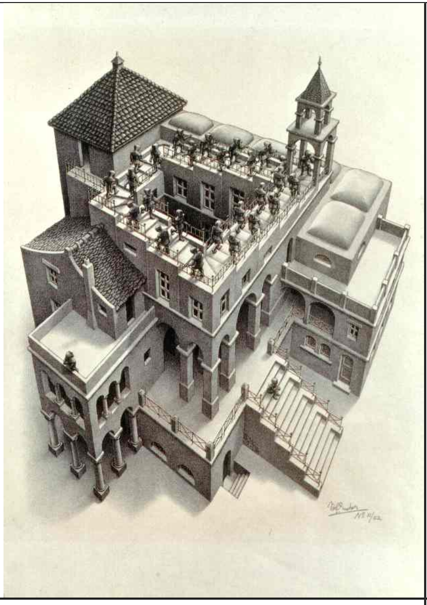
# List Recursion:

Practice & Examples



## One-Slide Summary

- Writing recursive functions that operate on recursive data structures takes **practice**. There are **standard approaches** to such problems.
- **list?**, **member**, **sumlist**, **intsto**, **map** and **filter** are all important recursive functions that operate on lists. You should know what they do and how to write them.
- DrScheme can **trace** the execution of a recursive function to make it easier to understand.

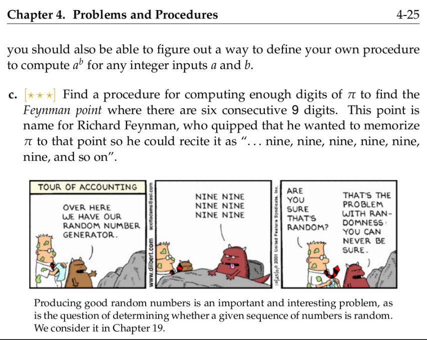
## Outline

- Review: Procedure Problem Solving
- Review: list, cons, car, cdr
- **`list?`**
- **`member`**
- **`sumlist`**
- **`intsto`**
- **`map`**
- **`filter`**
- Tracing

The scan has finished stopping.

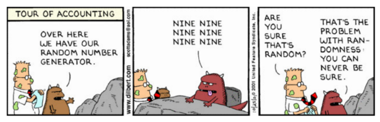
## Bookkeeping

- PS2 Partners Posted
  - Meet at structured lab hours?
- PS1 Written Grades Posted
  - Holding Fee
  - Pick them up
- Feynman Point?
  - Read the book!


Chapter 4. Problems and Procedures 4-25

you should also be able to figure out a way to define your own procedure to compute $a^b$ for any integer inputs $a$ and $b$.

**c.** [⋆⋆⋆] Find a procedure for computing enough digits of $\pi$ to find the *Feynman point* where there are six consecutive 9 digits. This point is name for Richard Feynman, who quipped that he wanted to memorize $\pi$ to that point so he could recite it as "… nine, nine, nine, nine, nine, nine, and so on".


TOUR OF ACCOUNTING
OVER HERE WE HAVE OUR RANDOM NUMBER GENERATOR.
NINE NINE NINE NINE NINE NINE
ARE YOU SURE THAT'S RANDOM?
THAT'S THE PROBLEM WITH RANDOMNESS. YOU CAN NEVER BE SURE.

Producing good random numbers is an important and interesting problem, as is the question of determining whether a given sequence of numbers is random. We consider it in Chapter 19.

## How To Write A Procedure

- Find out what it is supposed to do.
  - What are the **inputs**? What types of values?
  - What is the **output**? A number? Procedure? List?
- Think about some example inputs and outputs
- Define your procedure
  - More on this next slide
- Test your procedure

## Defining A Procedure

- Be optimistic!
- **Base case**: Think of the simplest input to the problem that you know the answer to.
  - For number inputs, this is often zero.
  - For list inputs, this is often the empty list (null).
- **Recursive step**: Think of how you would solve the problem in terms of a smaller input. Do part of the work now, then make a recursive call to handle the rest.
  - For numbers, this usually involves subtracting 1.
  - For lists, this usually involves cdr.

## Procedure Skeleton

- The vast majority of recursive functions look like this:

```
(define (my-procedure my-input)
  (if (is-base-case? my-input)
      (handle-base-case my-input)
      (combine (first-part-of my-input)
               (my-procedure (rest-of my-input)))))
```

## Pairs and Lists

- **cons** makes a pair of two things
  - (cons 1 2) --> (1 . 2)
  - (pair? (cons 1 2)) --> #t
- **car** and **cdr** get the first and second part
  - (car (cons “a” “b”)) --> “a”
  - (cdr (cons “y” “z”)) --> “z”
- A **list** is *either* **null** *or* a pair where the second element is also a list
  - (cons 1 (cons 2 (cons 3 null))) --> (1 2 3)
  - (list 1 2 3) --> (1 2 3)
  - (null? (list 1 2)) --> #f
  - (append (list 1 2) (list 3 4)) -> (1 2 3 4)

## More Power Needed!

## list?

- The **list?** function takes a single argument and returns #t if that argument is a list, #f otherwise.
  - Recall: a list is either null or a pair where the second element is a list
  - (list? null) --> #t
  - (list? (list 1 2)) --> #t
  - (list? (cons 1 null)) --> #t
  - (list? 5) --> #f
  - (list? (cons 1 2)) --> #f
- Write it now on paper. Base case? Recursion?

## list? Hint

- Here's a hint:

```
(define (list? something)
  (if (null? something) #t
      ...))
```

## Definition of list?

- Here it is:

```
(define (list? something)
  (if (null? something) #t
      (if (pair? something)
          (list? (cdr something))
          #f) ))
```

Base Case!

Inductive Step!

## Liberal Arts Trivia: Economics

- This 1930 Tariff Act raised US tariffs on imported goods to record levels. Over 1000 US Economists signed a petition against it, and after it passed many other contributed increased their tariffs in retribution. US exports and imports dropped by half and many view this Act as a major catalyst for the Great Depression.

## Liberal Arts Trivia: German Lit

- This tragic closet play is considered by many to be one of the greatest works of German literature. It centers on a man who makes a pact with the Devil in exchange for knowledge in his quest to discover the essence of life ("was die Welt im Innersten zusammenhält") The man's name officially means "Lucky" in Latin, but now has negative connotations.

## member

- Write a function **member** that takes two arguments: an element and a list. It returns #f if the list does not contain the element. Otherwise it returns the sublist starting with that element.
  - (member 2 (list 1 2 3)) -> (2 3)
  - (member 5 (list 1 2 3)) -> #f
  - (member 1 (list 1 2 3)) -> (1 2 3)
  - (member 3 (list 1 2 3)) -> (3)
  - (eq? 3 5) -> #f        (eq? 2 2) -> #t

## Definition of member

```
(define (member elt lst)
  (if (null? lst)
     #f           ;; empty list contains nothing
     (if (eq? elt (car lst))
        lst       ;; we found it!
        (member elt (cdr lst))))) ;; keep looking
```

- Where is the base case? Where is the inductive step?

## sumlist

- Write a procedure **sumlist** that takes as input a list of numbers. It returns the sum (addition) of all of the elements of the list. It returns 0 for the empty list.
  - (sumlist (list 1 2 3)) -> 6
  - (sumlist null) -> 0

## Definition of sumlist

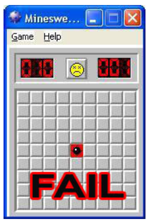
- And here it is ...

```
(define (sumlist lst)
  (if (null? lst)
     0                ;; base case
     (+ (car lst)     ;; else add current element
        (sumlist (cdr lst))))) ;; to rest of list
```

## intsto

- The function **intsto** takes a single non-negative integer as an argument. It produces a list of all of the integers between 1 and its argument.
  - (intsto 3) -> (1 2 3)
  - (intsto 7) -> (1 2 3 4 5 6 7)
  - (intsto 0) -> null

## Definition of intsto ?



```
(define (intsto x)
 (if (< x 1)
   null          ;; base case
   (cons         ;; else make a list
     x           ;; list contains x
     (intsto (- x 1))))) ;; and recursive result
```

- What's wrong?

## **Correct** Definition of intsto

```
(define (intsto x)
 (if (< x 1)
   null          ;; base case
   (append       ;; else make a list
     (intsto (- x 1))    ;; recursive result
     (list x))))         ;; followed by x
```

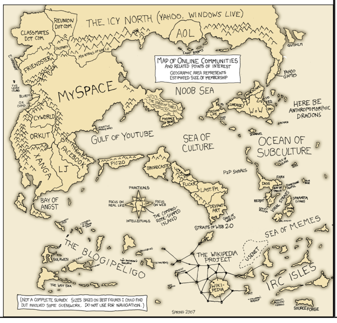
- Huzzah!

## Higher-Order Functions: map

- The **map** function takes two arguments: a work function and a list. It applies the work function to every element of the list in order and returns a list of the result.
  - (map sqrt (list 9 16 36)) -> (3 4 6)
  - (map square (list 1 2 3)) -> (1 4 9)
  - (map abs (list 2 -3 4)) -> (2 3 4)
  - (map string-length (list “I” “Claudius”)) -> (1 8)
  - (map sqrt null) -> null

## Mission Impossible: Write map

- You can do it!
  - (map square (list 1 2 3)) -> (1 4 9)
  - (map abs (list 2 -3 4)) -> (2 3 4)
  - (map sqrt null) -> null



## Definition of map

- Let's look in detail:

```
(define (map work-fun lst)
 (if (null? lst)
    null          ;; base case
    (cons         ;; else make a list
      (work-fun (car lst)) ;; first part of result
      (map work-fun (cdr lst))))) ;; rest o'result
```

## Liberal Arts Trivia: Philosophy

- This branch of philosophy deals with the theory, nature and scope of knowledge. Key questions include "what is knowledge?", "how is knowledge acquired?", "what do people know?", "how do we know what we know?", "what is the relationship between truth and belief?".

#25

## Liberal Arts Trivia: Norse Myth

- In Norse Mythology, this god is associated with light and beauty. His mother made every object on earth vow never to harm him, but she did not ask mistletoe. The other gods made a new pastime of hurling objects at him and watching them bounce off. The trickster Loki heard of this, fashioned a spear from mistletoe and had it thrown a him, with fatal results.

#26

## Using map to get iteration

- In C or Java:

```
for (x=1 ; x <= 5 ; x=x+1) {
    display(x*x);
} // output: 1 4 9 16 25
```

- Recall that we have intsto:
  - (intsto 3) -> (1 2 3)
  - (intsto 7) -> (1 2 3 4 5 6 7)
- How can we use map to get for?

#27

## Using map to get iteration

- In C or Java:

```
for (x=1 ; x <= 5 ; x=x+1) {
    display(x*x);
} // output: 1 4 9 16 25
```

- Recall that we have intsto:
  - (intsto 3) -> (1 2 3)
- Then we can do:

**(map (lambda (x) (display (square x))) (intsto 5))**

Expect me on tests or extra credit later!

#28

## filter

- The **filter** function takes two arguments: a predicate and a list. A **predicate** is a function that returns true or false. Filter returns the sublist consisting of those elements that satisfy the predicate.
  - (filter is-odd? (list 1 2 3 4)) -> (1 3)
  - (filter null? (list 1 null null "hi")) -> (null null)
  - (filter (lambda (x) (< x 5)) (list 1 9 2 0)) -> (1 2 0)
  - (filter null? (list "susan" "b" "anthony")) -> null
  - (filter is-odd? null) -> null

#29

## Definition of filter

```
(define (filter pred lst)
  (if (null? lst)
      null          ;; base case
      (if (pred (car lst))  ;; does this element
                            ;; satisfy the predicate?
          (cons (car lst) (filter pred (cdr lst)))
                ;; if so, include it in the result
          (filter pred (cdr lst)))))
                ;; if not, do not include it
```

#30

## Tracing

- DrScheme will **trace** through a functions execution for you.
- This can make it easier to debug or understand a function.

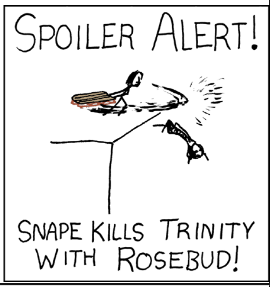
To enable tracing of ***myfun***:

```
(require (lib "trace.ss"))
(trace myfun)
```

## Tracing sumlist

```
(define (sumlist p)
  (if (null? p) 0 (+ (car p) (sumlist (cdr p)))))
```

```
> (trace sumlist)
> (sumlist (list 1 2 3 4))
|(sumlist (1 2 3 4))
| (sumlist (2 3 4))
| |(sumlist (3 4))
| | (sumlist (4))
| | |(sumlist ())
| | |0
| | 4
| |7
| 9
|10
10
```



## Tracing map

```
> (map (lambda (x) (* x 2)) (list 1 2 3))
|(map #<procedure> (1 2 3))
| (map #<procedure> (2 3))
| |(map #<procedure> (3))
| | (map #<procedure> ())
| | ()
| |(6)
| (4 6)
|(2 4 6)
(2 4 6)
```

```
(define (map f lst)
  (if (null? lst)
      null
      (cons (f (car lst))
            (map f (cdr lst)))))
```

## Homework

- Problem Set 2 Due Monday
- Read GEB 5 by Monday