Clojure Basic Training - Module 5

Destructuring

A language in a language

- Works on sequences or associative structures
- Use a small syntax ([], &, {}, [keys]…)
- Available in let, fn, defn, loop…
- Dramatically cuts down sequence boilerplate
- Only positional, no conditions (Pattern Matching)

Sequential destructuring

```
(defn rate [t]
  (let [tick (first t)
        currency (first (last t))]
  (* tick currency)))

(defn display [m]
  (str (:value m) (:unit m))))
```

- Known positions of elements in vectors
- Subset of keys in maps for computation
- Typical: “first of”, “last of”, “rest of”, “all keys”…

Naming vector elements

```
(def v [42 "foo" 99.2 [5 12]])
$> #'user/v
(let [[x _ z] v]
  (+ x z))
$> 141.2
```

- [x _ z] forces v destructuring from first element
- _ is conventional for “don’t care”
- Rest of v is ignored
- Works on anything sequential: vec, lists, strings…

“Special” vector elements

```
(let [[x y :as all] v] (first all))
$> 42
(let [[x y & others] v] (last others))
$> [5 12]
(let [[x y & [third & [[x y]]]] v] y)
$> 12
```

- “& others” fetch the rest and name it locally
- “:as all” fetch the entire vector
- Destructuring can be further nested

Debugging

```
$> (destructure '[[x y & others] v])
$> [v2 v x (nth v2 0 nil) y (nth v2 1 nil)
others (nthnext v2 2)]
```

- Under the hood, (destructure) expands the form
- Not meant for direct use, just debugging

Associative destructuring

```
(def m {:a 5 :b 6
        :c [7 8 9]
        :d {:e 10 :f 11}
        "foo" 88
        42 false})
```

- Dealing with map arguments is often the case
- Any associative form, not just maps
- Records, vectors, strings, Java Arrays

Keys correspondence

Put in local binding “a”

Key “:a” from map m

```
(let [{a :a b :b} m]
  (+ a b))
```

- Access non existent key return nil
- Any type can be used as key, not just keywords
- The list can get long, there is syntax sugar

Examples

```
(def m {:a 1 :b 2 :c 3 :d 4})

(let [{c :a :as orig} m]
  (assoc orig :b c)) ;; retaining the original

(let [{k :unknown x :a :or {k 50}} m]
  (+ k x)) ;; :or {map-lookup-for-default}

(let [{:keys [:a :b :c]} m]
  (str a b c)) ;; repetition killer
```

References

- “Clojure Programming”, O’Reilly
- “Destructuring cheat sheet” http://www.john2x.com/blog/clojure-destructuring/