WE MAKE SOFTWARE

WE MAKE SOFTWARE

METOSIN

WE MAKE SOFTWARE

clojure bootcamp

clojure bootcamp

clojure bootcamp

who are we?

Christophe Grand
λ→next
@cgrand

Juho Teperi
@JuhoTeperi

Tommi Reiman
@ikitommi

Jarppe Länsiö
@jarppe

clojure bootcamp

who are we?

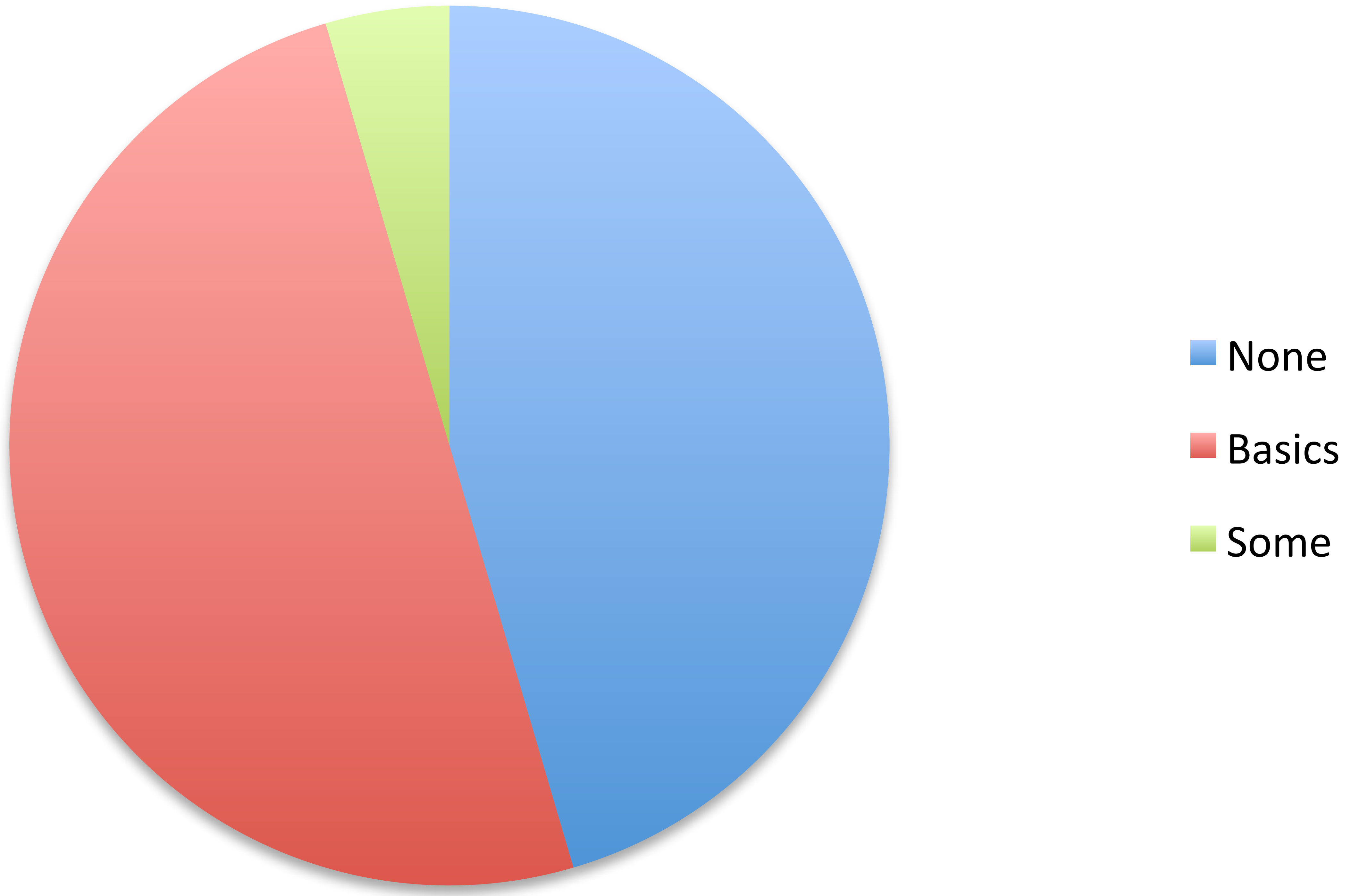
clojure bootcamp

who are you?

clojure bootcamp

who are you?

clojure experience



clojure bootcamp

who are you?

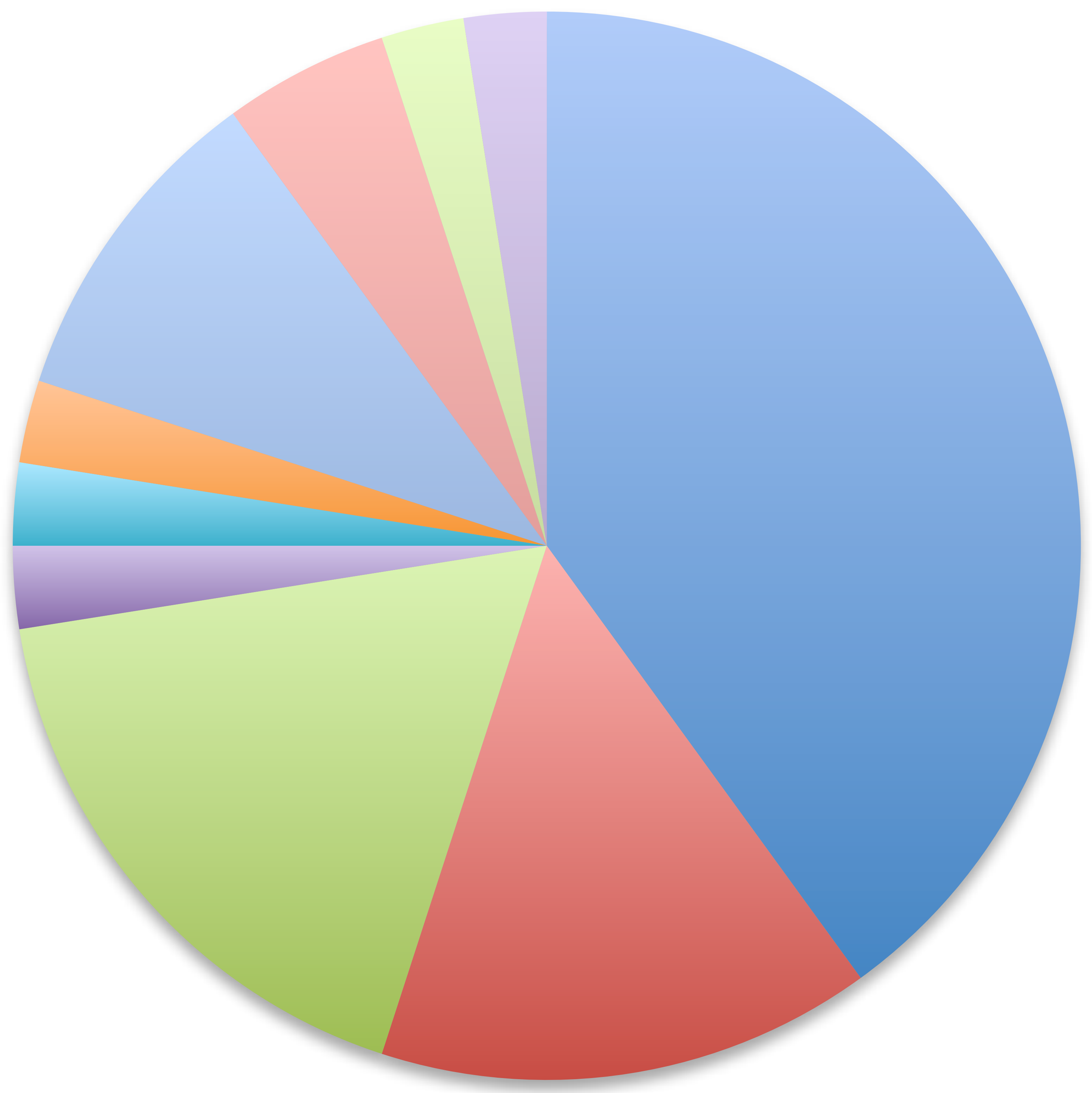
clojure bootcamp

who are you?

clojure bootcamp

who are you?

other FP languages



clojure bootcamp

who are you?

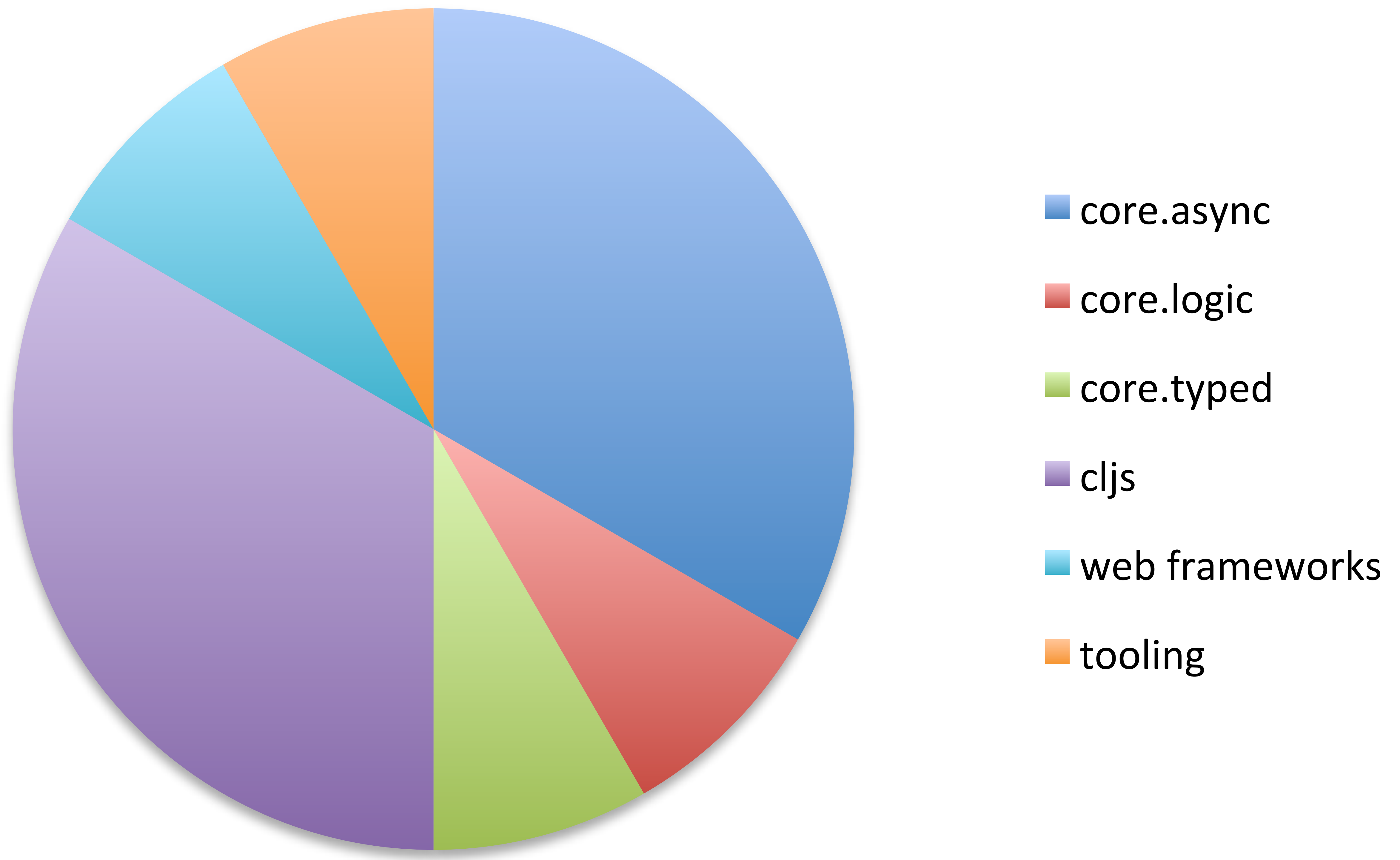
clojure bootcamp

who are you?

clojure bootcamp

who are you?

advanced interests



clojure bootcamp

who are you?

clojure bootcamp

agenda day 1

clojure basics

clojure programming

clojure bootcamp

agenda day 2

STM & core.async

schema

clojure for backend

clojure for frontend

magic

clojure basics

clojure basics

start your engines

clojure basics

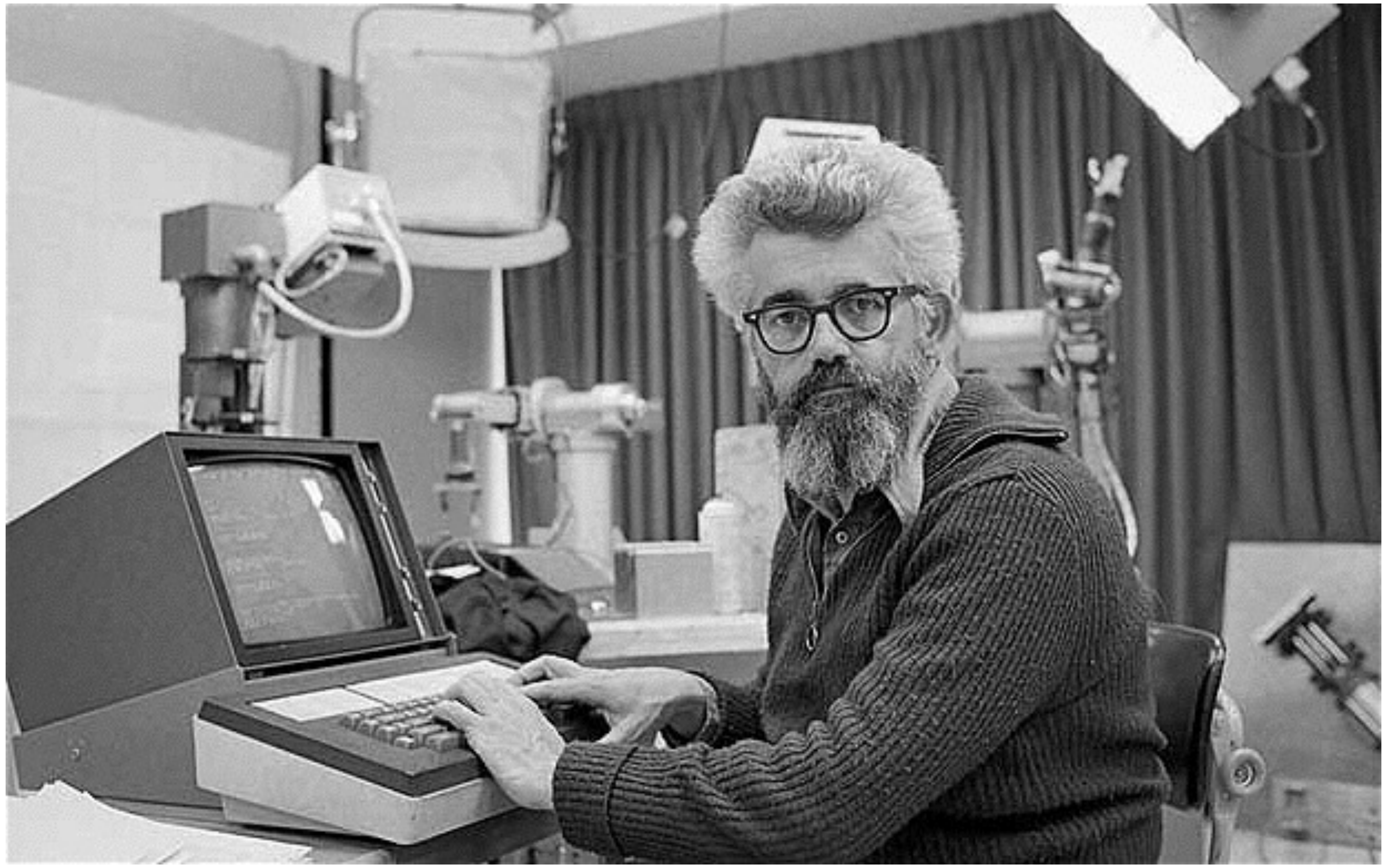
lisp?

clojure basics

lisp?

clojure basics

lisp?


John McCarthy

clojure basics

lisp?

evaluation

special forms

clojure basics

evaluation

clojure basics

evaluation

```
plus(2,  div(80, 2))
```

clojure basics

evaluation

```
(plus 2, (div 80, 2))
```

clojure basics

evaluation

```
(plus 2  (div 80  2))
```

clojure basics

evaluation

```
(+    2  (/    80  2))
```

clojure basics

evaluation

```
(+    2  (/   80  2))
(+    2       40    )
```

clojure basics

evaluation

$$\texttt{(+ 2 }\underbrace{\texttt{(/ 80 2)}}\texttt{)}$$

$$\underbrace{\texttt{(+ 2 40)}}_{\texttt{42}}$$

clojure basics

evaluation

```
(defmacro postfix [a op b]
  (list op a b))
```

clojure basics

evaluation

```
(defmacro postfix [a op b]
  (list op a b))
```

```
(postfix 2 + (postfix 80 / 2))
```

clojure basics

evaluation

```
(defmacro postfix [a op b]
  (list op a b))
```

```
(postfix 2 + (postfix 80 / 2))
```

```
(+ 2 (postfix 80 / 2))
```

clojure basics

evaluation

```
(defmacro postfix [a op b]
  (list op a b))
```

```
(postfix 2 + (postfix 80 / 2))

    (+ 2 (postfix 80 / 2))

        (+ 2 (/ 80 2))
```

clojure basics

evaluation

```
(defmacro postfix [a op b]
  (list op a b))
```

```
(postfix 2 + (postfix 80 / 2))

(+ 2 (postfix 80 / 2))

(+ 2 (/ 80 2))

(+ 2 40)
```

clojure basics

evaluation

```
(defmacro postfix [a op b]
  (list op a b))
```

```
(postfix 2 + (postfix 80 / 2))

(+ 2 (postfix 80 / 2))

(+ 2 (/ 80 2))

(+ 2 40)

42
```

clojure basics

evaluation

special forms

clojure basics

special forms

clojure basics

special forms

```
def

fn

if

let

do
```

http://clojure.org/special_forms

clojure basics

special forms

```
def

fn

if

let

do
```

clojure basics

special forms

def

fn

if

let

do

```
(def message "hello")
```

clojure basics

special forms

def

fn

if

let

do

```
(def message "hello")

=> message
"hello"
```

clojure basics

special forms

```
def

fn

if

let

do
```

clojure basics

special forms

def

fn

if

let

do

```
(fn [a b]
  (+ a b))
```

clojure basics

special forms

```
def

fn

if

let

do
```

clojure basics

special forms

def

fn

if

let

do

```
=> (if true
      "hello")
"hello"
```

clojure basics

special forms

def

fn

if

let

do

```
=> (if true
     "hello")
"hello"

=> (if false
     "hello"
     "world")
"world"
```

clojure basics

special forms

def

fn

if

let

do

```
=> (if true
      "hello")
"hello"

=> (if false
      "hello"
      "world")
"world"

=> (if (= (+ 39 3) 42)
      "ultimate answer"
      "no answer for you")
"ultimate answer"
```

clojure basics

truthy in the land of clojure

clojure basics

truthy in the land of clojure

clojure basics

truthy in the land of clojure

rule #1   `false` and `nil` are *falsey*

clojure basics

truthy in the land of clojure

rule #1 `false` and `nil` are *falsey*

rule #2 everything else is *truthy*

clojure basics

special forms

def

fn

if

let

do

```
=> (if true
      "hello")
"hello"

=> (if false
      "hello"
      "world")
"world"

=> (if (= (+ 39 3) 42)
      "ultimate answer"
      "no answer for you")
"ultimate answer"
```

clojure basics

special forms

`def`

`fn`

`if`

`let`

`do`

clojure basics

special forms

def

fn

if

let

do

```
(defn how-let-works? [a b]
  (println "before let: a:" a "b:" b)
  (let [a "x"]
    (println "inside let: a:" a "b:" b))
  (println "after let:  a:" a "b:" b))
```

clojure basics

special forms

def

fn

if

let

do

```
(defn how-let-works? [a b]
  (println "before let: a:" a "b:" b)
  (let [a "x"]
    (println "inside let: a:" a "b:" b))
  (println "after let:  a:" a "b:" b))
```

```
=> (how-let-works? "a" "b")
before let: a: a b: b
inside let: a: x b: b
after let:  a: a b: b
nil
```

clojure basics

special forms

def

fn

if

let

do

clojure basics

special forms

def

fn

if

let

do

```
=> (if (= 42 (+ 2 (/ 80 2)))
     (println "Math still works...")
     "It's 42"
     (println "Something is horribly wrong")
     "I don't even")
```

clojure basics

special forms

def

fn

if

let

do

```
=> (if (= 42 (+ 2 (/ 80 2)))
     (println "Math still works...")
     "It's 42"
     (println "Something is horribly wrong")
     "I don't even")
CompilerException java.lang.RuntimeException: Too
many arguments to if
```

clojure basics

special forms

def

fn

if

let

do

```
=> (if (= 42 (+ 2 (/ 80 2)))
     (do
       (println "Math still works...")
       "It's 42")
     (do
       (println "Something is horribly wrong")
       "I don't even"))
Math still works...
"It's 42"
```

clojure basics

literals

clojure basics

literals

Long	BigInteger	Double	BigDecimal	Boolean
`42`	`1337N`	`3.14159`	`2.71828M`	`true`
`2r101010`		`299.792458e6`		`false`

clojure basics

literals

Character

```
\F
\newline
\u24B6
```

String

```
"hello, world!"
"Copyright \u00A9 2014"
```

Regex pattern

```
#"hello, (\S+)"
```

clojure basics

literals

List

```
(println "hello")
```

Vector

```
["hello" 42]
```

Set

```
#{"foo" "bar"}
```

Map

```
{"c"          "Dennis"
 "python"     "Guido"
 "clojure"    "Rich"}
```

clojure basics

literals

Keyword

```
:hello
:ring.util.http-response/response
```

Symbol

```
hello
clojure.core/println
```

clojure basics

literals

nil (null)

```
nil
```

clojure basics

syntax

clojure basics

syntax

; comment, rest of the line is ignored

```
; Print some primes
(println 2 3 5 7 11 13)
```

, whitespace, ignored

```
[{:lang "c", :year 1969}
 {:lang "lisp", :year 1958}
 {:lang "clojure", :year 2007}]
```

clojure basics

syntax

;   comment, rest of the line is ignored

```
; Print some primes
(println 2 3 5 7 11 13)
```

,   whitespace, ignored

```
[{:lang “c”   :year 1969}
 {:lang “lisp”   :year 1958}
 {:lang “clojure”   :year 2007}]
```

clojure basics

syntax

; comment, rest of the line is ignored

```
; Print some primes
(println 2 3 5 7 11 13)
```

, whitespace, ignored

```
[{:lang "c", :year 1969}
 {:lang "lisp", :year 1958}
 {:lang "clojure", :year 2007}]
```

clojure basics

functions

clojure basics

functions

functional programming?

clojure basics

functions

functional programming?

clojure basics

functions

functional programming?

…style of building the structure and elements of computer programs, that treats computation as the evaluation of mathematical functions and avoids state and mutable data

http://en.wikipedia.org/wiki/Functional_programming

clojure basics

functions

…style

program

mathem

http://en.wikipe

mputer

n of

ble data

clojure basics

functions

functional programming?

…style of building the structure and elements of computer programs, that treats computation as the evaluation of mathematical functions and avoids state and mutable data
http://en.wikipedia.org/wiki/Functional_programming

clojure basics

functions

functional programming?

…style of building the structure and elements of computer programs, that treats computation as the evaluation of mathematical functions and avoids state and mutable data

http://en.wikipedia.org/wiki/Functional_programming

clojure basics

functions

functional programming?

…style of building the structure and elements of computer programs, that treats computation as the evaluation of mathematical functions and avoids state and mutable data

http://en.wikipedia.org/wiki/Functional_programming

first-class functions

higher-order functions

pure functions

clojure basics

functions

functional programming?

…style of building the structure and elements of computer programs, that treats computation as the evaluation of mathematical functions and avoids state and mutable data
http://en.wikipedia.org/wiki/Functional_programming

first-class functions create stand-alone functions
higher-order functions
pure functions

clojure basics

functions

functional programming?

…style of building the structure and elements of computer programs, that treats computation as the evaluation of mathematical functions and avoids state and mutable data
http://en.wikipedia.org/wiki/Functional_programming

first-class functions	create stand-alone functions
higher-order functions	functions as arguments and return values
pure functions	

clojure basics

functions

functional programming?

…style of building the structure and elements of computer programs, that treats computation as the evaluation of mathematical functions and avoids state and mutable data
http://en.wikipedia.org/wiki/Functional_programming

first-class functions	create stand-alone functions
higher-order functions	functions as arguments and return values
pure functions	immutable data-structures

clojure basics

functions

```
(fn [a b] (+ a b))
```

clojure basics

functions

```
(def plus (fn [a b] (+ a b)))
```

clojure basics

functions

```
(def plus (fn [a b] (+ a b)))

(plus 39 3)
```

clojure basics

functions

```
(def plus (fn [a b] (+ a b)))
```

clojure basics

functions

```
(def plus (fn [a b] (+ a b)))
(defn plus [a b] (+ a b))
```

clojure basics

functions

```
(def plus (fn [a b] (+ a b)))

(defn plus [a b]
  (+ a b))
```

clojure basics

functions

```
(def plus (fn [a b] (+ a b)))

(defn plus [a b]
  (+ a b))
```

side note:
indent is 2 spaces

clojure basics

functions

```
(def plus

(defn plus
  (+ a b))
```

side note:
indent is 2 spaces

clojure basics

functions

```
(def plus (fn [a b] (+ a b)))

(defn plus [a b]
  (+ a b))
```

side note:
indent is 2 spaces

clojure basics

functions

clojure basics

functions

```clojure
(defn no-args []
  42)
```

clojure basics

functions

```
(defn no-args []                 => (no-args)
  42)                            42
```

clojure basics

functions

```
(defn no-args []                  => (no-args)
  42)                             42


(defn one-arg [a]
  a)
```

clojure basics

functions

```
(defn no-args []            => (no-args)
  42)                       42


(defn one-arg [a]           => (one-arg)
  a)                        ArityException Wrong number of args (0) passed to: example$one-arg…
```

clojure basics

functions

```
(defn no-args []
  42)
```

```
=> (no-args)
42
```

```
(defn one-arg [a]
  a)
```

```
=> (one-arg)
ArityException Wrong number of args (0) passed to: example$one-arg…

=> (one-arg 1337)
1337
```

clojure basics

functions

```clojure
(defn no-args []
  42)
```

```
=> (no-args)
42
```

```clojure
(defn one-arg [a]
  a)
```

```
=> (one-arg)
ArityException Wrong number of args (0) passed to: example$one-arg…

=> (one-arg 1337)
1337
```

```clojure
(defn two-args [a b]
  [a b])
```

clojure basics

functions

```
(defn no-args []              => (no-args)
  42)                         42


(defn one-arg [a]             => (one-arg)
  a)                          ArityException Wrong number of args (0) passed to: example$one-arg…

                              => (one-arg 1337)
                              1337


(defn two-args [a b]          => (two-args 42 1337)
  [a b])                      [42 1337]
```

clojure basics

functions

clojure basics

functions

```
(defn one-or-two-args
  ([a]
    (one-or-two-args a 0))
  ([a b]
    [a b]))
```

clojure basics

functions

```clojure
(defn one-or-two-args
  ([a]
    (one-or-two-args a 0))
  ([a b]
    [a b]))
```

```
=> (one-or-two-args)
ArityException Wrong number of args (0) passed to: …
=> (one-or-two-args 1)
[1 0]
=> (one-or-two-args 1 2)
[1 2]
```

clojure basics

functions

```
(defn one-or-two-args
  ([a]
    (one-or-two-args a 0))
  ([a b]
    [a b]))


(defn any-args [& args]
  args)
```

```
=> (one-or-two-args)
ArityException Wrong number of args (0) passed to: …
=> (one-or-two-args 1)
[1 0]
=> (one-or-two-args 1 2)
[1 2]
```

clojure basics

functions

```
(defn one-or-two-args
  ([a]
    (one-or-two-args a 0))
  ([a b]
    [a b]))
```

```
=> (one-or-two-args)
ArityException Wrong number of args (0) passed to: …
=> (one-or-two-args 1)
[1 0]
=> (one-or-two-args 1 2)
[1 2]
```

```
(defn any-args [& args]
  args)
```

```
=> (any-args)
nil
=> (any-args 1)
(1)
=> (any-args 1 2 3)
(1 2 3)
```

clojure basics

functions

```
(defn one-or-two-args
  ([a]
    (one-or-two-args a 0))
  ([a b]
    [a b]))
```

```
=> (one-or-two-args)
ArityException Wrong number of args (0) passed to: …
=> (one-or-two-args 1)
[1 0]
=> (one-or-two-args 1 2)
[1 2]
```

```
(defn any-args [& args]
  args)
```

```
=> (any-args)
nil
=> (any-args 1)
(1)
=> (any-args 1 2 3)
(1 2 3)
```

```
(defn one-or-more-args [a & more]
  [a more])
```

clojure basics

functions

```clojure
(defn one-or-two-args
  ([a]
    (one-or-two-args a 0))
  ([a b]
    [a b]))
```

```
=> (one-or-two-args)
ArityException Wrong number of args (0) passed to: …
=> (one-or-two-args 1)
[1 0]
=> (one-or-two-args 1 2)
[1 2]
```

```clojure
(defn any-args [& args]
  args)
```

```
=> (any-args)
nil
=> (any-args 1)
(1)
=> (any-args 1 2 3)
(1 2 3)
```

```clojure
(defn one-or-more-args [a & more]
  [a more])
```

```
=> (one-or-more-args)
ArityException Wrong number of args (0) passed to:…
=> (one-or-more-args 1)
[1 nil]
=> (one-or-more-args 1 2)
[1 (2)]
=> (one-or-more-args 1 2 3 4)
[1 (2 3 4)]
```

clojure basics

functions

many things in clojure are functions

clojure basics

functions

many things in clojure are functions

clojure basics

functions

many things in clojure are functions

```
=> (def clojure {:invented 2007
                 :inventor "Rich Hickey"})
#'user/clojure
```

clojure basics

functions

many things in clojure are functions

```
=> (def clojure {:invented 2007
                 :inventor "Rich Hickey"})
#'user/clojure

=> (get clojure :inventor)
"Rich Hickey"
```

clojure basics

functions

many things in clojure are functions

```
=> (def clojure {:invented 2007
                 :inventor "Rich Hickey"})
#'user/clojure

=> (get clojure :inventor)
"Rich Hickey"

=> (clojure :inventor)
"Rich Hickey"
```

clojure basics

functions

many things in clojure are functions

```
=> (def clojure {:invented 2007
                 :inventor "Rich Hickey"})
#'user/clojure

=> (get clojure :inventor)
"Rich Hickey"

=> (clojure :inventor)
"Rich Hickey"

=> (:invented clojure)
2007
```

clojure basics

namespaces

clojure basics

namespaces

clojure basics

namespaces

Used to group things

clojure basics

namespaces

Used to group things

Analogous to packages in Java

clojure basics

namespaces

Used to group things

Analogous to packages in Java

Mapping from symbols to values

clojure basics

namespaces

Used to group things

Analogous to packages in Java

Mapping from symbols to values via `var`'s

clojure basics

namespaces

clojure basics

namespaces

```
(ns metosin.bootcamp.greeter)

(def message "Hello")

(defn greet [your-name]
  (println message your-name))
```

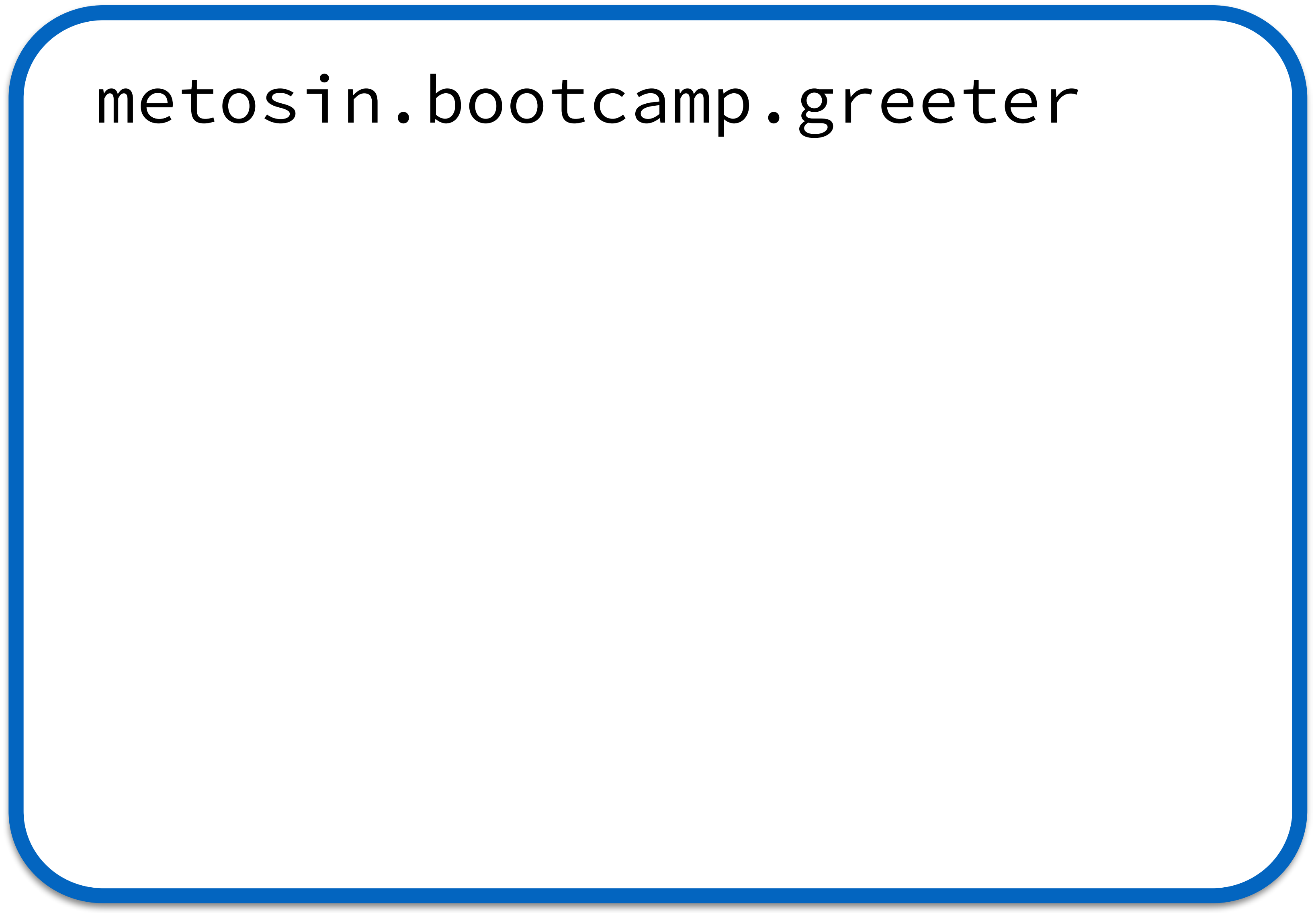
clojure basics

namespaces

```clojure
(ns metosin.bootcamp.greeter)

(def message "Hello")

(defn greet [your-name]
  (println message your-name))
```



metosin.bootcamp.greeter

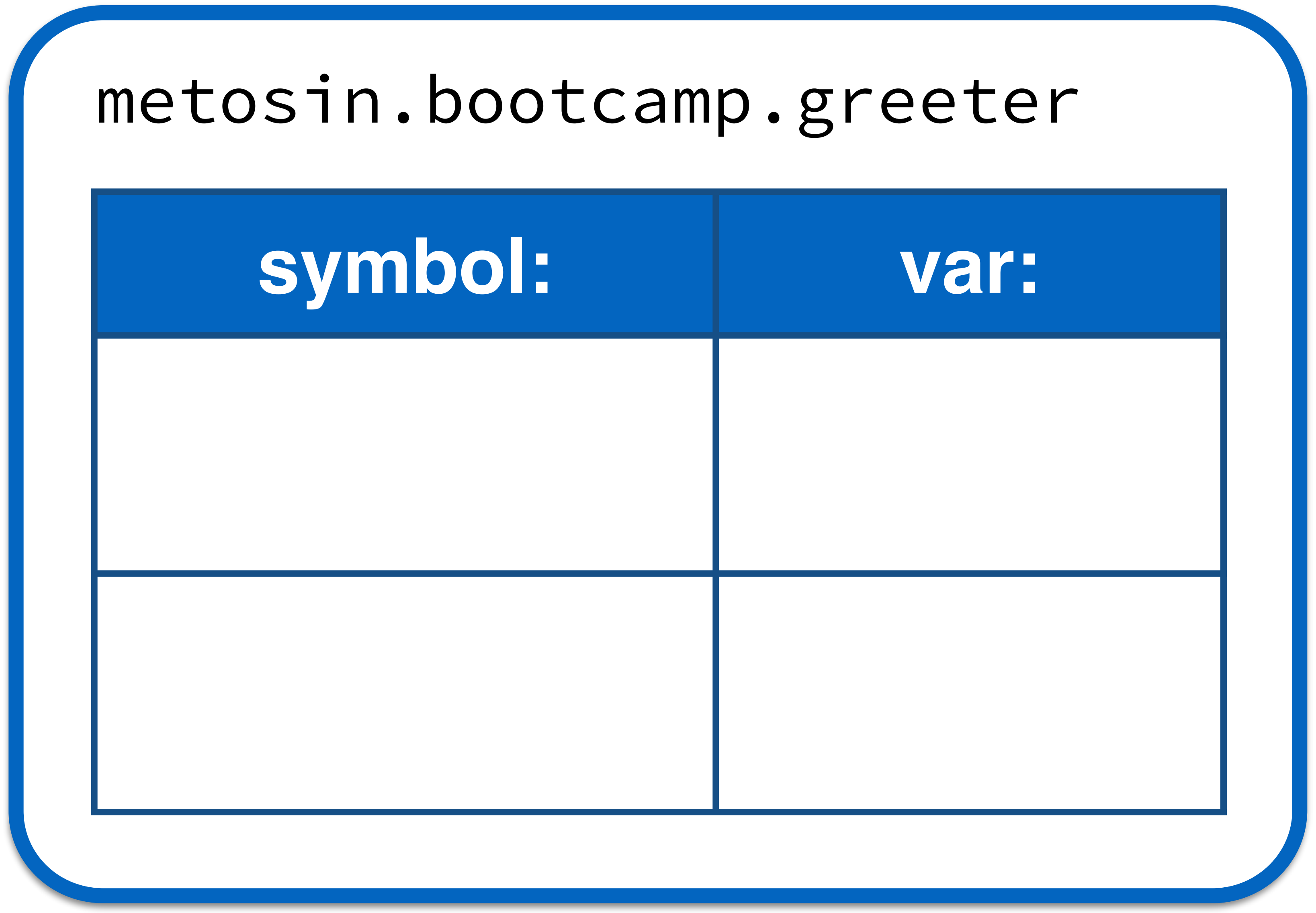
clojure basics

namespaces

```clojure
(ns metosin.bootcamp.greeter)

(def message "Hello")

(defn greet [your-name]
  (println message your-name))
```


metosin.bootcamp.greeter

symbol:	var:

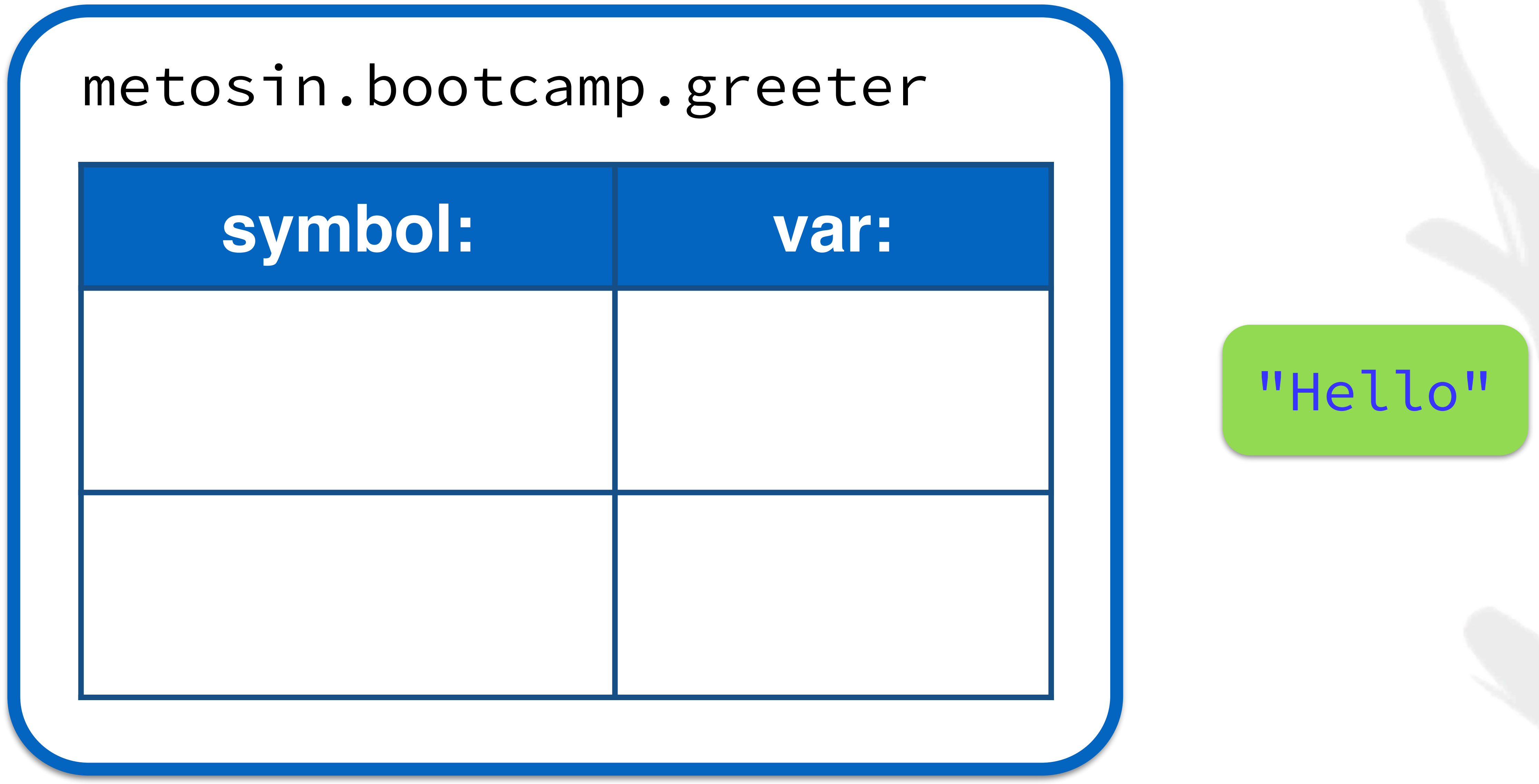
clojure basics

namespaces

```
(ns metosin.bootcamp.greeter)

(def message "Hello")

(defn greet [your-name]
  (println message your-name))
```



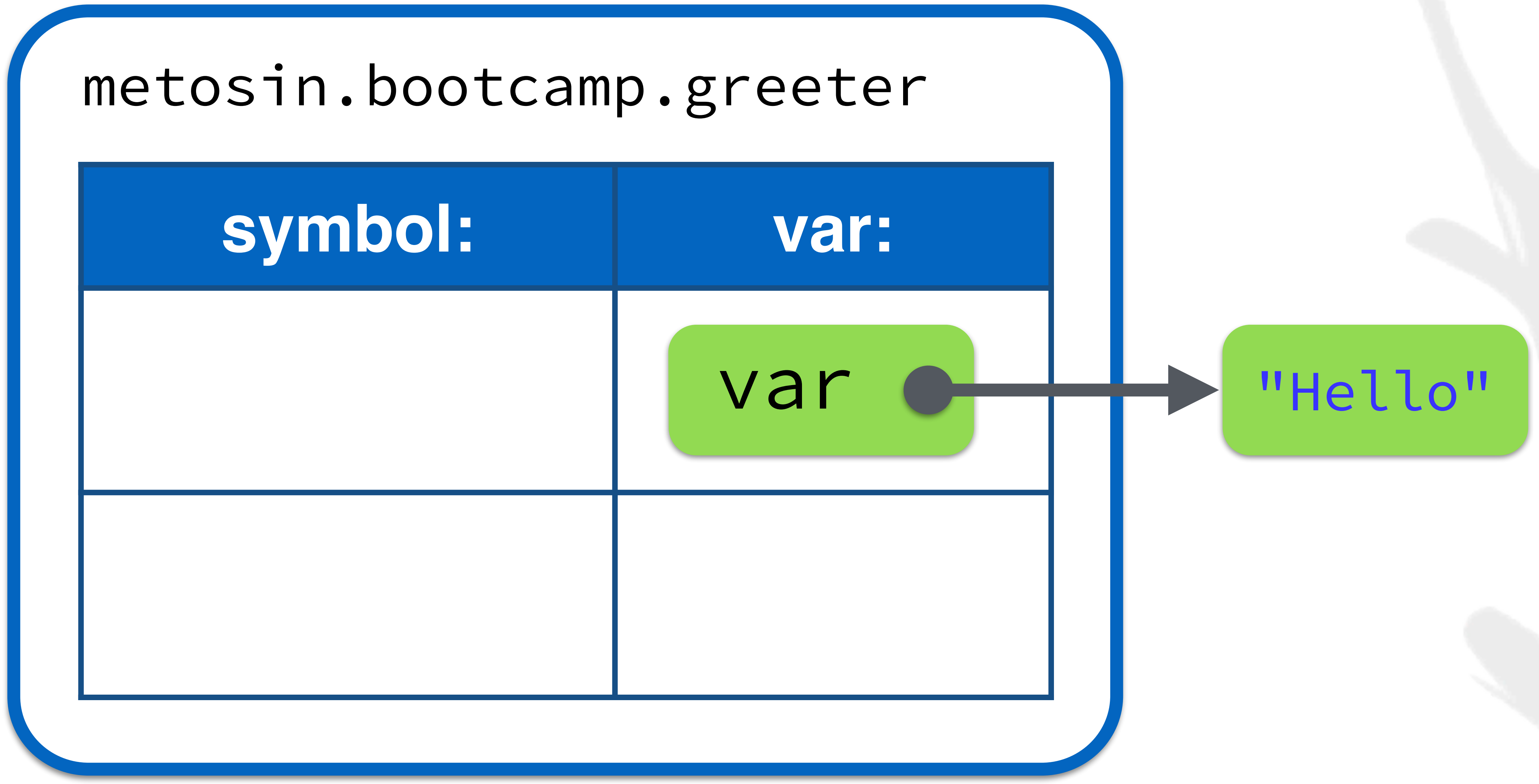
clojure basics

namespaces

```clojure
(ns metosin.bootcamp.greeter)

(def message "Hello")

(defn greet [your-name]
  (println message your-name))
```



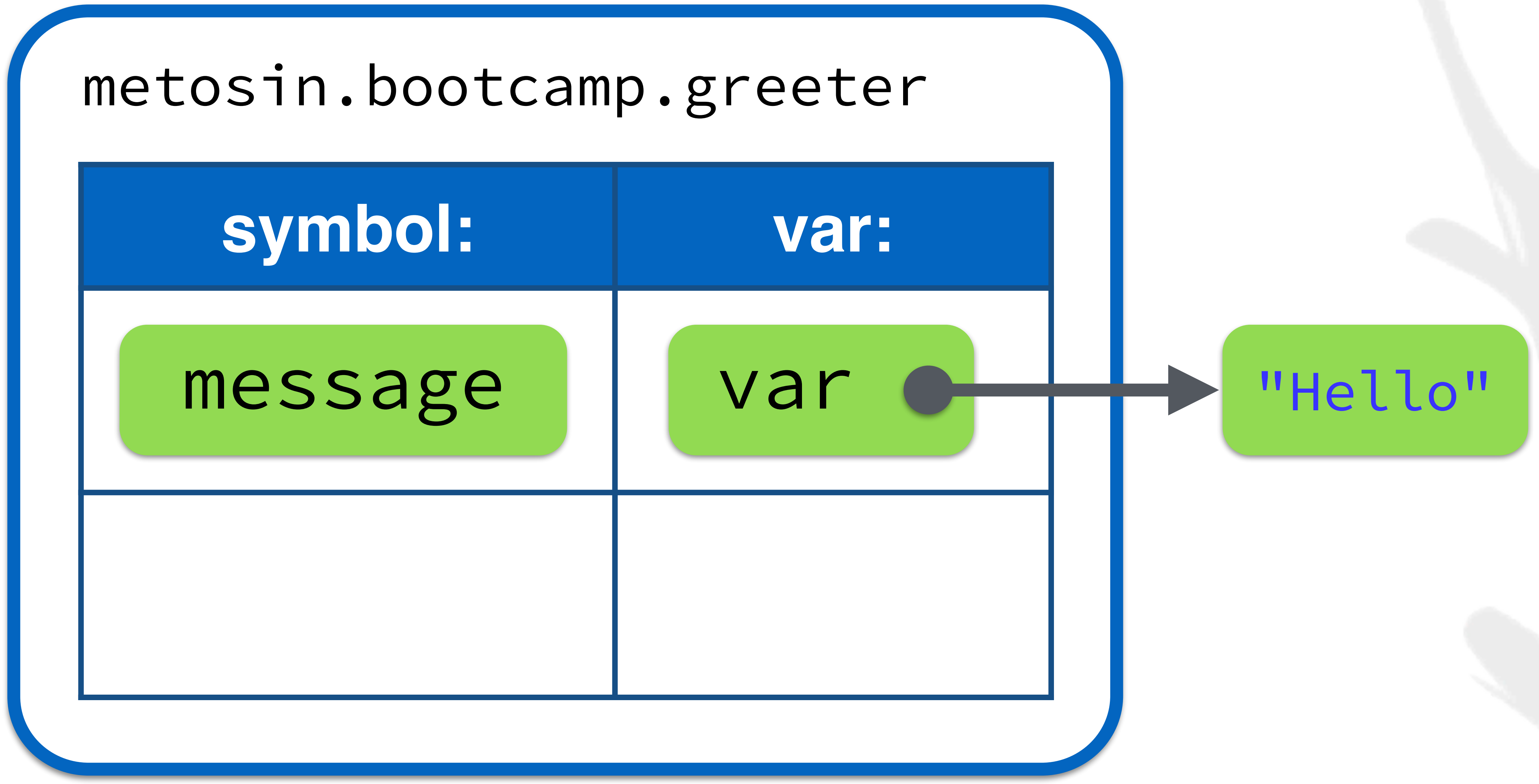
clojure basics

namespaces

```clojure
(ns metosin.bootcamp.greeter)

(def message "Hello")

(defn greet [your-name]
  (println message your-name))
```



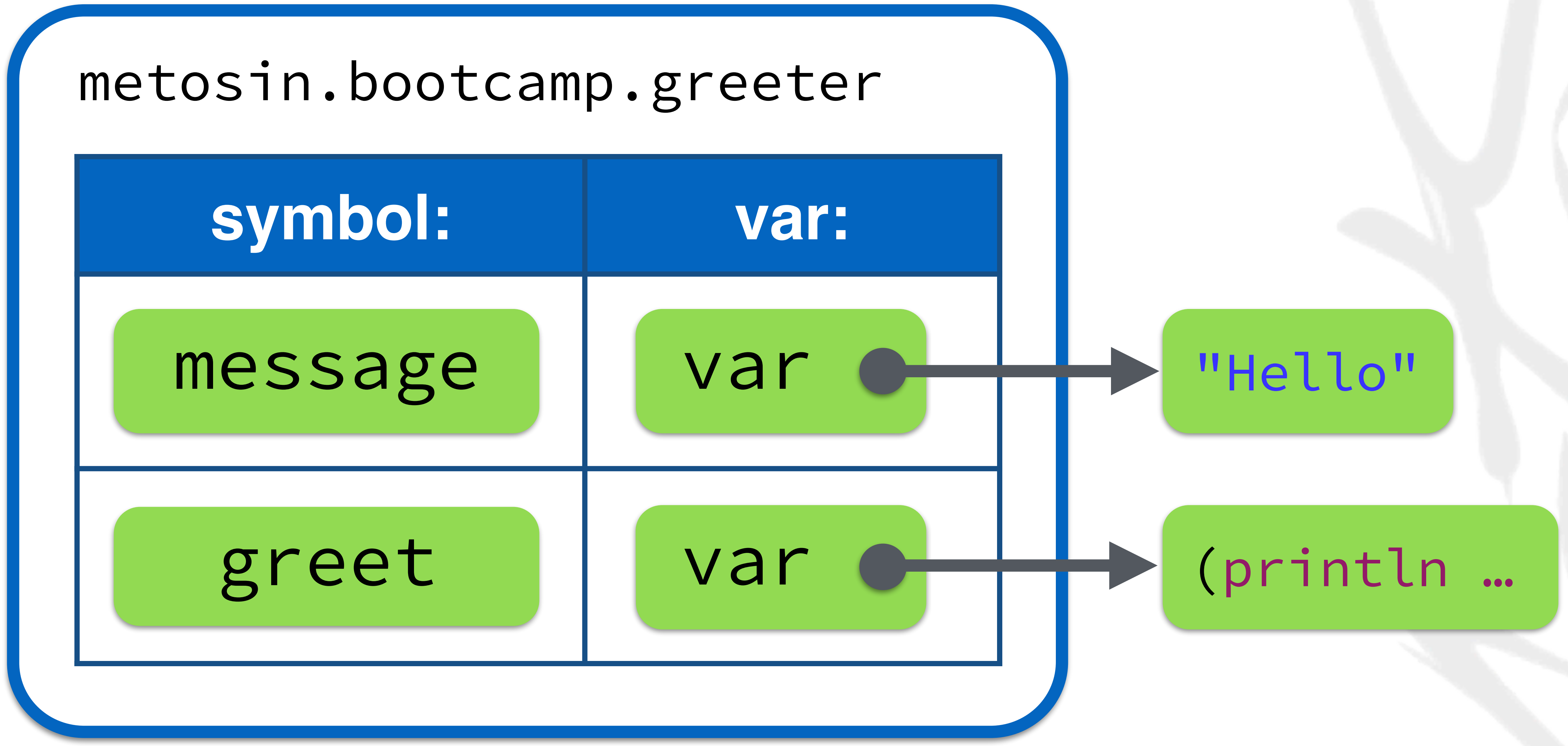
clojure basics

namespaces

```clojure
(ns metosin.bootcamp.greeter)

(def message "Hello")

(defn greet [your-name]
  (println message your-name))
```



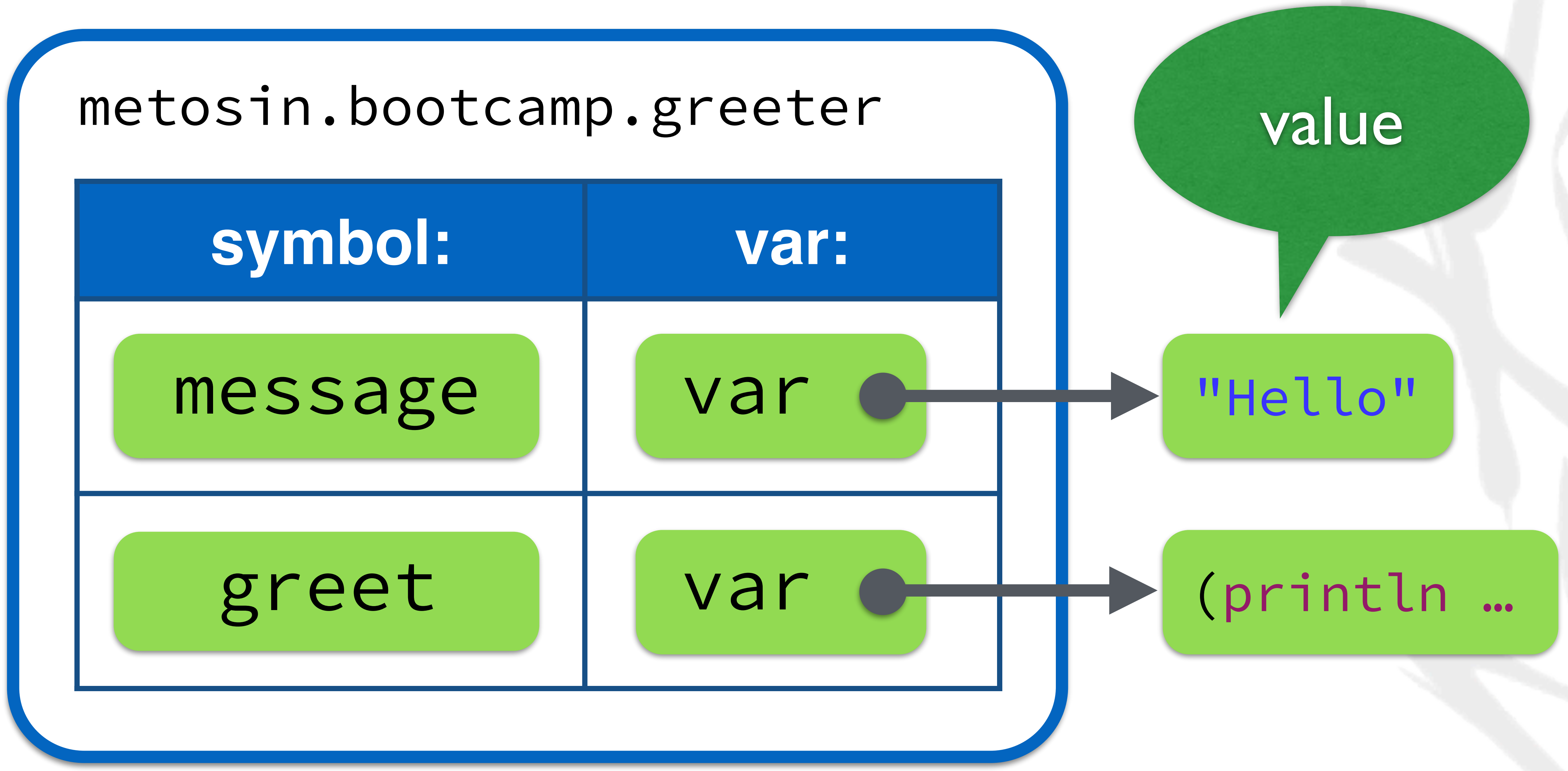
clojure basics

namespaces

```clojure
(ns metosin.bootcamp.greeter)

(def message "Hello")

(defn greet [your-name]
  (println message your-name))
```



clojure basics

namespaces

clojure basics

namespaces

in a namespace far far away…

```
(require 'metosin.bootcamp.greeter)
(metosin.bootcamp.greeter/greet "Jarppe")
```

clojure basics

namespaces

in a namespace far far away…

```
(require (quote metosin.bootcamp.greeter))
(metosin.bootcamp.greeter/greet "Jarppe")
```

clojure basics

namespaces

in a namespace far far away…

```
(require 'metosin.bootcamp.greeter)
(metosin.bootcamp.greeter/greet "Jarppe")
```

clojure basics

namespaces

in a namespace far far away…

```clojure
(require 'metosin.bootcamp.greeter)
(metosin.bootcamp.greeter/greet "Jarppe")
```

```clojure
(require '[metosin.bootcamp.greeter :as g])
(g/greet "Jarppe")
```

clojure basics

namespaces

in a namespace far far away…

```clojure
(require 'metosin.bootcamp.greeter)
(metosin.bootcamp.greeter/greet "Jarppe")
```

```clojure
(require '[metosin.bootcamp.greeter :as g])
(g/greet "Jarppe")
```

```clojure
(require '[metosin.bootcamp.greeter :refer [greet]])
(greet "Jarppe")
```

clojure basics

namespaces

in a namespace far far away…

```
(require 'metosin.bootcamp.greeter)
(metosin.bootcamp.greeter/greet "Jarppe")
```

```
(require '[metosin.bootcamp.greeter :as g])
(g/greet "Jarppe")
```

```
(require '[metosin.bootcamp.greeter :refer [greet]])
(greet "Jarppe")
```

```
(require '[metosin.bootcamp.greeter :refer :all])
(greet "Jarppe")
```

clojure basics

namespaces

use is ~ depreciated:

```
(use '[metosin.bootcamp.greeter])
```

same as:

```
(require '[metosin.bootcamp.greeter :refer :all])
```

```
(use '[metosin.bootcamp.greeter :only [greet]])
```

same as:

```
(require '[metosin.bootcamp.greeter :refer [greet]])
```

clojure basics

namespaces

the ns macro

clojure basics

namespaces

the ns macro

```
(ns in.a.namespace.far.far.away)

(require '[metosin.bootcamp.greeter :as g])

(g/greet "Jarppe")
```

clojure basics

namespaces

the ns macro

```
(ns in.a.namespace.far.far.away
  (:require [metosin.bootcamp.greeter :as g]))


(g/greet "Jarppe")
```

clojure basics

namespaces

the ns macro

```
(ns in.a.namespace.far.far.away
  (:require [metosin.bootcamp.greeter :as g]
            [clojure.string :as s]))

(g/greet "Jarppe")
```

clojure basics

namespaces

the ns macro

```
(ns in.a.namespace.far.far.away
  (:require [metosin.bootcamp.greeter :as g]
            [clojure.string :as s]))

(g/greet (s/join ", " ["Jarppe" "Juho"]))
```

clojure basics

namespaces

the default namespace is `user`

clojure basics

namespaces

the default namespace is `user`

clojure basics

namespaces

the default namespace is user

```
~/swd/clojutre-workspace/clojure-bootcamp> lein repl
nREPL server started on port 59438 on host 127.0.0.1
REPL-y 0.3.0
Clojure 1.5.1
    Docs: (doc function-name-here)
          (find-doc "part-of-name-here")
  Source: (source function-name-here)
 Javadoc: (javadoc java-object-or-class-here)
    Exit: Control+D or (exit) or (quit)
 Results: Stored in vars *1, *2, *3, an exception in *e

user=>
```

clojure basics

working with data

clojure basics

working with data

list	`(1 2 3)`
vector	`["foo" "bar"]`
set	`#{"maybe" "yes" "no"}`
map	`{:lang "clojure" :year 2007}`

clojure basics

working with data

list	`(1 2 3)`
vector	`["foo" "bar"]`
set	`#{"maybe" "yes" "no"}`
map	`{:lang "clojure" :year 2007}`

seq

clojure basics

working with data

seq

clojure basics

working with data

seq

```
=> (seq '(1 2 3))
```

clojure basics

working with data

seq

```
=> (seq '(1 2 3))
(1 2 3)
```

clojure basics

working with data

seq

```
=> (seq '(1 2 3))
(1 2 3)
=> (seq [1 2 3])
```

clojure basics

working with data

seq

```
=> (seq '(1 2 3))
(1 2 3)
=> (seq [1 2 3])
(1 2 3)
```

clojure basics

working with data

seq

```
=> (seq '(1 2 3))
(1 2 3)
=> (seq [1 2 3])
(1 2 3)
=> (seq {:lang "clojure" :year 2007})
```

clojure basics

working with data

seq

```
=> (seq '(1 2 3))
(1 2 3)
=> (seq [1 2 3])
(1 2 3)
=> (seq {:lang "clojure" :year 2007})
([:lang "clojure"] [:year 2007])
```

clojure basics

working with data

seq

```
=> (seq '(1 2 3))
(1 2 3)
=> (seq [1 2 3])
(1 2 3)
=> (seq {:lang "clojure" :year 2007})
([:lang "clojure"] [:year 2007])
=> (seq "Hello")
```

clojure basics

working with data

seq

```
=> (seq '(1 2 3))
(1 2 3)
=> (seq [1 2 3])
(1 2 3)
=> (seq {:lang "clojure" :year 2007})
([:lang "clojure"] [:year 2007])
=> (seq "Hello")
(\H \e \l \l \o)
```

clojure basics

working with data

seq

```
=> (seq '(1 2 3))
(1 2 3)
=> (seq [1 2 3])
(1 2 3)
=> (seq {:lang "clojure" :year 2007})
([:lang "clojure"] [:year 2007])
=> (seq "Hello")
(\H \e \l \l \o)
=> (seq '())
```

clojure basics

working with data

seq

```
=> (seq '(1 2 3))
(1 2 3)
=> (seq [1 2 3])
(1 2 3)
=> (seq {:lang "clojure" :year 2007})
([:lang "clojure"] [:year 2007])
=> (seq "Hello")
(\H \e \l \l \o)
=> (seq '())
nil
```

clojure basics

working with data

seq

```
=> (seq '(1 2 3))
(1 2 3)
=> (seq [1 2 3])
(1 2 3)
=> (seq {:lang "clojure" :year 2007})
([:lang "clojure"] [:year 2007])
=> (seq "Hello")
(\H \e \l \l \o)
=> (seq '())
nil
=> (seq nil)
```

clojure basics

working with data

seq

```
=> (seq '(1 2 3))
(1 2 3)
=> (seq [1 2 3])
(1 2 3)
=> (seq {:lang "clojure" :year 2007})
([:lang "clojure"] [:year 2007])
=> (seq "Hello")
(\H \e \l \l \o)
=> (seq '())
nil
=> (seq nil)
nil
```

clojure basics

working with data

seq

clojure basics

working with data

seq

```
=> (first (seq "Hello"))
```

clojure basics

working with data

seq

```
=> (first (seq "Hello"))
\H
```

clojure basics

working with data

seq

```
=> (first (seq "Hello"))
\H
=> (rest (seq "Hello"))
```

clojure basics

working with data

seq

```
=> (first (seq "Hello"))
\H
=> (rest (seq "Hello"))
(\e \l \l \o)
```

clojure basics

working with data

seq

```
=> (first (seq "Hello"))
\H
=> (rest (seq "Hello"))
(\e \l \l \o)
=> (next (seq "Hello"))
```

clojure basics

working with data

seq

```
=> (first (seq "Hello"))
\H
=> (rest (seq "Hello"))
(\e \l \l \o)
=> (next (seq "Hello"))
(\e \l \l \o)
```

clojure basics

working with data

seq

```
=> (first (seq "Hello"))
\H
=> (rest (seq "Hello"))
(\e \l \l \o)
=> (next (seq "Hello"))
(\e \l \l \o)
=> (rest (seq [1]))
```

clojure basics

working with data

seq

```
=> (first (seq "Hello"))
\H
=> (rest (seq "Hello"))
(\e \l \l \o)
=> (next (seq "Hello"))
(\e \l \l \o)
=> (rest (seq [1]))
()
```

clojure basics

working with data

seq

```
=> (first (seq "Hello"))
\H
=> (rest (seq "Hello"))
(\e \l \l \o)
=> (next (seq "Hello"))
(\e \l \l \o)
=> (rest (seq [1]))
()
=> (next (seq [1]))
```

clojure basics

working with data

seq

```
=> (first (seq "Hello"))
\H
=> (rest (seq "Hello"))
(\e \l \l \o)
=> (next (seq "Hello"))
(\e \l \l \o)
=> (rest (seq [1]))
()
=> (next (seq [1]))
nil
```

clojure basics

working with data

seq

```
=> (first (seq "Hello"))
\H
=> (rest (seq "Hello"))
(\e \l \l \o)
=> (next (seq "Hello"))
(\e \l \l \o)
=> (rest (seq [1]))
()
=> (next (seq [1]))
nil
```

clojure basics

working with data

-> and ->>

clojure basics

working with data

`->` and `->>`

clojure basics

working with data

-> and ->>

```
=> (dissoc (assoc {:lang "clojure" :year 2007} :inventor "Rich") :year)
{:inventor "Rich", :lang "clojure"}
```

clojure basics

working with data

-> and ->>

```
=> (dissoc (assoc {:lang "clojure" :year 2007} :inventor "Rich") :year)
{:inventor "Rich", :lang "clojure"}


=> (-> {:lang "clojure" :year 2007}
       (assoc :inventor "Rich")
       (dissoc :year))
{:inventor "Rich", :lang "clojure"}
```

clojure basics

working with data

`->` and `->>`

clojure basics

working with data

-> and ->>

```
=> (map str (filter odd? (range 10)))
("1" "3" "5" "7" "9")
```

clojure basics

working with data

-> and ->>

```
=> (map str (filter odd? (range 10)))
("1" "3" "5" "7" "9")

=> (->> (range 10)
        (filter odd?)
        (map str))
("1" "3" "5" "7" "9")
```

clojure basics

working with data

REPL session

clojure basics

java interop

clojure basics

java interop

clojure basics

java interop

```
=> (new java.util.Date)
#inst "2014-05-30T15:28:09.650-00:00"
```

clojure basics

java interop

```
=> (new java.util.Date)
#inst "2014-05-30T15:28:09.650-00:00"
=> (import [java.util Date])
java.util.Date
=> (new Date)
#inst "2014-05-30T15:28:21.903-00:00"
```

clojure basics

java interop

```
=> (new java.util.Date)
#inst "2014-05-30T15:28:09.650-00:00"
=> (import [java.util Date])
java.util.Date
=> (new Date)
#inst "2014-05-30T15:28:21.903-00:00"
=> (new Date 114 5 4)
#inst "2014-06-03T21:00:00.000-00:00"
```

clojure basics

java interop

```
=> (new java.util.Date)
#inst "2014-05-30T15:28:09.650-00:00"
=> (import [java.util Date])
java.util.Date
=> (new Date)
#inst "2014-05-30T15:28:21.903-00:00"
=> (new Date 114 5 4)
#inst "2014-06-03T21:00:00.000-00:00"
=> (Date. 114 5 4)
#inst "2014-06-03T21:00:00.000-00:00"
```

clojure basics

java interop

```
=> (new java.util.Date)
#inst "2014-05-30T15:28:09.650-00:00"
=> (import [java.util Date])
java.util.Date
=> (new Date)
#inst "2014-05-30T15:28:21.903-00:00"
=> (new Date 114 5 4)
#inst "2014-06-03T21:00:00.000-00:00"
=> (Date. 114 5 4)
#inst "2014-06-03T21:00:00.000-00:00"
=> (import [java.text SimpleDateFormat])
java.text.SimpleDateFormat
```

clojure basics

java interop

```
=> (new java.util.Date)
#inst "2014-05-30T15:28:09.650-00:00"
=> (import [java.util Date])
java.util.Date
=> (new Date)
#inst "2014-05-30T15:28:21.903-00:00"
=> (new Date 114 5 4)
#inst "2014-06-03T21:00:00.000-00:00"
=> (Date. 114 5 4)
#inst "2014-06-03T21:00:00.000-00:00"
=> (import [java.text SimpleDateFormat])
java.text.SimpleDateFormat
=> (let [fmt (SimpleDateFormat. "yyyy-MM-dd")
         now (Date.)]
     (. fmt format now))
"2014-05-30"
```

clojure basics

java interop

```
=> (new java.util.Date)
#inst "2014-05-30T15:28:09.650-00:00"
=> (import [java.util Date])
java.util.Date
=> (new Date)
#inst "2014-05-30T15:28:21.903-00:00"
=> (new Date 114 5 4)
#inst "2014-06-03T21:00:00.000-00:00"
=> (Date. 114 5 4)
#inst "2014-06-03T21:00:00.000-00:00"
=> (import [java.text SimpleDateFormat])
java.text.SimpleDateFormat
=> (let [fmt (SimpleDateFormat. "yyyy-MM-dd")
         now (Date.)]
     (. fmt format now))
"2014-05-30"
=> (let [fmt (SimpleDateFormat. "yyyy-MM-dd")
         now (Date.)]
     (.format fmt now))
"2014-05-30"
```

clojure basics

java interop

reify

clojure basics

java interop

reify

create anonymous object that implements one or more interfaces

clojure basics

java interop

reify

clojure basics

java interop

reify

```
=> (import [java.io File FilenameFilter])
java.io.FilenameFilter
=> (let [dir (File. ".")
         fnf (reify FilenameFilter
                (accept [this dir name]
                  (if (re-find #"\.clj$" name) true false)))]
     (.list dir fnf))
#<String[] [Ljava.lang.String;@77c774c7>
```

clojure basics

java interop

reify

```
=> (import [java.io File FilenameFilter])
java.io.FilenameFilter
=> (let [dir (File. ".")
         fnf (reify FilenameFilter
                (accept [this dir name]
                  (if (re-find #"\.clj$" name) true false)))]
     (seq (.list dir fnf)))
("project.clj")
```

clojure basics

java interop

reify

Supports only Java interfaces (and clojure protocols)

If you need to extend classes, use `proxy`

clojure basics

java interop

proxy

clojure basics

java interop

proxy

```
(proxy [class-name interface1 interface2...] [ctor-args]
  (method [arg]
    (implementation)))
```

clojure basics

java interop

proxy

```
(proxy [class-name interface1 interface2...] [ctor-args]
  (method [arg]
    (implementation)))
```

this is available implicitely

clojure basics

java interop

proxy

```
(proxy [javax.servlet.http.HttpServlet] []
  (doGet [req resp]
    (.setStatus resp 404)))
```

clojure basics

leiningen

clojure basics

leiningen

clojure basics

leiningen

maven for clojure

clojure basics

leiningen

maven for clojure

uses same packaging and repos

clojure basics

leiningen

maven for clojure

uses same packaging and repos

no XML

clojure basics

leiningen

project.clj

```
(defproject hello-world "0.1.0-SNAPSHOT"
  :dependencies [[org.clojure/clojure "1.6.0"]
                 [joda-time/joda-time "2.1"]])
```

clojure basics

leiningen

download deps	`$ lein deps`
clean	`$ lein clean`
make JAR	`$ lein jar`
make JAR in profile prod	`$ lein with-profile prod jar`
same with two profiles	`$ lein with-profiles prod,debug jar`
multiple commands	`$ lein do clean, cljsbuild clean, uberjar`

clojure basics

testing

clojure basics

testing

clojure.test

clojure basics

testing

clojure.test

clojure basics

testing

clojure.test

```
=> (require '[clojure.test :refer [deftest is]])
nil
```

clojure basics

testing

clojure.test

```
=> (require '[clojure.test :refer [deftest is]])
nil

=> (is (= (+ 39 3) 42))
true
```

clojure basics

testing

clojure.test

```
=> (require '[clojure.test :refer [deftest is]])
nil

=> (is (= (+ 39 3) 42))
true

=> (is (= (+ 39 3) 1337))

FAIL in clojure.lang.PersistentList$EmptyList@1
expected: (= (+ 39 3) 1337)
  actual: (not (= 42 1337))
false
```

clojure basics

testing

clojure.test

clojure basics

testing

clojure.test

```
=> (deftest addition
     (is (= (+ 39 3) 42))
     (is (= (+ 1295 42) 1337)))
```

clojure basics

testing

clojure.test

```
=> (deftest addition
     (is (= (+ 39 3) 42))
     (is (= (+ 1295 42) 1337)))

=> (clojure.test/run-tests 'user)

Testing user

Ran 1 tests containing 2 assertions.
0 failures, 0 errors.
{:type :summary, :fail 0, :error 0, :pass 2, :test 1}
```

clojure basics

testing

midje

clojure basics

testing

midje

clojure basics

testing

midje

```
=> (require '[midje.sweet :refer :all])
nil
```

clojure basics

testing

midje

```
=> (require '[midje.sweet :refer :all])
nil

=> (facts
     (+ 39 3)     => 42
     (+ 1295 42) => 1337)
true
```

clojure basics

testing

midje

```
=> (require '[midje.sweet :refer :all])
nil

=> (facts
     (+ 39 3)     => 42
     (+ 1295 42) => 1337)
true

=> (facts
     (/ 42 0) => (throws ArithmeticException))
true
```

clojure basics

testing

midje

```
=> (require '[midje.sweet :refer :all])
nil

=> (facts
     (+ 39 3)      => 42
     (+ 1295 42) => 1337)
true

=> (facts
     (/ 42 0) => (throws ArithmeticException))
true

=> (facts (+ 39 3) => 1337)
FAIL at (form-init3563197658688881023.clj:1)
    Expected: 1337
      Actual: 42
false
```

METOSIN

WE MAKE SOFTWARE

clojure bootcamp