CS61A

SCHEME

LOGISTICS AND REMINDERS

- Lab 10: Scheme **due Today**
- HW07 : Scheme **due Tomorrow**
- Regrade requests for midterm are **due Monday**
- Advising OH this week + feel free to contact me

SCHEME

- Intro to many scheme things

SCHEME DATATYPES

- Everything in Scheme is about evaluating expressions so let’s start with basic ones
- Primitives:
  - Numbers (ex. 1, 3.14 , 6.28)
  - Booleans (ex. #t #f)
  - Symbols (ex. foo, potato, hello, hi2, any reasonably alphanumeric)
  - Strings (ex. “foo”, “potato”, “hello”, “hi2”, any reasonable alphanumeric)
  - undefined(like None in python undefined is its own thing)

BOOLEAN CONTEXTS

- Like Python in boolean contexts we can use any datatype
- Unlike Python the only falsey value is #f
  - Implication: (if 0 1 2) returns 1
  - Note: (if <condition> <true_consequence> <false_consequence>)

STRINGS VS SYMBOLS

- STRINGS ARE NOT THE SAME AS SYMBOLS
- STRINGS are always declared with double quotes(ex. “hello”)
- Symbols are things that can be evaluated, but do not automatically evaluate
- Symbols are created using the **quote** special form
    - Ex. (quote hello) gives the symbol hello
    - Will talk about quote special form in more detail later but the above is the primary way to create symbols

UNDEFINED

- How do these pop up?
- scm> undefined
- scm> (print 1)
- Very rare, never need to work with these, only mentioning for completeness

COMPLEX EXPRESSIONS

- (<operator> <operands>) or (<special_form> <operands>)
- What are special forms we have seen before in Python?
  - and, or, if, def etc?
  - Why is **and** a special form?
- What are scheme special forms?
  - **define**, **if**, **cond**, **and**, **or**, let, begin, **lambda**, **quote** , quasiquote, unquote, some other irrelevant ones
- **DANGER:** Function calls and special forms look the same, but don't act the same?
- Can you make an example illustrating this?

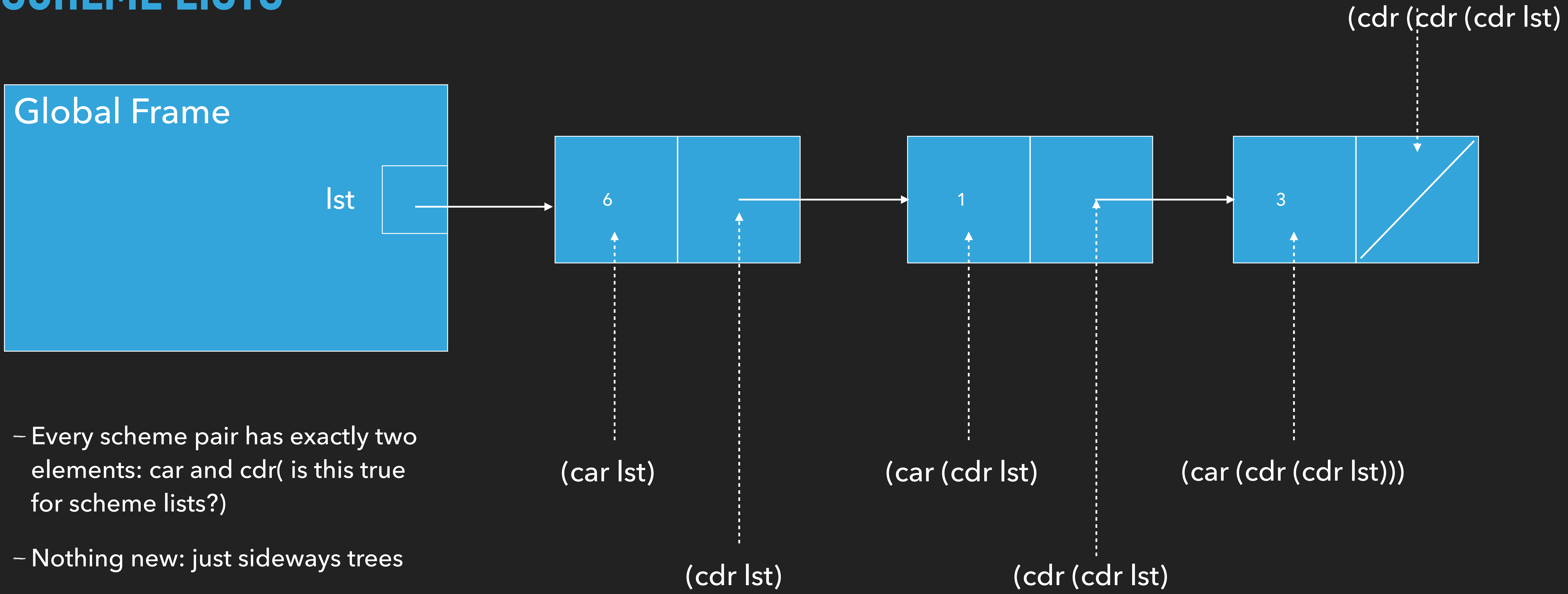
BUILT IN OPERATORS

- (+ 1 0)
- (modulo 5 2)
- (= 2 2)
- (eq? 2 2)
- (equal? 2 2)
- (/ 10 2)
- (quotient 5 2)
- What’s the difference between / and quotient?
- Difference between = , eq?, equal?
    - = is the same as python == but exclusively for numbers
    - eq? Is the same is python **is**
    - equal? Is like python == for things besides numbers(example think about == for lists in Python)

COMPLICATED DATATYPES

- Functions or Procedures as we call them
- Pairs(exactly like linked lists)
- Lists -> Pairs + nil
  - Same as Linked Lists + Link.empty(nil is scheme equivalent)
  - Created by cons procedure
    - Ex. (cons 1 (cons 2 nil))
  - Access via car, cdr procedures
  - lnk is Link.empty is equivalent to (null? lnk)

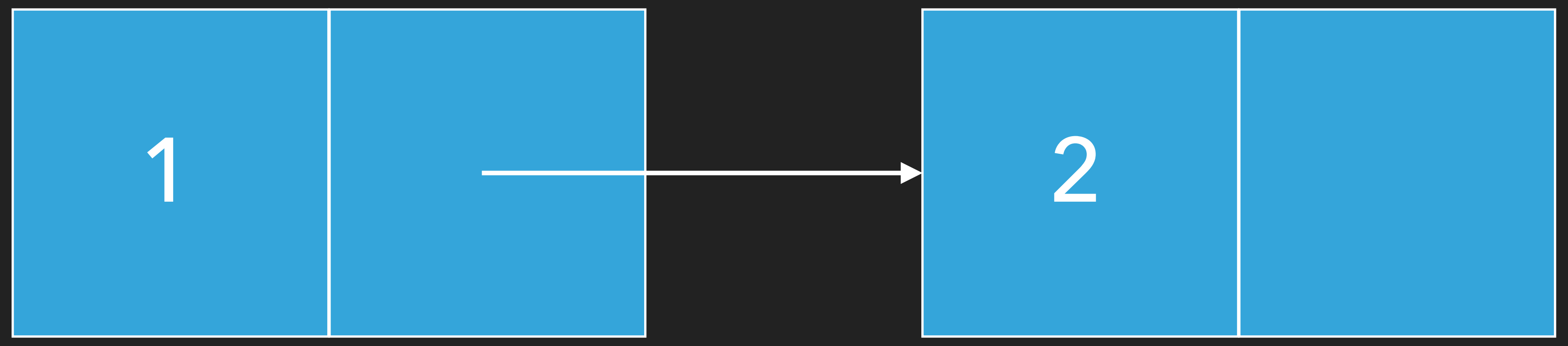
SCHEME LISTS



- Every scheme pair has exactly two elements: car and cdr(is this true for scheme lists?)
- Nothing new: just sideways trees

REPRESENTATION

- Three ways to represent scheme lists
  1. Code - (cons 1 (cons 2 nil))
  2. Box and Pointer Diagram -
  3. Parenthetical Form (1 2)



QUOTE SPECIAL FORM

- (quote <primitive>)
  - (quote 1) gives number 1
  - (quote #t) gives boolean #t
  - (quote “hi”) gives string “hi”
  - quote evaluates atoms to their value
- (quote <alphanumeric>)
  - (quote hello) gives symbol hello
  - (quote potato) gives symbol potato
  - quote in this contexts gives symbols!
- (quote <parenthetical representation of lst>)
- (quote (1 2 3))
- (quote (a 1 2))
- (quote (cons 1 2))
- quote creates box-and-pointer corresponding to this representation and all atoms are as if quote was called on them

QUOTE SPECIAL FORM

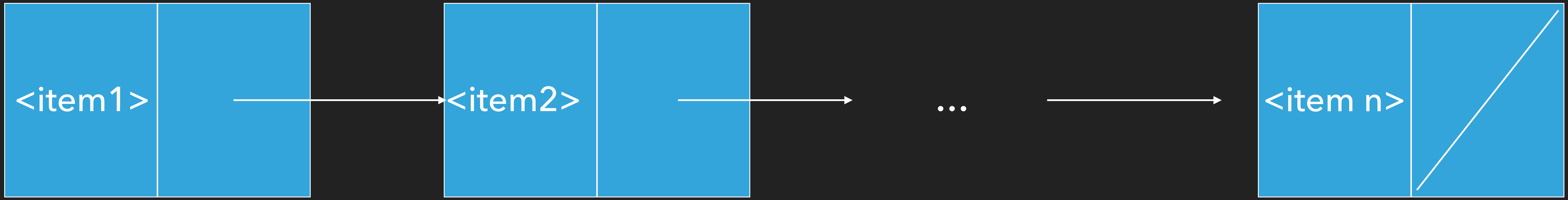
- Makes scheme lists super fast!
  - (quote (1 2 3 4)) makes the same thing as (cons 1 (cons 2 (cons 3 (cons 4 nil))))
- Be careful and always check datatypes for understanding!

3 WAYS TO MAKE LISTS

- cons procedure
- quote special form
- list procedure

LIST RULES

- (list <item1> <item2> <item 3> … <item n>)
- Creates a pair for every item then chains them together



LAMBDA FORM

- (lambda (<args>) <body>)
- Ex. (lambda (x) (* x x))
- These are expressions what do they return?

DEFINE FORM

- (define a <expression>)
  - ex. (define a 5) (define potato (+2 1))
- (define (<procedure name> <procedure args>) <body>)
  - ex. (define (foo x) (* x x))
- define is an expression what does it return?
- Is this different than python?

IF FORM

- (if <condition> <true_consequence> <false_consequence>)
  - false consequence is optional
- What does (if 0 1 2) return?
- What does (if nil 3) return?
- What does (if #f 0) return?

COMMENTS

- Notice no iteration,
    - Recursion is the heart of scheme problem solving
- Special forms and functions look the same!
    - (and #f (/ 1 0)) vs (+ 1 (/1 0)) don't have the same order of operations. Why?
- #f is only falsey value
- Symbols don't have a nice Python analogue
- When in doubt trust your Python gut
    - Example. (define (foo x) (lambda (y) (+ x y))
        - Do you understand why this is an example?
    - Would define work in the above instance? Why or Why not?