CS135 L05 — Lists & Composite Data

Your Name • 2025-09-25

CS135 L05 — Lists & Composite Data

Prepared by Your Name

L05.0 — List values & expressions

- The **empty list** is a value; it is not the same thing as "no list".
- In Racket, lists are built with `cons` (front insertion).
- Accessors: `first` (head), `rest` (tail). Both error on the empty list.
- Predicates: `empty?`, `list?`, `cons?`. :contentReference[oaicite:0]{index=0}

```
empty           ; the empty list value
(cons 'apple empty)
(cons 'eggs (cons 'apple empty))
(first (cons 'a (cons 'b empty))) ; => 'a
(rest (cons 'a (cons 'b empty)))  ; => (cons 'b empty)
(empty? empty) ; => true
```

Common list ops (Racket)

- `cons`: value × list ⟶ list
- `first`: non-empty list ⟶ first value
- `rest`: non-empty list ⟶ list without first
- `empty?`: any ⟶ Bool (true iff empty)
- `list?`, `cons?` as additional predicates. :contentReference[oaicite:1]{index=1}

L05.1 — Composite data with lists

- A 2D point `(x, y)` can be represented as `(cons x (cons y empty))`.
- Distance to origin uses the usual formula:

$$\sqrt{x^2 + y^2}$$

- Start with a clear **purpos** , **contrac** , and **test** . :contentReference[oaicite:2]{index=2}

```
;; distance-to-origin: (cons Num (cons Num empty)) -> Num
;; Purpose: consume a point (x,y) and produce distance to (0,0).
(define (distance-to-origin point)
  (sqrt (+
         (sqr (first point))             ; x
         (sqr (first (rest point))))))   ; y

(check-expect (distance-to-origin (cons -3 (cons 4 empty))) 5)
(check-within (distance-to-origin (cons 3 (cons -1 empty))) 3.1622 0.001)
(check-expect (distance-to-origin (cons 6 (cons 0 empty))) 6)
(check-expect (distance-to-origin (cons 0 (cons 0 empty))) 0)
```

L05.2 — Data definitions + helpers

- Use **data definition** to name composite types and simplify contracts.
- Example:

```
;; A Point is a (cons Num (cons Num empty))
;; mk-point: Num Num -> Point
(define (mk-point x y) (cons x (cons y empty)))
(define (get-x p) (first p))
(define (get-y p) (first (rest p)))
```

- With helpers, function bodies become clearer:

```
;; distance-to-origin: Point -> Num
(define (distance-to-origin p)
  (sqrt (+ (sqr (get-x p)) (sqr (get-y p)))))
```

- Symbol “enums” via data definitions, e.g.:

```
;; an Outerwear is (anyof 'jacket 'sweater 'shirt)
;; what-to-wear: Num -> Outerwear
(define (what-to-wear t)
  (cond [(< t 8)  'jacket]
        [(< t 16) 'sweater]
        [else      'shirt]))
```

:contentReference[oaicite:3]{index=3}

L05.3 — Cards as composite data

- Suits as symbols; Ranks as numbers or symbols; Card is two-element list.

```
;; A Suit is (anyof 'spade 'heart 'diamond 'club)
;; A Rank is (anyof 2 3 4 5 6 7 8 9 10 'jack 'queen 'king 'ace)
;; A Card is (cons Suit (cons Rank empty))
(define (mk-card suit rank) (cons suit (cons rank empty)))
(define (get-suit c) (first c))
(define (get-rank c) (first (rest c)))
```

- Equality across mixed kinds (rank comparison):

```
;; rank=?: Rank Rank -> Bool
(define (rank=? r0 r1)
  (or (and (symbol? r0) (symbol? r1) (symbol=? r0 r1))
      (and (number? r0) (number? r1) (= r0 r1))))
```

- Face-card predicate:

```
;; face-card?: Card -> Bool
(define (face-card? c)
  (or (rank=? (get-rank c) 'jack)
      (rank=? (get-rank c) 'queen)
      (rank=? (get-rank c) 'king)))
```

:contentReference[oaicite:4]{index=4}

L05 — You should know

- List primitives (`cons`, `first`, `rest`, `empty`, `empty?`, `list?`, `cons?`).
- How to write **data definition** and **helper** for composite types.
- How to follow the **design patter** (purpose, contract, tests, body). :contentReference[oaicite:5]{index=5}

L05 — Allowed constructs (highlights)

New: `Any cons cons? empty empty? first list? rest` Previously allowed: arithmetic/logic, `cond`, `define`, `check-expect`, `check-within`, numeric & symbol predicates, etc. (See full slide list.)
Recursion must follow first-version rules. :contentReference[oaicite:6]{index=6}