CMPSC 461: Programming Language Concepts, Spring 2024
Assignment 5 Practice Notes Packet

Prof. Suman Saha

October 14, 2025

Problem 1: Racket Basics : Knowledge check

What do the following expressions evaluate to:

1. (* 2 (+ 4 5))
2. (= 3 (+ 1 3))
3. (car '(elmer fudd daffy duck))
4. (cdr '(elmer fudd daffy duck))
5. (and (= 1 2) (= 5 (/ 5 1)))

Solution

1. 18 (4 + 5 evaluates to 9, 9 * 2 = 18)
2. #f (1 + 3 evaluates to 4 and 3 is not equal to 4)
3. elmer (first element)
4. (fudd daffy duck) (Everything except the first element)
5. #f (1 is not equal to 2 hence f and 5 is equal to 5 hence t therefore f & t = f)

Problem 2: Racket Basics : Knowledge check

1. What does the following expression evaluate to:

```
(let (( x (+ 2 8))
         (y 100)) (/ y x) )
```

2. What does the following expression evaluate to:

```
(let (( x 100) (y 5) )
     (let ((x 1))
          (+ x y) ) )
```

Solution

1. We are given the following expression:

`(let ((x (+ 2 8)) (y 100)) (/ y x))`

- First, we evaluate the expression `(+ 2 8)` to get:

$$x = 10$$

- Next, we have `y = 100`. - The body of the `let` expression is `(/ y x)`, which computes:

$$\frac{100}{10} = 10$$

Thus, the expression evaluates to:

$$\boxed{10}$$

2. We are given the following expression:

`(let ((x 100) (y 5)) (let ((x 1)) (+ x y)))`

- The outer `let` binds `x = 100` and `y = 5`. - The inner `let` rebinds `x = 1` within its scope.
- The body of the inner `let` is `(+ x y)`, where:

$$x = 1 \quad \text{and} \quad y = 5$$

So, we compute:

$$1 + 5 = 6$$

Thus, the expression evaluates to:

$$\boxed{6}$$

Problem 3: Racket programs: Recursion

Define a recursive program **remove-at** which removes Kth element from a list and returns the remaining elements of the list in order. Additionally, when K is negative the goal is to remove they Kth element from the right. The function should work as follows:

```
    (remove-at '() 3) returns '()
    (remove-at '(a b c d) 2) returns '(a c d)
    (remove-at '(a b c d) -2) returns '(a b d)
    (remove-at '(a b c d) 0) returns '(a b c d)
```

Solution A possible solution is as follows:

```
(define (remove-at lst k)
   (if (< k 0)
       ; If k is negative, reverse the list and adjust k to be positive
       (reverse (remove-at (reverse lst) (- k)))
       ; If the list is empty, return an empty list
       (cond ((null? lst) '())
             ;  If k is zero, return the list unchanged
             ((zero? k) lst)
             ; If k is 1, remove the first element and return the rest
             ((= k 1) (cdr lst))
             ; Otherwise, recursively process the rest of the list
             (else
              (cons (car lst) (remove-at (cdr lst) (- k 1)))))))
```

Problem 4: Map practice : Deviation

"Squared deviations from mean" (or SDM) is a metric commonly used in statistical analysis, and is defined as the squared difference of the value from the mean of a given list. Implement a function "sdm" to calculate the squared deviations from mean for each value in a list, and return the answer as a list of SDMs. Use the **map** function to accomplish the same. You can use the *average* function such that "(average lst)" would return the average value of the list of numbers *lst*. The function should work as follows :

```
(sdm '(1 3 2 4 5)) returns '(4 0 1 1 4)
(sdm '(5 5 5 5 5)) returns '(0 0 0 0 0)
```

Solution A possible solution is as below :

```
(define (average lst)
  (cond ((null? lst) 0)
        (else (/ (foldl + 0 lst) (length lst)))))

(define (sdm lst)
    (let ((avg (average lst)))
        (map (lambda (x) (expt (- x avg) 2)) lst)))
```

1. Compute the average of the list
2. Use `map` to apply the following to each element in `lst`.
3. Calculate $(x - \text{avg})^2$ for each element x.
4. Apply this operation to every element in the list `lst`.

Problem 5: Foldl practice : New Filter

Consider the following scheme programs:

```
(define (foo l) (map (lambda (x) (/ x 2)) l))

(define (goo f l)
    (cond ((null? l) '())
        (else (let ((result (goo f (cdr l))))
            (append (f (car l)) result)))))
```

What will be the output of the following code and describe what does this function return in one sentence:

1. (foo '(2 4 6 8))
2. (goo (lambda (x) (list x x)) '(1 4 9 16))?

Solution

1. Final Output; (1 2 3 4)
   In general, this function returns a list where each element is divided by 2.
2. Final Output; (1 1 4 4 9 9 16 16)
   In general, this function returns a list where elements are the concatenation of the results of applying f to elements in l, in the same order.

Problem 6: Map practice : Addition

Using *map*, define a function *addN* that takes a list of numbers and a constant "N" as input and returns the list with N added at each element. Using a lambda function can help with the same.

Solution One possible solution is :

```scheme
(define (addN alist n)
  ; Use map to apply the lambda to each element
  (map (lambda (x) (+ n x)) alist))
```

Problem 7: Building Up 1

Using any way you like, define a function called *sum_cubes* that takes in a list of numbers and output the summation of each item cubed. If the list is empty, return 0. Assume that the input list would only be numbers and no other input that would make the function dysfunctional. Assume all cases and you may create helper function.

```
(sum_cubes '()) ; returns 0
(sum_cubes '(2)) ; returns 8
(sum_cubes '(2 4 3)) ; returns 99
; Note this is not all possible cases
```

Solution

```
(define (cube x) (* (* x x) x))

(define (sum_cubes lst)
  (cond ((null? lst) 0) ; Base case: if the list is empty, return 0
        (else (+ (cube (car lst)) ; Cube the first element of the list
                    #| Recursively call sum_cubes on remaining list |#
                 (sum_cubes (cdr lst)))))) ;
```

Problem 8: Building Up 2

Extending from the previous question for *sum_cubes*. The result and score is not ideal. Plans for a curve is in the making. Define a function or functions *curve* with *foldl* and *length* function to determine the curve for the assignment. The curve is relevant to the class average. If the average is greater than 70, no curve. If the average is equal to 70, no curve. If the average is lower than 70, the curve is equal to 70 minus the class average. You can safely assume all scores are between 0 to 100. Return 0 when scores are empty.

```
(curve '()) ; returns 0
(curve '(61)) ; returns 9
(curve '(70 70 70)) ; returns 0
(curve '(46 61 47 73)) ; returns 13.25
; Note this is not all possible cases
```

Solution

```
(define (average lst)
  (cond ((null? lst) 0)
        (else (/ (foldl + 0 lst) (length lst)))))

(define (curve lst)
  (cond ((null? lst) 0)
        ((= (average lst) 0) 0)
        ((> (average lst) 70) 0)
        (else (- 70 (average lst)))))
```

1. The average function computes the average of a list of numbers.
2. If the list is empty (null? lst), it returns 0.
3. Otherwise, it uses foldl + 0 lst to sum all elements in the list and divides by the length of the list using (length lst) to compute the average.
4. The curve function calculates the appropriate curve based on the class average.
5. If the list is empty (null? lst), the result is 0.
6. If the average is 70, no curve is applied, and the result is 0.
7. If the average is greater than 70, no curve is applied, and the result is 0.
8. If the average is less than 70, the curve is calculated as *70 - average*.

Problem 9: Building Up 3

After the *curve* function is complete, it is in time to apply the curve. But nobody wants to manually add the curve. Define a function *apply* to the take in a list of scores, and add the curve to each score in the list, then output the list. Use the *map* function and feel free to use your curve function previous defined. If scores are empty, return 0. Assume all scores are from 0 to 100.

```
(apply '()) ; returns '()
(apply '(61)) ; returns '(70)
(apply '(70 70 70)) ; returns '(70 70 70)
(apply '(46 61 47 73)) ; returns '(59.25 74.25 60.25 86.25)
; Note this is not all possible cases
```

Solution

```
(define (apply students)
  ; compute the curve score add add it to each student using map
  (let ((c (curve students))) (map (lambda (x) (+ x c)) students)))
```

Problem 10: Building Up 4

Define a function `discount`, which takes the name of an organization that someone belongs to and produces the discount (as a factor) that the person should receive on a purchase. Members of BBB get 25%, members of ACM or IEEE get 35%, and members of UPE get 20%. All other organizations get no discount.

Sample Input	Sample Output
"IEEE"	0.35
"UPE"	0.20

Solution

```
(define (discount organization)
  (cond
    [(equal? organization "BBB") 0.25]   ; BBB members get 25% discount
    #| ACM and IEEE members get 35% discount |#
    [(or (equal? organization "ACM") (equal? organization "IEEE")) 0.35]
    [(equal? organization "UPE") 0.20]   ; UPE members get 20% discount
    [else 0.0]))  ; All other organizations get no discount
```