"Good" Functional Programming

- How to indent and make your program readable
- How to document your program
- How to prepare test cases for your program

Indentation and Readability

A **BAD** example

```
(define (a n1 n2) (cond (
        (null?  n1) n2 (else (cons (car n1)
        (a (cdr n1) n2)))))
```

A **GOOD** example

```
(define (append l1 l2)
    (cond
        ((null?  l1) l2)
        (else (cons (car l1) (append (cdr l1) l2)))
    )
)
```


How to Document Your Program

On your specification you should state:

- What is the function for, i.e., how does it process the arguments to produce the output
- What are the auxiliary functions for
- What are the assumptions

How to Document Your Program (cnt.)

```
; Purpose: compute the number of elements in the top-most
; level of list x. For example: (length '(1 (2 3))) = 2
; Assumptions: none
; Auxiliary functions called: none

(define (length x)
   (cond
      ((null? x) 0)
      (else (+ 1 (length (cdr x))))
   )
)

; Purpose: same as above
; Assumptions: x consists of at least one element
; Auxiliary functions called: none

(define (length x)
   (+ 1 (length (cdr x)))
   )
)
```

How to Document Your Program (cnt.)

```
; Purpose: auxiliary function for reverse.
; While going through list x, accumulate results in Result.
; Motivated by: (reverse (x1 x2 x3)) = (cons x3 (reverse (x1 x2)))
; Assumptions: Result is a list
; Auxiliary functions called: none

(define (rev x Result)
    (cond
/* when the list is finished, Result contains the answer */
        ((null? x) Result)
/* build Result by adding the first element of x to the */
/* beginning of Result. Thus, Result ends up containing*/
/* elements of x in reverse order */
        (else (rev (cdr x) (cons (car x) Result)))
    )
)

(define (reverse x)
    (rev x '())
)
```


How to Document Your Program (cnt.)

```
; Purpose: This function verifies if an element is in a list
; Assumptions: (is-in? 'a '(b (a))) => #f, but
;                  (is-in? '(a) '(b (a))) =>#t
; Auxiliary functions called: none

(define (is-in? elem lis)
    (cond
/* If the list is empty, then return false*/
        ((null? lis) #f)
/* If the element and the head of the list are equal, then return true*/
        ((eq? elem (car lis)) #t)
/* Otherwise, continue analyzing the rest of the list*/
        (else (is-in? elem (cdr lis)))
    )
)
```


Preparing Test Cases for Your Program

- Determine the data coverage of your program (function)
- Determine test cases for:
  - trivial cases, e.g., stop conditions for recursions;
  - more elaborate cases, e.g, those which are “almost” stop conditions;
  - complex cases, e.g., nested lists

Examples of Tests Cases

```
(define (length x)
   (cond
      ((null? x) 0)
      (else (+ 1 (length (cdr x))))))
```

Simple cases: $\begin{cases} (length\ '()) \\ (length\ '(a)) \end{cases}$

More elaborate case: $(length\ '(a\ b\ c))$

Complex cases: $\begin{cases} (length\ '((e\ d)\ a\ b)) \\ (length\ '(a\ (b\ c\ (d\ e))\ ()\ (g\ e))) \end{cases}$

Examples of Tests Cases (cnt.)

```
(define (is-in? elem lis)
    (cond
        ((null? lis) #f)
        ((eq? elem (car lis)) #t)
        (else (is-in? elem (cdr lis)))
    )
)
```

Simple cases: $\begin{cases} (is-in?\ 'a\ '()) \\ (is-in?\ 'a\ '(a)) \\ (is-in?\ '(a)\ '(a)) \end{cases}$

More elaborate cases: $\begin{cases} (is-in?\ 'a\ '(b\ c\ a)) \\ (is-in?\ '(a)\ '(a\ b\ (a))) \end{cases}$

Complex cases: $\begin{cases} (is-in?\ 'a\ '(b\ (a\ b))\ (a)) \\ (is-in?\ 'a\ '((a)\ (b\ c\ a)\ a)) \end{cases}$

food for slide eater

food for slide eater

food for slide eater