Course Introduction

Mattox Beckman

University of Illinois at Urbana-Champaign
Department of Computer Science

Objectives
You should be familiar with...

- the basic list operations,
- the basic vector operations,
- the basic hash-map operations,
- ISeq, and
- sets.

The purpose...

- **Clojure** in Real Life™ will use these built-in structures extensively.
- We will use them in this course sporadically.
- Your goal today: be introduced.
- Your goal eventually: be annoyed with languages that don’t include these.

Why they are special

- Most languages contain these already: as library calls.
  
  ```java
  Hashtable balance = new Hashtable();
  balance.put("Zara", new Double(3434.34));
  balance.put("Mahnaz", new Double(123.22));
  balance.put("Daisy", new Double(99.22));
  balance.put("Qadir", new Double(-19.08));
  ```

- Clojure has literal syntax to express these.
  
  ```clojure
  (def balance {
      "Zara" 3434.34, "Mahnaz" 123.22,
      "Daisy" 99.22, "Qadir" -19.08
  })
  ```
Creating Lists

- Create empty list with '()', or sometime nil.
- Create whole lists using list or use the literal form.

1 (list 1 2 3)
2 ;; => '(1 2 3)
3 '(1 2 3)
4 ;; => '(1 2 3)
5 (list (+ 1 2) (* 3 4))
6 ;; => (3 12)
- Add to lists using cons

1 (cons (* 2 3) '(1 3 6))
2 ;; => (6 1 3 6)

Accessing List Elements

- Get the first element with first (like car from other Lisps).
- Get the rest of the elements with rest.
- Get a specific element with nth.
- Is the list empty? Use empty?

1 (def x '(1 2 3))
2 (empty? x)
3 ;; => false
4 (first x)
5 ;; => 1
6 (rest x)
7 ;; => (2 3)
8 (nth x 2)
9 ;; => 3

Other things

- Lists are used frequently, so there are many operations for them.
- You will see map, some, filter, apply, and reduce a lot.

1 (some odd? x)
2 ;; => true
3 (apply + x)
4 ;; => 6
5 (filter odd? x)
6 ;; => (1 3)
7 (reduce * 1 x)
8 ;; => 6
9 (map inc x)
10 ;; => (2 3 4)

Creating Vectors

- Similar to arrays, but some major differences!
- Create them using the vector function.
- Convert another structure to a vector with vec.
- Use square brackets as literal syntax.

1 (vector 1 2 3)
2 ;; => [1 2 3]
3 (vector '(1 2 3))
4 ;; => [(1 2 3)]
5 (vec '(1 2 3))
6 ;; => [1 2 3]
7 [1 2 3]
8 ;; => [1 2 3]
Accessing Vector Parts

```
(def v [1 2 3 5 8]) ;; => #'user/v
(empty? v) ;; => false
(count v) ;; => 5
(v 4) ;; => 8
(conj v 2) ;; => [1 2 3 5 8 2]
```

Vector Operations

- The list operations will work on vectors.
- Use the vector-specific versions if you want to preserve “vectorness.”

```
(map inc v) ;; => (2 3 4 6 9)
(mapv inc v) ;; => [2 3 4 6 9]
(apply + v) ;; => 19
```

Sequences

- Many of CLOJURE’s data structures are instances of Sequence.
- Provides: first, rest, empty?, count, map, etc.
- Advantage: uniformity; Disadvantage: unwanted format changes.
- Usually a good trade.

```
(map inc v) ;; => (2 3 4 6 9)
(map inc s1) ;; => (2 3 4 5)
(for [x s1] (* x 2)) ;; => (2 4 6 8)
(for [x v] (* x 2)) ;; => (2 4 6 10 16)
```

Credits

- The Java hash table example is from the Tutorials Point web site.
- More examples can be found at http://www.tutorialspoint.com/java/java_hashtable_class.htm.
- Can you tell which operating system they used to host their site?