

General Info

;Everything is a function
that means everything is enclosed in brackets
;multiple pieces of data often need to be passed in a vector
(get-in myFunc [:ok :go])
;commas are considered whitespace in clojure

Primitives

Integer

- corresponds to long int

Float

- corresponds to double

String

- Always enclosed by double quotes
- "My big dog"

Boolean

- true or false

nil

- is used as the NULL value

Binding

; general form
; similar to type variable_name
= value
(def name value)
; or int x = 4
(def x 4)

IF (Control Flow)

; general syntax
(if boolean-form
the n-form
optional-else-form)
; an example (if else)
(if (> 5 0)
"positive"

IF (Control Flow) (cont)

> "negative")
;-> "positive"
; optional else
(if (> 5 0)
"positive")
;-> "positive"
(if (< 5 0)
"negative")
;-> nil
; Maximum of two terms per if. The following is an invalid form
(if (> 5 0)
"positive"; then 1
"good"; then 2
"negative"); else

Arity (or Number of Arguments)

; basic arity overloading
(defn func_name
([arg1] ; one parm version
(expression))
([arg1 arg2] ; two parm version
(expression)))
; using other arities
; similar to constructors in java
(defn func_name
([arg1]
(expression))
([arg1 arg2]
(func_name arg1)) ; call the
with one arg
(expression))

Lists (Composite Data Structures)

;basic structure is '(elem1 elem2 elem3)
(1 2 3) ; => error: 1 is not a function
'(1 2 3) ; => valid list
(+ 2 2) ; => function call
'(+ 2 2) ; => valid list
;very similar to vectors, but have to use nth instead of get
(nth '(1 2 3) 1) ; => 2
(nth '(1 2 3) 10) ; error

Basic Maths & Logic

; simple Math
(+ 1 1)
(- 2 1)
(* 1 2)
(/ 2 1)
; Equality testing
(= 1 1) ; => true
(= 2 1) ; => false
; Negation
(not true) ; => false

Function Syntax

;best way to bind functions
(defn hello [] "hello world")
;anonymous function
(fn [parm list] body)
;example
(fn [] "hello world")
;without the defn keyword have to do this
(def hello (fn [] "hello world"))



By [melbatoast](#)

Not published yet.
Last updated 25th April, 2018.
Page 1 of 3.

Sponsored by CrosswordCheats.com
Learn to solve cryptic crosswords!
<http://crosswordcheats.com>

DO (Control Flow)

```
;used for executing more than
one line
;will return the last value of
the line
(if (> 5 0)
  (do
    (pr intln "looks good")
    "po sit ive")
  (do
    (pr intln "looks bad")
    "ne gat ive"))
;-> " pos iti ve"
```

When (Control Flow)

```
;when is a combination of IF and
DO
(when (> 5 0)
  (pr intln "looks good")
  "po sit ive")
;-> " pos iti ve"
;always returns nil on false
```

Immutability and Adding

```
;Clojure data structures are
immutable.
;That means once you assign data
to a variable, you can't change
it
(def foo1 `(1 2 3))
(conj foo1 0) ; => (0 1 2 3)
foo1 ; => (1 2 3), no change
(def foo2 (conj foo1 0))
foo1 ; => (1 2 3)
foo2 ; => (0 1 2 3)
(def foo [1 2 3])
foo ; => [1 2 3]
(assoc foo 1 "dog") ; => [1
"dog" 3]
foo ; => [1 2 3]
```

Vectors (Composite Data Structure)

```
;Vectors are essentially arrays
indexed at 0
;two forms are
(def myVec [1 2 3])
(def myVec (vector 1 2 3))
;again can put functions inside
(def things [
  "dog"
  2
  (fn [] "do someth ing")
])
;and of course we can use get to
access the data
(get things 0) ; => "dog"
(get things 1) : => 2
(get things 10) : => nil
```

Truthy, Falsey & Nil

```
;Can test for nil
- (nil? "boo") ; => false
- (nil? nil) ; => true
;false and nil are considered
falsey
;every thing else is considered
truthy
```

Printing

```
(print "woo")
; => "woo"
(print "woo" "hoo")
; => "woo hoo"
(print "Number:" 4)
; => "Number: 4"
```

Nested Maps (Composite Data Structure)

```
can nest a map in a map
(def nest {
  :a "eh"
  :b {:dog "fido" :cat "fluffy"}
})
(get nest :a) ; => "eh"
(get nest :b) ; => {:dog "fido",
:cat "fluffy"}
;use the get-in function to
access inner maps
(get-in nest [:b :cat]) ; =>
"fluffy"
```

Set (Composite Data Structure)

```
;like in Python, the Clojure set
maintains a collection of unique
values
#{1 2 "dog" "cat"}
(hash-set 1 2 "dog" :c)
;sets will automa tically
discard duplicates
#{1 2 "dog" 2} ; => #{1 2 "dog"}
;can also create sets from lists
and vectors
;use the set keyword
(set `( 1 1 2 2)) ; => #{1 2}
(set ["dog" "cat" "dog"]) ; => #
{"dog" "cat"}
;searching the set
;can use get, a keyword (:john)
or contains?
(def foo #{ 1 2 3 :a} )
(get foo 3) ; => 3
(get foo 4) ; => nil
(:a foo) ; =>:a, returns itself
(contains? foo 2) ; => true
```



By [melbatoast](#)

cheatography.com/melbatoast/

Not published yet.

Last updated 25th April, 2018.

Page 2 of 3.

Sponsored by [CrosswordCheats.com](#)

Learn to solve cryptic crosswords!

<http://crosswordcheats.com>

Maps (Composite Data Structures)

```
; maps are key value pairs
; like dictionaries from python

(def jobs {
  :john "professor"
  :sue "doctor"
  :ahmed "astronaut"
})

; keywords
; :john is a keyword, which is
basically a string
; it just has faster internal
lookup
; the get function
; returns the value associated w
a key
(get jobs :sue) ; => "doctor"
(get jobs :Sue) ; => nil
; function lookup table
(def funcs {
  :func1 (fn [] (println
"func1"))
  :func2 (fn [] (println
"func2"))
  :func3 (fn [] (println
"func3"))
})
; execute the third function
((get funcs :func3)) ; => func3
```



By [melbatoast](#)

cheatography.com/melbatoast/

Not published yet.

Last updated 25th April, 2018.

Page 3 of 3.

Sponsored by [CrosswordCheats.com](#)

Learn to solve cryptic crosswords!

<http://crosswordcheats.com>