Numbers

Arithmetic	+ - * / quot rem mod inc dec max min +' -' *' inc' dec'
Compare / Test / Random	== < > <= >= compare zero? pos? neg? even? odd? number? integer? rand rand-int

Collections

Generic ops	count empty not-empty into conj
Content tests	distinct? empty? every? not-every? some not-any?
Type tests	coll? list? vector? set? map? seq?
Lists	list list* first nth peek .indexOf .lastIndexOf cons conj rest pop
Vectors	vector vec vector-of mapv filterv get peek .indexOf .lastIndexOf assoc assoc-in pop subvec replace conj update update-in
Sets	set hash-set contains? conj disj (clojure.set/) union difference intersection select subset? superset?
Maps	hash-map frequencies group-by get-in contains? find keys vals assoc assoc-in dissoc merge merge-with select-keys update update-in (clojure.set/) index rename-keys map-invert
Lazy Seqs	seq vals keys rseq subseq rsubseq sequence lazy-seq repeatedly iterate repeat range keep keep-indexed

Seq in, Seq out

Get shorter	distinct filter remove take-nth for dedupe random-sample
Get longer	cons conj concat lazy-cat mapcat cycle interleave interpose
Tail-items	rest nthrest next fnext nnext drop drop-while take-last for
Head-items	take take-while butlast drop-last for
'Change'	conj concat distinct flatten group-by partition partition-all partition-by split-at split-with filter remove replace shuffle
Rearrange	reverse sort sort-by compare
Process items	map pmap map-indexed mapcat for replace

Using a Seq

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 mapv filterv
Search	some filter
Force evaluation	doseq dorun doall run!

Strings

Use	count get subs (clojure.string/) join escape split split-lines replace replace-first reverse index-of last-index-of
Trim	(clojure.string/) trim trim-newline triml trimr
Test	string? (clojure.string/) blank? starts-with? ends-with? includes?

Misc

Compare	= identical? not= not compare
Test	true? false? instance? nil? some?

Functions

Create	fn defn identity constantly comp complement partial juxt memoize fnil
Call	apply -> ->> trampoline as->
Test	fn? ifn?

Types etc.

Protocols	(defprotocol Slicey (slice [at])) (extend-type String Slicey (slice [at] …)) extend extend-protocol extenders
Records	(defrecord Pair [h t]) (:h (Pair. 1 2)) → 1 Pair. ->Pair map->Pair
Types	(deftype Pair [h t]) (.h (Pair. 1 2)) → 1 Pair. ->Pair
Multimethods	(defmulti my-mm dispatch-fn) (defmethod my-mm :dispatch-value [args] ...)

Macros

Branch	and or when when-not when-let when-first if-not if-let cond condp case when-some if-some
Loop	for doseq dotimes while
Arrange	.. doto -> ->> as-> cond-> cond->> some-> some->>
Lazy	lazy-cat lazy-seq delay

Concurrency

Atoms	atom swap! reset!
Refs and Transactions	ref deref dosync ensure ref-set alter commute
Agents	agent send send-off await await-for
Watchers	add-watch remove-watch

Special Forms

def if do let letfn quote var fn loop recur throw try