CS 230
Programming Languages

11 / 03 / 2022

Instructor: Michael Eckmann

Today's Topics

- Questions? / Comments?
- Handout on eq? eqv? equal?
- More Scheme
- Introduce the parser homework

Functional Programming

- Scheme
  - Please remind us of map and apply

Map

- map
  - a function that takes two parameters which are a function and a list
  - map applies the function to each element of the list and returns a list of the returned values
  - the following returns (#t #t #t #f #f #f #f #t)
  - (map number? '(1 2 3 a b r tttt 4.5))

  - another example:
  - (map floor '(1 2 3 4 4.5 4.6 7.8))

Apply

- apply
  - a function that takes two parameters which are a function and a list
  - it applies the function to all elements of the list and returns whatever the function would have returned if called using those elements of this list as individual parameters.

  - only functions that take a collection of arguments, not as a list, individually (like + and <) can be passed to apply

  - (apply < '(3 4 5 1))   ; ok, because (< 3 4 5 1) is a valid call.
  - (apply + '(4 5 6 7))   ; ok, because (+ 4 5 6 7) is a valid call.

How to write count-if

;; count-if is supposed to take a function and a list as parameters and count how many trues would result when the function is applied to each element of the list.

```
(define (count-if  fun  lis)


        (count-t (map fun lis))


    )
```

;; this will return 4 because 4 of the list elements are numbers

```
(count-if  number?  '(1 2 3 a b r tttt 4.5))
```

How can we use count-if to ...

```
;; write a function named every which takes in a function and a list as parameters and will
    return #t if the result of the function applied to each element is true
(define (every  fun  lis)


;; what goes here?
   )


;; write a function named any which takes in a function and a list as parameters and will
    return #t if the result of the function applied to at least one element is true
(define (any  fun  lis)


;; what goes here?
   )
```

How can we use count-if to ...

;; write a function named every which takes in a function and a list as parameters and will return #t if the result of the function applied to each element is true

```
(define (every  fun  lis)


(=  (length lis)  (count-if fun lis) )
    )
```

;; write a function named any which takes in a function and a list as parameters and will return #t if the result of the function applied to at least one element is true

```
(define (any  fun  lis)


(>= (count-if fun lis) 1 )
    )
```

optional parameters

- A function that takes optional parameters is specified in this way:

```
(define (fun reqp1 reqp2 . optparms)
    ; reqp1 is required
    ; reqp2 is required
    ; optparms will be a list of all the rest of the arguments passed in
)
; if you want a function with only optional parameters:
(define (fun2 . optparms)
    ; optparms will be a list of all the arguments passed in
)
```

How about writing avg ...

```
;; write a function named avg which takes in any number of numbers as parameters and
     averages them.
;; how will I write the parameters when defining the function?
;; how will we handle the parameters inside the function?
;; can we use anything we just learned to allow us to write the code inside the function?


(define (avg          )



   )
```

How about writing avg ...

;; write a function named avg which takes in any number of numbers as parameters and averages them.

;; how will I write the parameters when defining the function?

;; how will we handle the parameters inside the function?

;; can we use anything we just learned to allow us to write the code inside the function?

```
(define (avg   .   nums  )


(cond
((null? nums) 0) ; returns 0 as the average of no nums
(else (/ (apply + nums) (length nums))))


   )
```

Examples of functions as parameters

- So, we just witnessed several ways in which we can use functions as parameters.
- This is interesting since passing functions as parameters is not common in imperative languages.
- Hopefully you get the sense of how useful this feature is.

To do a series of statements in order (compound statement)

- Many constructs, like if expect a specific number of parameters.
- If expects exactly three parameters.
- Suppose you wanted to do more than one thing in the true or false part of the if?
- Example
- (if ____ _____ ____)

Example use of begin

```
(define (testbegin parm)
 (if   (> parm 0)                                    ; the condition parameter of the if
      (begin
        (display "Enter a number")
        (let ((num (read)))
            (display "You entered ")
            (display num)    (newline)  )
       )                                              ; ends the true portion parameter of if
      (begin
        (display "you must not have wanted to enter anything")
        (newline)
      )                                               ; ends the false portion parameter of if
 )
)
```

Let's write ...

- a function that makes a list from input (until a sentinel is entered, say q for quit).
- a function sumOddsUpTo which will take one parameter which is the upper limit. The function should return the sum of all the odds from 1 to that upper limit. Assume the upper limit is >= 1.
- lengthDeep --- which, given a list, will count up the number of elements even within sublists.
- thirdElement that returns the 3rd element of the given list or 0 if fewer than 3 elements.