq re-seq tree-seq xml-seq iterator-seq enumeration-seq From other

keep keep-indexed

From sea Seq in, Seq out

Get shorter distinct filter remove take-nth for dedupe random-sample cons conj concat lazy-cat mapcat cycle interleave interpose rest nthrest next fnext nnext drop drop-while take-last for Get longer Tail-items

take take-while butlast drop-last for

Head-items 'Change' coni concat distinct flatten group-by partition partition-all partition-by split-at split-with filter remove replace shuffle

Rearrange reverse sort sort-by compare

map pmap map-indexed mapcat for replace seque Process items

Using a Seg

Extract item first second last rest next ffirst nfirst fnext nnext nth

nthnext rand-nth when-first max-key min-key Construct coll

zipmap into reduce reductions set vec into-array to-array-2d mapv filterv apply

Pass to fn Search some filter

Force evaluation doseq dorun doall run! Check for forced realized?

Transducers (clojure.org/reference/transducers)

Off the shelf map mapcat filter remove take take-while take-nth drop drop-while replace partition-by partition-all keep keep-indexed map-indexed distinct interpose cat dedupe

random-sample (1.9) halt-when completing ensure-reduced unreduced See also section Concur-Create your own

rency/Volatiles

into sequence transduce eduction reduced reduced? deref Early termination

Spec (rationale, guide)

valid? conform unform explain explain-data explain-str Operations

explain-out form describe assert check-asserts

check-asserts?

Generator ops gen exercise exercise-fn Defn. & registry def fdef registry get-spec spec? spec with-gen

Logical and or

coll-of map-of every every-kv keys merge
cat alt * + ? & keys* Collection

Regex int-in inst-in double-in int-in-range? inst-in-range? Range

Other

nilable multi-spec fspec conformer
explain-printer *explain-out* Custom explain

Predicates with test.check generators Numbers

number? rational? integer? ratio? decimal? float? zero?
(1.9) double? int? nat-int? neg-int? pos-int? keyword? symbol? (1.9) ident? qualified-ident? qualified-keyword? qualified-symbol? simple-ident? Symbols

simple-keyword? simple-symbol?

string? true? false? nil? some? (1.9) boolean? bytes? Other

inst? uri? uuid? list? map? set? vector? associative? coll? sequential? seq? empty? (1.9) indexed? seqable? scalars Collections

10

to/from

keywords

to *out* ${\tt pr \ print \ printf \ println \ newline \ (clojure.pprint/) \ print-table}$ to writer (clojure.pprint/) pprint cl-format also: (binding [*out* writer] to string from *in* format with-out-str pr-str prn-str print-str println-str read-line (clojure.tools.reader.edn/) read from reader line-seq (clojure.tools.reader.edn/) read also: (binding [*in* reader] ...) java.io.Reader
with-in-str (clojure.tools.reader.edn/) read-string from string with-open (cloiure.java.jo/) text: reader writer binary: Open

spit slurp (to writer/from reader, Socket, string with file name, URI, etc.)

input-stream output-stream

(.write ostream byte-arr) (.read istream byte-arr) java.io.OutputStream java.io.InputStream GitHub: gloss

byte-spec flush (.close s) file-seq *in* *out* *err* (clojure.java.io/) file copy delete-file resource as-file as-url as-relative-path

GitHub: fs Data readers *data-readers* default-data-readers *default-data-reader-fn*

Functions

Binary

Misc

fn defn defn- definline identity constantly memfn comp complement Create partial juxt memoize fnil every-pred some-fn Call

apply -> ->> trampoline as-> cond-> cond->> some->> fn? ifn? Test

Define (defprotocol Slicey (slice [at])) Extend extend-type String Slicey (slice [at] ...)) Binding Forms (examples) let fn defn defmacro loop for doseq if-let Extend null (extend-type nil Slicey (slice [_] nil)) Destructuring when-let if-some when-some Reify (reify Slicey (slice [at] ...)) Test satisfies? extends? Vars and global environment (clojure.org/reference/vars) Other extend extend-protocol extenders Def variants def defn defn- definline defmacro defmethod defmulti defonce Records (clojure.org/reference/datatypes) defrecord Interned vars declare intern binding find-var var Define (defrecord Pair [h t]) Var objects with-local-vars var-get var-set alter-var-root var? bound? Access (:h (Pair. 12)) \rightarrow 1 Pair. ->Pair map->Pair record? Create Var validators set-validator! get-validator Test Types (clojure.org/reference/datatypes) Namespace Define (deftype Pair [h t]) Current *ns* (.h (Pair. 1 2)) → 1 Pair. ->Pair Access Create/Switch (tutorial) ns in-ns create-ns Create Add alias def import intern refer (deftype Pair [h t] Find all-ns find-ns Object (toString [this] (str "<" h "," t ">"))) With methods Examine ns-name ns-aliases ns-map ns-interns ns-publics ns-refers ns-imports From symbol resolve ns-resolve namespace the-ns (1.10) requiring-resolve Multimethods (clojure.org/reference/multimethods) Remove ns-unalias ns-unmap remove-ns Define (defmulti my-mm dispatch-fn) Method define (defmethod my-mm :dispatch-value [args] ...) Loading get-method methods Dispatch Load libs (tutorial) require use import refer Remove remove-method remove-all-methods prefer-method prefers Prefer Load misc load load-file load-reader load-string Relation derive underive isa? parents ancestors descendants make-hierarchy atom swap! reset! compare-and-set! (1.9) swap-vals! reset-vals! Atoms Futures future future-call future-done? future-cancel future-cancelled? Macros future? Threads bound-fn bound-fn* get-thread-bindings push-thread-bindings defmacro definline Create pop-thread-bindings thread-bound? volatile! vreset! vswap! volatile? macroexpand-1 macroexpand (clojure.walk/) macroexpand-all Debug Volatiles and or when when-not when-let when-first if-not if-let cond condp locking pcalls pvalues pmap seque promise deliver Misc case when-some if-some for doseq dotimes while Loop Refs and Transactions (cloiure.org/reference/refs) Arrange doto -> ->> as-> cond-> cond->> some->> Create ref binding locking time with-in-str with-local-vars with-open Scope $\texttt{deref @ (@form} \rightarrow (\texttt{deref form}))$ Examine with-out-str with-precision with-redefs with-redefs-fn Transaction sync dosync io! Lazv lazy-cat lazy-seq delay In transaction ensure ref-set alter commute Doc assert comment doc Validators set-validator! get-validator History ref-history-count ref-min-history ref-max-history Agents and Asynchronous Actions (clojure.org/reference/agents) Special Characters (clojure.org/reference/reader, guide) Create agent Comma reads as white space. Often used between map key/value Examine agent-error pairs for readability. send send-off restart-agent send-via Change state quote: 'form → (quote form) set-agent-send-executor! set-agent-send-off-executor! Namespace separator (see Primitives/Other section) await await-for set-validator! get-validator Block waiting Character literal (see Primitives/Other section) Ref validators Keyword (see Primitives/Other section) Watchers add-watch remove-watch Single line comment Thread handling shutdown-agents Metadata (see Metadata section) error-handler set-error-handler! error-mode set-error-mode! *foo 'earmuffs' - convention to indicate dynamic vars, compiler Misc *agent* release-pending-sends warns if not dynamic 0 Deref: $Qform \rightarrow (deref form)$ Java Interoperation (clojure.org/reference/java_interop) Syntax-quote .. doto Classname/ Classname. new bean comparator 'auto-gensym', consistently replaced with same foo# enumeration-seq import iterator-seq memfn set! class class? auto-generated symbol everywhere inside same '(...) bases supers type gen-class gen-interface definterface Unquote Cast boolean byte short char int long float double bigdec bigint num ~@ Unquote-splicing cast biginteger 'thread first' macro -> 'thread last' macro ->> throw try catch finally pst ex-info ex-data (1.9) Exceptions ->> StackTraceElement->vec (1.10) ex-cause ex-message (clojure.main/) core.async channel macros >!! <!! >! <! clojure.main/ex-str clojure.main/ex-triage List literal (see Collections/Lists section) Vector literal (see Collections/Vectors section) Arrays **{** Map literal (see Collections/Maps section) Create make-array object-array boolean-array byte-array short-array $\label{eq:var_quote} $$\operatorname{"Yx} \to (\operatorname{var} x)$$ $$\#"p"$ reads as regex pattern p (see Strings/Regex section) $$$ char-array int-array long-array float-array double-array aclone to-array to-array-2d into-array ** P reads as regex pattern p (see strings) regex section) Set literal (see Collections/Sets section) Anonymous function literal: #(...) \rightarrow (fn [args] (...)) Anonymous function argument: %N is value of anonymous function arg N. % short for %1. %% for rest args. #{ aget aset aset-boolean aset-byte aset-short aset-char aset-int Use #(aset-long aset-float aset-double alength amap areduce booleans bytes shorts chars ints longs floats doubles Reader conditional: #?(:clj x :cljs y) reads as x on JVM, #? Proxy (Clojure type selection flowchart) y in ClojureScript, nothing elsewhere. Other keys: :cljr:default Create proxy get-proxy-class construct-proxy init-proxy Misc proxy-mappings proxy-super update-proxy Splicing reader conditional: [1 #?@(:clj [x y] :cljs [w z]) 3] reads as [1 x y 3] on JVM, [1 w z 3] in ClojureScript, [1 3] elsewhere. #?@ Zippers (clojure.zip/) map namespace syntax e.g. #:foo{:a 1 :b 2} is equal to Create $\verb|zipper seq-zip vector-zip xml-zip|\\$ #foo Get loc up down left right leftmost rightmost lefts rights path children {:foo/a 1 :foo/b 2} 'Change make-node replace edit insert-child insert-left insert-right (1.9) symbolic values: ##Inf ##-Inf ##NaN ## append-child remove JavaContainerClass\$InnerClass Move next prev foo? conventional ending for a predicate, e.g.: zero? vector? instance? (unenforced) Misc root node branch? end? conventional ending for an unsafe operation, e.g.: set! Other swap! alter-meta! (unenforced) conventional name for an unused value (unenforced) XMI clojure.xml/parse xml-seq Ignore next form REPL *1 *2 *3 *e *print-dup* *print-length* *print-level* *print-meta* *print-readably* *compile-files* *compile-path* *file* *warn-on-reflection* compile Code loaded-libs test eval force hash name *clojure-version* clojure-version Metadata (clojure.org/reference/reader, special_forms) Misc *command-line-args* ^{:key1 val1 :key2 val2 ...} ^Type \rightarrow ^{:tag Type}, ^:key \rightarrow ^{:key true} ^:dynamic ^:private ^:doc ^:const General (clojure.java.browse/) browse-url (clojure.java.shell/) sh with-sh-dir Browser Abbrevs / Shell with-sh-env Common (def ^:dynamic *dyn-var* Examples (defn ^:private ^String my-fn ...)

Special Forms (clojure.org/reference/special_forms)

def if do let letfn quote var fn loop recur set! throw try monitor-enter

Abstractions (Clojure type selection flowchart)

Protocols (clojure.org/reference/protocols)

val)

meta with-meta vary-meta alter-meta! reset-meta! doc find-doc

On Vars