**List Constructing Functions**

-**List:** Build a list from its arguments

-**Append:** concatenate list; take apart 2 lists and put them together

-**Cons:** prepend an element to a list

-**equal?**: the = function for lists

**Creating a Recursive Function**

*First, recall what happens during a  function call:*

1. *Copy argument(s) from the call into the parameter(s) of the function definition.*
2. *Execute the body of the function.*
3. *The result of the function replaces the call.*

**Steps for Writing a Recursive Function**

1. write the function signature and calls for ≥  2 test cases, and the result expected for each case
2. [tailor the general form of recursion](http://www.davidmholmes.net/Stuy/1intro_2014Spring/notes.html#2014-04-10), producing an English explanation of how recursion will succeed.
3. write body of function  
   label the base case and recursive case
4. test it

**Examples for Using List Construct Functions**

;Problem 1

(display "Problem 1a: Rev3-Cons") (newline)

(define (Rev3-Cons L)

(cons (caddr L) (list (cadr L) (car L))

)

)

;test

(Rev3-Cons '(a b c)) "...should be (c b a)"

(Rev3-Cons '(can we yes)) "...should be (yes we can)"

(newline)

;Problem 1b

(display "Problem 1b: Rev3-Append") (newline)

(define (Rev3-Append LL)

(Append (cddr LL) (list (cadr LL) (car LL))

)

)

;test

(Rev3-Append '(7 0 5)) "...should be (5 0 7)"

(Rev3-Append '(m i k)) "...should be (k i m)"

(newline)

;Problem 1c

(display "Problem 1c: Rev3-List") (newline)

(define (Rev3-List LLL)

(list (caddr LLL) (cadr LLL) (car LLL))

)

;test

(Rev3-List '(b u Y)) "...should be (Y u b)"

(Rev3-List '(1 1 9)) "...should be (9 1 1)"

;Problem 1d

(display "Problem 1d: Rev3-NoCons") (newline)

(define (Rev3-NoCons LLLL)

(Append (cddr LLLL) (list (cadr LLLL) (car LLLL)))

;test

(Rev3-NoCons '(1 2 3)) "...should be (3 2 1)"

(Rev3-NoCons '(x y z)) "...should be (z y x)"

(newline)

;-------------------------------------------

;Problem 2

(display "Problem 2: Last2Equal?") (newline)

(define (Last2Equal? L)

(equal? (car (reverse L)) (cadr (reverse L))

)

)

;test

(Last2Equal? '(a b c c)) "...should be #t"

(Last2Equal? '(yubin kim was here)) "...should be #f"

(Last2Equal? '(3 0 4 5 5 6 6)) "...should be #t"

;--------------------------------------------

;Problem 3

(display "Problem 3: MakePalindrome") (newline)

(define (MakePalindrome L)

(if (= (modulo (length L) 2) 0)

(append L (reverse L))

(append L (cdr (reverse L)))

)

)

;test

(MakePalindrome '(a b c)) "...should be (a b c b a)"

(MakePalindrome '(1 2 3 4)) "...should be 1 2 3 4 3 2 1"

**Problems of Recursion:**

(display "Problem: Last Element") (newline)

;Song of Recursion

(display "Song of Recursion") (newline)

(display "The question is how to find the last element of a list of n elements,

If i could solve this problem for a list with n-1 elements,

I know how to use that ability to solve for the last element for a list with n elements.

I first the cdr of the n elements and keep going

I know how to find the last element of a list of 1 element." )

;end of song of recursion

(newline)

(newline)

;definition

(define (lastElement L)

(if (= (length L) 1)

(car L)

(lastElement (cdr L))

)

) ;end of define

;test

(lastElement '(3 2 1 liftoff) ) "...should be liftoff"

(lastElement '(mks21)) "...should be mks21"

(lastElement '(first (last element is a sublist)) ) "...should be (last element is a sublist)"

(newline)

;-------------------------------------------

(display "Problem: All But Last") (newline)

;Song of Recursion

(display "Song of Recursion") (newline)

(display "The question is how to find the list of all but the last element of a list of n elements,

If i could solve this problem for a list with n-1 elements,

I know how to use that ability to solve for the list of all elements but the last element.

I first the first element of the list of n elements

and pair it with the all but last list of n-1 element.

I know how to find the allButLast list of a list of 1 element.");end of song of recursion

(newline)

;definition

(define (allButLast inputList)

(if (> (length inputList) 1)

(cons (car inputList) (allButLast (cdr inputList)))

(cdr inputList)

)

); end of define

;test

(allButLast '(3 2 1 liftoff)) "...should be (3 2 1)"

(allButLast '(mks21)) "...should be null list"

(allButLast '((sublists ok) omega )) "...should be (sublists ok)"

(newline)

;-----------------------------------------

;hw\_13

(display "Problem: Get Element") (newline)

;Song of Recursion

(display "Song of Recursion") (newline)

"The question is how to get the nth Element of a list.

If I could get the n-1th element of a lsit, I know that I can find the nth element of a list.

I can do this by decreasing the list and number of element by one every time.

I know how to solve this problem when n=0 by taking the car of the list."

;Definition

(define (getElement List nElement)

;base Case

(if (= nElement 0) (car List)

;Recursive case

(getElement (cdr List) (- nElement 1))

)

) ;end of define

;test

(getElement '(Brooks Dyrland-Weaver Brown Zamansky) 2) "...should be Brown"

(getElement '(0 1 2 3 4 5 6) 0) "...should be 0"

(getElement '(the cold (never bothered) me anyways) 4) "...should be anyways"

;-------------------------------------

(display "Problem: Ax Element") (newline)

;Song of Recursion

(display "Song of Recursion") (newline)

"The question is how to extract the nth element of a list.

If I could extract the n-1 element of a lsit, I know that I can extract the nth element of a list.

I can do this by decreasing the list and number of element by one every time.

I know how to solve this problem when n=0 by taking the cdr of this list."

;Definition

(define (axElement L n)

;base case

(if (= n 0) (cdr L)

;recursive case

(cons (car L) (axElement (cdr L) (- n 1)))

)

) ;end of define

;test

(axElement '(Brooks Dyrland-Weaver Brown Zamansky) 2) "...should be (Brooks Dyrland-Weaver Zamansky)"

(axElement '(0 1 2 3 4 5 6) 0) "...should be (1 2 3 4 5 6)"

(axElement '(let the storm rage on) 4) "....should be (let the storm rage)"

;-------------------------------------------

(diplay "Problem List Length")

(define (listLength L)

;base case

(if (null? L) 0

;recursive case

(+ (listLength (cdr L)) 1)

)

)

(listLength '(1 2 3 4 5 6)) "...should be 6"

(listLength '(Yubin Kim)) "...should be 2"

(listLength '(1)) "...should be 1"

(listLength '( )) "...should be 0"

;-------------------------------------------

(display “Problem: isPrefix?”

(define (isPrefix? Submaybe Main)

(cond

;base case

((null? Submaybe) #t)

;special case

((and (= 1 (length Submaybe)) (equal? Submaybe (list (car Main)))) #t)

;recursive case

((and (equal? (list (car Main)) (list (car Submaybe)))

(isPrefix? (cdr Submaybe) (cdr Main))) #t)

;what the function will return for everything else

(else #f)

)

) ;end of define