CPSC 481 — AI —Project 2b GP Arithmetic Support Fcns #2

```
;; Illustrations of how some GP Framework things could be done.

;; TOC
```

```
;; ---------------------------------------------------- get-front-upto-nth ----
(defun get-front-upto-nth ( rn rlist )
  "Return list head from 0-th thru N-th elt."
  (let ((elt-n (nth rn rlist)))
    (reverse (member elt-n (reverse rlist)))))
;; Tests
;; (setq my-list '((1 a) (2 b) (3 c) (4 d) (5 e) (6 f) (7 g)))
;; (get-front-from-nth 4 my-list)
;; ((1 a) (2 b) (3 c) (4 d) (5 e))
;; (get-front-from-nth 2 my-list)
;; ((1 a) (2 b) (3 c))

;; ------------------------------------------------------------ get-score ----
(defun get-score (rcritter)
  "Get score for critter.  Dummy: return its length."
  (length rcritter))
;; Tests
;; (get-score '(+ 3 4))
;; 3

;; ------------------------------------------------------------ score-pop ----
(defun score-pop ( rpop )
  "Create Pop-Scored pairs (Score Critter) given Pop list of critters."
  (mapcar #'(lambda (critter)
              (let ((score (get-score critter)))
                (list score critter)))
          rpop))
;; Tests
;; (setq my-pop '((a b c)
;;                (a)
;;                (e f g)
;;                (a d)))
;; ((a b c) (a) (e f g) (a d))
;; (setq my-pop-scored (score-pop my-pop))
;; ((3 (a b c)) (1 (a)) (3 (e f g)) (2 (a d)))
```

CPSC 481 — AI —Project 2b GP Arithmetic Support Fcns #2

```lisp
;; ----------------------------------------------- safe-sort-scored-pop ----
(defun safe-sort-scored-pop ( rscored-pop )
  "Return a sorted list of scored-critter elts.  Don't change given list.
   NB, your Lisp's built-in sort fcn may damage the incoming list."
  (let ((sacrifice-list (copy-list rscored-pop)))
    (sort sacrifice-list
          #'(lambda (scored-critter-1 scored-critter-2)
              (< (car scored-critter-1) (car scored-critter-2))))))
;; Tests
;; my-pop-scored
;; ((3 (a b c)) (1 (a)) (3 (e f g)) (2 (a d)))
;; (safe-sort-scored-pop my-pop-scored)
;; ((1 (a)) (2 (a d)) (3 (a b c)) (3 (e f g)))
;; my-pop-scored
;; ((3 (a b c)) (1 (a)) (3 (e f g)) (2 (a d)))

;; ------------------------------------------------ get-pop-from-scored ----
(defun get-pop-from-scored (rscored-pop)
  "Return just the Pop of critters from the Scored Pop."
  ;;Alt: (mapcar #'(lambda (elt) (nth 1 elt)) rscored-pop)
  (mapcar #'cadr rscored-pop))
;; Tests
;; my-pop-scored
;; ((3 (a b c)) (1 (a)) (3 (e f g)) (2 (a d)))
;; (get-pop-from-scored my-pop-scored)
;; ((a b c) (a) (e f g) (a d))
```