# CS 230
# Programming Languages

10 / 25 / 2022

Instructor:  Michael Eckmann

# Today's Topics

- Questions? / Comments?
- Functional Programming
- More Scheme

# Functional Programming Languages

- Remind us of the features of functional languages

# Scheme

**`CAR`** takes a list parameter; returns the first **element** of that list

e.g., (CAR '(A B C)) yields A

(CAR '((A B) C D)) yields (A B)

Notice: CAR can evaluate to an **atom** or a **list**.

**`CDR`** takes a list parameter; returns the **list** after removing its first element

e.g., (CDR '(A B C)) yields (B C)

(CDR '((A B) C D)) yields (C D)

(CDR '(A B)) yields (B)

(CDR '(A)) yields ()

Notice: CDR always evaluates to a **list**.

# Scheme

- Scheme

**CONS** takes two parameters, the first of which can be either an **atom** or a **list** and the second of which is a **list**; returns a new **list** that includes the first parameter as its first element and the second parameter as the remainder of its result

e.g., (CONS 'A '(B C)) returns (A B C)

(CONS '(A B) '(B C)) returns ((A B) B C)

**LIST** - takes any number of parameters; returns a list with the parameters as elements

# Scheme

- Scheme
  - DEFINE – used to create programmer-defined functions / or bind names to values of expressions
  - When names are bound to values of expressions they are NOT variables, instead they are named constants.
  - Either two atoms as parms, or two lists.
  - Example when two atoms are given (creates a named constant):

(DEFINE games_in_season 162)

# Scheme

- Scheme
  - DEFINE – used to create programmer-defined functions / or bind names to values of expressions
  - The form when two lists are given
    - binds the expressions collectively as a function to a function name and its parameters:

(DEFINE (func_name parameters)
expression {expression}
)

  - Examples on next slide

# Scheme

- Scheme
  - Examples:

```
(DEFINE (square x) (* x x))


(DEFINE (hypotenuse side1 side1)
    (SQRT (+ (square side1)
    (square side2)))
    )
```

What would these look like as imperative language functions?

# Functional Programming

- Scheme
  - Numeric “predicate” functions return a Boolean
  - #T or #F
  - =, <>, >, <, >=, <=, EVEN?, ODD?, ZERO?
  - Empty list () is considered #F
  - Any non-empty list is considered #T

  - Symbolic Atoms and Lists “predicate” functions
  - EQ?, LIST?, NULL?, EQUAL?

# Functional Programming

- Scheme
  - There is a difference between
    - =
    - eq?
    - equal?
  - = is used for numeric comparison
  - eq? is used for symbol/atom comparison
  - equal? is used for symbol/atom or list comparison
- There's more to it than what I say above ---- I'll photocopy some examples and their results for eq?, equal? and eqv? for your edification.

# Functional Programming

- Scheme
  - =, <>, >, <, >=, <=, EVEN?, ODD?, ZERO?
  - EQ?, LIST?, NULL?, EQUAL?

  - Examples:

(= n 0)

(NULL? somelist)

(EQ? 'A (CAR somelist))

(LIST? '() )

(EQUAL? '(a b z) '(a b z))

# Functional Programming

- Scheme
  - and, or, not
  - Examples:

(or (= n 0) (= n 1))

(not (= n 0))

(and (eq? 'a (car lis)) (= y 1))

# Functional Programming

- Scheme
  - input / output
  - (read)
  - (display "Hello World")
  - (newline)
  - (write "Hello World")
  - (define x 5)
  - (write x)

# Functional Programming

- Scheme
  - IF takes three parms

(IF predicate then_expression else_expression )

  - COND – is like switch / case statements

(COND
( predicate1 expression { expression } )
( predicate2 expression { expression } )
...
( predicaten expression { expression } )
( ELSE expression { expression } )
)

# Functional Programming

- COND
  - Predicates are evaluated in order until one is #T
  - Then the expressions that follow that first #T predicate are evaluated and the value of the last one evaluated is the returned value.
  - If no predicate is #T, then COND returns (), the empty list.
  - ELSE at the end acts like a default if none of the other predicates are true it will evaluate that one.
  - Can you see any differences or similarities to switch statements in Java?

# Functional Programming

- COND
  - Can you see any differences or similarities to switch statements in Java?
  - In COND, there are implied “breaks” after each predicate (case.)

# Scheme

- Recursion
  - To do any repetition, purely functional languages require recursion as there are no loops/iteration in purely functional languages.
  - Scheme does have imperative features, but we will prefer to use the functional features only.

# Functional Programming

- Example of a function containing an IF
- Here, we are defining a function named factorial which takes one parameter n:

```
(DEFINE (factorial n)
        (IF  (= n 0)
             1
             (* n (factorial (- n 1)))
        )
)
```