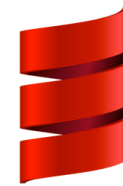
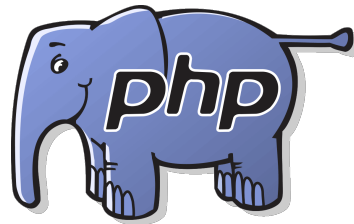


# CS 360

## Programming Languages

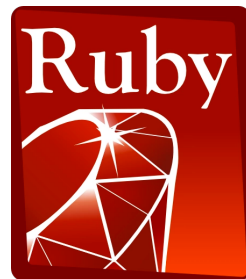
### Lecture 3



**Scala**

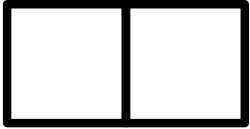


Swift



# Review

- Cons cell: two-piece structure (like a 2-member struct in C++)



- 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)`

# *Lists*

- Lists are built in Racket using linked lists of cons cells.

Need ways to *build* lists and *access* the pieces...

# *Building Lists*

- The empty list is a value: `' ()`
- In general, a list of values is a value; elements are separated by spaces:  
`' (v1 v2 ... vn)`
- If `e1` evaluates to `v1` and `e2` evaluates to a list `(v2 v3 ... vn)`, then `(cons e1 e2)` evaluates to `(v v1 v2 v3 ... vn)`
  - Key to remember: If `e2` is a list, then `cons` makes a new list with `e1` at the front.

# *Accessing Lists*

- `(null? e)` evaluates to `#t` if and only if `e` evaluates to `'()`.
- If `e` evaluates to `(v1 v2 ... vn)` then `(car e)` evaluates to `v1`
  - throw exception if `e` evaluates to `'()`
  - Think of `car` as "get the first element of the list."
- If `e` evaluates to `(v1 v2 ... vn)` then `(cdr e)` evaluates to `(v2 ... vn)`
  - throw exception if `e` evaluates to `'()`
  - Think of `cdr` as "get everything but the first element of the list."
  - Notice result is a list

## *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?

## *Two other ways to build lists*

- **list** function
  - Makes a list out of all arguments.
  - Arguments can be of any data type.
  - **(list e1 e2 ... en)** evaluates **e1** through **en** to values **v1** through **vn**; returns the list **'(v1 v2 ... vn)**.
- **append** function
  - Concatenates values inside lists given as arguments.
  - Arguments *must* be lists.
  - **(append e1 e2 ... en)** evaluates **e1** through **en** to values **v1** through **vn**;
  - If **v1 = (v11 v12 ... )** and **v2 = (v21 v22 ... )** etc, then return value is **(v11 v12 ... v21 v22 ... )**.

# *Exercises*