# CS 360
# Programming Languages
# Day 3





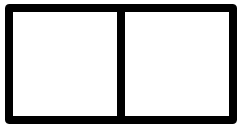
Swift





# *Review*

- Cons cell: two-piece structure (like a 2-member class in Java)



  - Also called a pair. left side called "car"; right side called "cdr"
  - **`(cons e1 e2)`** constructs a new cons cell (and returns it)
  - **`(car e)`** returns the car part of ***e***; **`(cdr e)`** returns the cdr of ***e***

- **`'(v1 . v2)`** constructs a "literal" cons cell.
- Drawing cons cells:
  - **`(cons 1 2)`**
  - **`(cons 1 (cons 2 3))`**
  - **`(cons (cons 1 2) 3)`**

# *Box-and-pointer notation with lists*

- Key to differentiating pairs from lists: lists never have dots in them.
- `'(1 . 2)` versus `'(1 2)`

- How would you create `'(1 . 2)` with call(s) to cons?

- How would you create `'(1 2)` with call(s) to cons?

- What does `(cons 1 '(2 3))` create?

- What does `(cons '(1) '(2 3))` create?

# *Review*

Huge progress in two lectures on the core pieces of Racket:

- Variables
  - **`(define variable expression)`**
- Functions
  - Build: **`(define (f x1 x2 …) e)`**
  - Use: **`(f e1 … en)`**
- Pairs
  - Build: **`(cons e1 e2)`** OR **`'(v1 . v2)`**
  - Use: **`(car e)`**, **`(cdr e)`**
- Lists
  - Build: **`'()`** **`(cons e1 e2)`** OR **`'(v1 v2 v3 …)`**
    **`(list e1 e2 …)`** **`(append e1 e2 …)`**
  - Use: **`(null? e)`** **`(car e)`** **`(cdr e)`**

# *The* ***cond*** *expression*

We have two "if-then-else" expressions in Racket:

- **`(if test e1 e2)`**
  - evaluates to **`e1`** if test is **`#t`**, otherwise evaluates to **`e2`**.

- ```
  (cond (test1 e1)
        (test2 e2)
        ...
        (#t en))
  ```

  - evaluates to **`e1`** if **`test1`** is **`#t`**
  - evaluates to **`e2`** if **`test2`** is **`#t`**
  - (etc)
  - evaluates to **`en`** if all prior tests are **`#f`**
  - The last **`#t`** clause is optional, but is useful as an "else".
  ```

# *Processing nested lists*

```
(define (length lst)
  (if (null? lst) 0
    (+ 1 (length (cdr lst)))))
```

```
(define (length-nested lst)
  (cond ((null? lst) 0)
        ((list? (car lst))
          (+ (length-nested (car lst))
             (length-nested (cdr lst))))
        (#t (+ 1 (length-nested (cdr lst))))))
```

# *Other useful functions and reminders*

- **`(and e1 e2...)`**
- **`(or e1 e2...)`**
- **`(not expr)`**
  - e.g., **`(not (= a b))`**
- **`(remainder x y)`**
  - returns remainder of **`x`** divided by **`y`**
- Remember the differences between **`cons`**, **`list`**, and **`append`**:
- **`(cons item lst)`**
  - makes a new list with **`item`** as the first element, and the items in **`lst`** as the rest of the list.
- **`(list a b c...)`**
  - makes a new list of (**`a b c...`**)
- **`(append lst1 lst2...)`**
  - makes a new list of the items inside of **`lst1`**, then the items inside of **`lst2`**...