destructuring, maps, variadic arguments & named arguments

Pat says: this is not the most interesting Clojure session but it covers some useful stuff...
- cheers Pat.

shadowing values

```
(let [x 'cat
      y 'dog
      ]
  (println `(x= ~x y= ~y))
  (let [x 'rat
        y 'frog
        ]
    (println `(x= ~x y= ~y))
    )
  (println `(x= ~x y= ~y)))

> (user/x= cat user/y= dog)
> (user/x= rat user/y= frog)
> (user/x= cat user/y= dog)
> nil
```

destructuring

```
(let [ [a b c] '(cat dog frog) ]       ;; using a list
  (println `(a= ~a b= ~b c= ~c)))

> (user/a= cat user/b= dog user/c= frog)
> nil

(let [ [a b c] '[cat dog frog] ]       ;; using a vector
  (println `(a= ~a b= ~b c= ~c)))

> (user/a= cat user/b= dog user/c= frog)
> nil

(let [ [a [b c]] '[cat [dog frog]] ]    ;; also works with lists
  (println `(a= ~a b= ~b c= ~c)))

> (user/a= cat user/b= dog user/c= frog)
> nil

(let [ [a [b c]] '[cat dog frog] ]
  (println `(a= ~a b= ~b c= ~c)))

> error!
```

```
(let [ [a [b c]] '(cat [dog frog]) ]          ;; mixed data types are ok
  (println `(a= ~a b= ~b c= ~c)))

> (user/a= cat user/b= dog user/c= frog)
> nil

(let [ [a [b c]] '[cat (dog frog)] ]
  (println `(a= ~a b= ~b c= ~c)))

> (user/a= cat user/b= dog user/c= frog)
> nil
```

destructuring fn args

```
(defn baa [a [b c]]
  (+ a (* b c)))

user=> (baa 1 [2 4])
9
user=> (baa 1 '(3 6))
19
```

variadic fns

```
(defn foo
  ([a]   (inc a))
  ([a b] (+ a b))
  )

user=> (foo 5)
6
user=> (foo 5 11)
16
```

variable length argument lists

```
(defn tee [a b & body]
  (list 'a= a 'b= b 'body= body))

user=> (tee 1 2 3 4 5 6 7 8)
(a= 1 b= 2 body= (3 4 5 6 7 8))
user=> (tee 1 2 :moo 'cow :bark 'dog)
(a= 1 b= 2 body= (:moo cow :bark dog))
```

destructuring maps

```
(let [m {:a 5 :b 4}]
  (let [{y :b x :a} m]
    (- x y)))

> 1

(let [m {:a 5 :b 4}    ;; or within a single let
      {y :b x :a} m
      ]
    (- x y)))

> 1
```

using :keys and :or

```
(let [{:keys [a b c]} {:b 'boo :c 'cow}]
  (list a b c))

> (nil boo cow)

(let [{:keys [a b c] :or {a 'apple b 'banana c 'cherry}} {:b 'boo :c 'cow}]
  (list a b c))

> (apple boo cow)

(defn laa [& {:keys [fruit veg] :or {fruit 'apple veg 'sprouts}}]
  (list 'fruit= fruit, 'veg= veg))

user=> (laa :veg 'sorgum)
(fruit= apple veg= sorgum)
user=> (laa :veg 'sorgum :fruit 'durian)
(fruit= durian veg= sorgum)
user=> (laa :fruit 'durian)
(fruit= durian veg= sprouts)


(defn baa [fish & {:keys [fruit veg] :or {fruit 'apple veg 'sprouts}}]
  (list 'fish= fish, 'fruit= fruit, 'veg= veg))

user=> (baa 'cod :veg 'mooli)
(fish= cod fruit= apple veg= mooli)
user=> (baa 'cod :veg 'mooli :fruit 'mango)
(fish= cod fruit= mango veg= mooli)
user=> (baa 'cod :meat 'spam :veg 'mooli :fruit 'mango)
(fish= cod fruit= mango veg= mooli)
```