

ClojureScript Cheat Sheet

<http://github.com/clojure/clojurescript>

Documentation

<http://github.com/clojure/clojurescript/wiki>
<http://www.clojurescript.net>
<http://www.clojuredocs.org>

Namespace Declaration

```
(ns my-cool-lib
  (:require [some-lib :as lib])
  (:use [another-lib :only (a-func)])
  (:require-macros [my.macros :as macs])
  (:use-macros [mo.macs :only (my-mac)]))
```

Rich Data Literals

Maps: `{:key1 :val1, :key2 :val2}`
Vectors: `[1 2 3 4 :a :b :c 1 2]`
Sets: `#{:a :b :c 1 2 3}`
Truth/nullity: `true, false, nil`
Keywords: `:kw, :a-2, :prefix/kw, ::pi`
Symbols: `sym, sym-2, prefix/sym`
Characters: `\a, \u1123, \space, \newline`
Numbers/Strings: `same as in JavaScript`
Regex: `#"[Cc]lojure[Ss]cript"`

Frequently Used Functions & Macros

Functions

Math: `+ - * / quot rem mod inc dec max min`
Comparison: `= == not= < > <= >=`
Tests: `nil? identical? zero? pos? neg? even? odd? true? false? nil?`
Keywords: `keyword keyword?`
Symbols: `symbol symbol? gensym`
Data Processing: `map reduce filter partition split-at split-with`
Data Create: `vector vec hash-map set list list* for`
Data Examination: `first rest count get nth get get-in contains? find keys vals`
Data Manipulation: `seq into conj cons assoc assoc-in dissoc zipmap merge merge-with select-keys update-in`
Arrays: `into-array to-array aget aset amap areduce alength`

Macros

Defining: `defmacro`
Implementation: `Must be written in Clojure`
Emission: `Must emit ClojureScript`
Macros: `if if-let cond and or -> -> doto when when-let ..`

Extra ClojureScript Libraries

`clojure.{string set zipper}`
`clojure.browser.{dom event net repl}`

Abstraction (<http://clojure.org/protocols>)

Protocols

Definition: `(defprotocol Slicey (slice [at]))`
Extend: `(extend-type js/String Slicey (slice [at] ...))`
Extend null: `(extend-type nil Slicey (slice [] nil))`
Reify: `(reify Slicey (slice [at] ...))`

Records

Definition: `(defrecord Pair [h t])`
Access: `(:h (Pair. 1 2)) ;=> 1`
Constructing: `Pair. ->Pair map->Pair`

Types

Definition: `(deftype Pair [h t])`
Access: `(.-h (Pair. 1 2)) ;=> 1`
Constructing: `Pair. ->Pair`
With Method(s): `(deftype Pair [h t] Object (toString [] ...))`

Multimethods

Definition: `(defmulti my-mm dispatch-function)`
Method Define: `(defmethod my-mm :dispatch-value [args] ...)`

JS Interop (<http://fogus.me/cljs-js>)

Method Call: `(.meth obj args)`
`(. obj (meth args))`
Property Access: `(.-prop obj)`
`(. obj -prop)`
`(aget obj prop-str)`
Set Property: `(set! (.-prop obj) val)`
`(aset obj prop-str val)`
Set Array element: `(aset arr idx val)`
JS Global Access: `js/window`
JS this: `(this-as me (.method me))`
Create JS Object: `(js-obj)`
Create JS Array: `(array var-args)`
`(make-array size)`
Transf. JS value: `(js->clj js-val)`
Transf. CLJ value: `(clj->js clj-val)`

Compilation (<http://fogus.me/cljsc>)

`cljsc src-home`
Compile: `'{:optimizations :simple :pretty-print true}'`
Adv. Compile: `cljsc src-home`
`'{:optimizations :advanced}'`

Other Useful Libraries

Lein build: <https://github.com/emezeske/lein-cljsbuild>
Client/Server: <http://github.com/cemerick/shoreleave-remote-ring>
DOM: <http://github.com/levand/domina>
jQuery: <http://github.com/ibdknox/jayq>
Templating: <https://github.com/Prismatic/dommy>