Primitiva datatyper:

- List, vektor ([1 2 3]), map (nyckel-värde par - {:a 12 :b 17 42 “hello”})
    - Nyckelvärde paret ger a till 12, b till 17, 42 till hello
    - :a är en nyckel. Inte en symbol. : betyder at a vill användas. inte a som symbol
    -

def evaluerar en symbol

- (def a hello)
    - a ger hello när den används
- (def m {:a 12 _b “hello” 42 “forty-two”})
    - Den första används alltid som funktionsnamn och resten är det sm det är bundet till
        - Därför man skriver + 1 2 så gör man plus, på 1 och 2.
            - + 1 2 3 översätts till 1+2+3. Den är inte en binär funktion. Sätts mellan allt.

Vektor

- Indexering
    - (def v [1 2 3 4])
    - (v 3)
        - ger 4
- Lägga till
    - (conj [1 2 3] 4)
        - Fungerar även för list
    - Lägger till sist på en vektor, men först på en list
        - Gör det som är snabbast tror jag

Set

- #{1 2 3 4}
    - Osorterad
- Sorted-set: (sortet-set 1 2 3 4)
    - Har en ordning

(m 42)

- Kolla upp 42 i m.
    - Tänk det som att göra m som funktion, och passera 42 in i den?

(m 17)

- Ger nil. Dvs null för java

(:a m)

- Ger 12
- Innan har man passerat det som en funktion.
- I detta fall är map en funktion av key
- Så man kan göra m på a.

{1 2 3 4 5 6}

- ger {1 2, 3 4, 5 6}
    - För komma är ett mellanslag i clojure

Note:

- Programmera med immutable values
- Publika def är alltid muttable. icke-idiomatiskt)

(assoc m “hello” “world)

- lägger till “hello” bindet till “world”
- Håller funktionen lokalt

(let [x 10 y 15] (+ x y))

- Ger 25
    - Bind x till 10, bind 15 till y.
    - Lokalt gr en lista av par, för att göa kod mer l’sbart

(def plusone (fn [x] (+ 1 x)))

- ger (plusone 17)

defn plusone’ [x](+ 1 x))

- defn används för att ta bort paranteser. ‘ är bar ett unikt tecken

(def plusone’’ #(+ 1 %))

- #(är en lambda funktion. Som fn ovan.

‘(+ 1 2)

- ger (+ 1 2)

(+ 1 (* 2 3))

- ger 7
- Läser utifrån in.
- Note att den alltid evaluerar innan funktionen kallar
- Den är inte lazy

*1

- ger senaste resultet
- *2 ger näst senaste
- kan gra (def s *1)

(ger m :c)

- ger nil
- (ger m :c “not-found”)
    - Ger “not-found”
        - Strängen är ett defaultvärde för om det r nil

(filter odd? #_ => v)

- ger (1 3)
- På funktionen filter, ger den predikatet odd och lambdafunktion? för vad den ska ge

(type *1)

- Typen på den senaste resultatet

Bra funktionen

- filter, map, last, first, reverse, repeat, take antal(), range 1 5

Partial funktioner

- funktion f (a,b) -> a+b
- (partial f 17) ger (b) -> 17+b)
- Så partial * 2, ger * som f, 2 som a, b som det som kommer näst?

Macro Expand är bra för att göra om något från vanlig kod till omvänd lisp kod

- Macroexpand ‘(->> f g h)
    - tror den gör h(g(f))

For loop

- (for [x [1 2 3]] x)
- för x i 1 2 3, ge x (print x)
- ger 2 3 4

(for [x (range 1 10) :when (odd? x)] x)

- tilldelar x till varje tal mellan 1 till 10 som är ojämt och ger x
- I [] deklareras parametrar, som kan filtreras med when. Och i andra gör det givna

(let [[a b & rest] v] [a b rest])

- a bind till första elmentet i v, b binds till andra, rest är en lista av resten av v

(if predikat “a” “b”)

- ger a om predikatet är sant, annars ger den b.