Clojure cheat sheet

Online references:

- clojuredocs.org (go to the clojure core quick reference)
- clojure.org/sequences – generic sequence operations (map, reduce, etc.)
- clojure.org/data_types – vectors, maps, etc.
- clojure.org/special_forms – let, defn, control flow
- clojure.org/documentation (index page for clojure docs)

Misc.

`doc` – get documentation for a function. `(doc conj)`

`prn` – print data as literals

Functions

`defn` – define a function. `(defn add1 [x] (+ 1 x))`

`fn` – anonymous function. `(fn [x] (+ 1 x))`

`apply` – call a function with a collection as arguments.

`(apply vector (list 1 2)) ;=> [1 2]`

Control flow

`let` – bind local variables. `(let [x 1] x) ;=> 1`

`if` – `(if true :then :else) ;=> :then`

`and`, `or` – short-circuiting, variadic

`cond` – multibranch if. `(cond (even? 1) :a (odd? 3) :b :else :c) ;=> :b`

`loop` – tail recursion/iteration.

`quote` – suspend computation

`do` – execute multiple statements in one expression.

Collections

`first` – Get the first element of a collection `rest` – Get the rest of a collection after the first

`conj` – add an element to a collection. `(conj [1 2] 3) ;=> [1 2 3]`

`reduce` – left fold. `(reduce + [1 2 3]) ;=> 6`

`map` – `(map (fn [x] (* 2 x)) [1 2 3]) ;=> (2 4 6)`

`count` – number of entries in a collection

`empty?` – is a collection empty?

`list? vector? set? map?` – is a thing of a type

`filter` – keep items matching a predicate

`concat` – concatenate two collections

`interleave` – `(interleave [:x :y] [1 2]) ;=> (:x 1 :y 2)`

Lists

`list` – Make a list `(list 1 2 3) ;=> (1 2 3)`

`list*` – make a list from another collection `(list* 1 [2 3]) ;=> (1 2 3)`

Vectors

Create: `[] vector`

Examine `(my-vec idx)` → `(nth my-vec idx)`

Maps

Create: `{} hash-map zipmap`

Examine: `(:key my-map)` → `(get my-map :key)`

`contains?` – Does a map contain a key?

`keys` – Keys in a map

`vals` – values in a map

`assoc` – associate a key with a value

Macros

Create `defmacro`

`macroexpand-1` – expand form exactly once

`macroexpand` – expand until form is not a macro call

`clojure.walk/macroexpand-all` – expand all macros within the form until no macro calls are left