

ClojureScript Cheat Sheet

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

Documentation

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

Listing 1: Example Namespace Declaration

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

Rich Data Literals

Maps:	<code>{:key1 :val1, :key2 :val2}</code>
Vectors:	<code>[1 2 3 4 :a :b :c 1 2]</code>
Sets:	<code>#{:a :b :c 1 2 3}</code>
Truth and nullity:	<code>true, false, nil</code>
Keywords:	<code>:kw, :a-2, :prefix/kw, :pi</code>
Symbols:	<code>sym, sym-2, prefix/sym</code>
Characters:	<code>\a, \u1123, \space</code>
Int, Float, String:	same as in JavaScript

Frequently Used Functions

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

More information

<http://clojuredocs.org>

Frequently Used Macros

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

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

Protocols

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

Records

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

Types

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

Multimethods

Definition:	<code>(defmulti my-mm dispatch-function)</code>
Method Define:	<code>(defmethod my-mm :dispatch-value [args] ...)</code>

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

Method Call:	<code>(.meth obj args)</code>
Method Call:	<code>(. obj (meth args))</code>
Property Access:	<code>(. obj -prop)</code>
Property Access:	<code>(.-prop obj)</code>
Set Property:	<code>(set! (.-prop obj) val)</code>
JS Direct Access:	<code>js/something</code>
JS this:	<code>(this-as me (.method me))</code>
Create JS Object:	<code>(js-obj)</code>

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

Simple Compile:	<code>cljsc src-home '{:optimizations :simple :pretty-print true}'</code>
Advanced Compile:	<code>cljsc src-home '{:optimizations :advanced}'</code>

Extra ClojureScript Libraries

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

Other Useful Libraries

App Sample:	http://clojurescriptone.com
Client/Server:	http://github.com/ibdknox/fetch
D3:	http://github.com/lynaghk/cljs-d3
DOM:	http://github.com/levand/domina
jQuery:	http://github.com/ibdknox/jayq
Templating:	http://github.com/ibdknox/crate