Lecture 10: Lists in Scheme

Kaleel Mahmood

Department of Computer Science and Engineering

University of Connecticut

Previously in CSE 1729, two concepts…

Rams need Cake: Tail Recursion using a ridiculous example

```
(define (fact-accumulate n a)
  (if (= n 0) a
      (fact-accumulate (- n 1)
                       (* n a))))
```

Previously in CSE 1729, two concepts…

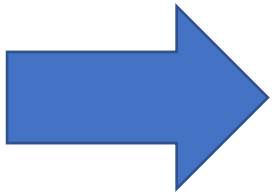
Pairs in Scheme:

PAIRS

- Construction

```
(define z (cons x y))
```


```
1 (define z (cons 2 3))
2
3 (car z)
4
5 (cdr z)
```



```
Welcome to DrRacket, version 8.3 [cs].
Language: R5RS; memory limit: 128 MB.
2
3
>
```

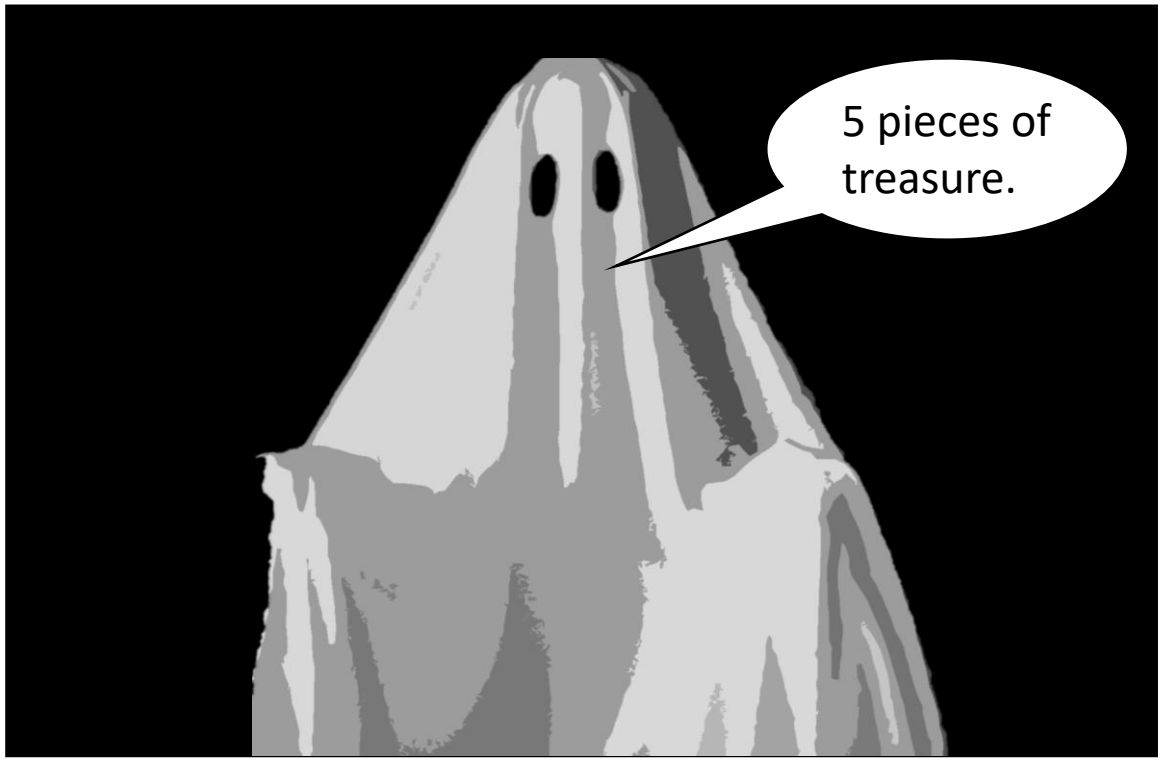
Ghost Guarded Treasure!

- You stumble upon a cavern of riches.
- The treasure is guarded by a ghost who says you can only take a pouch in.
- The pouch can hold two items. Once you enter the cavern one time and leave, you can never return again.
- *How do you recover all the treasure in one trip?*

Solution to the Ghost Guarded Treasure problem

- The ghost says you can only take in a pouch that can hold two items.
- However the ghost did *NOT* specify the pouch had to be empty.
- Do the following:

1. Ask the ghost how many pieces of treasure they are guarding.
2. Get your first pouch. Keep the first slot empty. Put a SECOND pouch in the second slot of the first pouch.

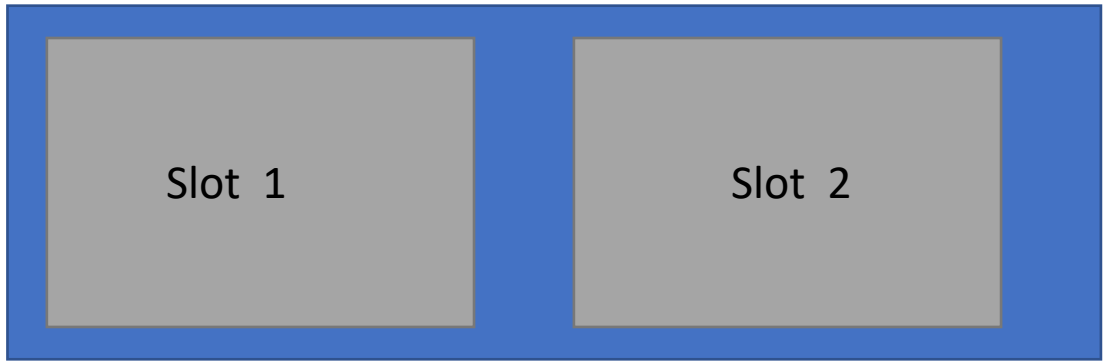


Solution to the Ghost Guarded Treasure problem

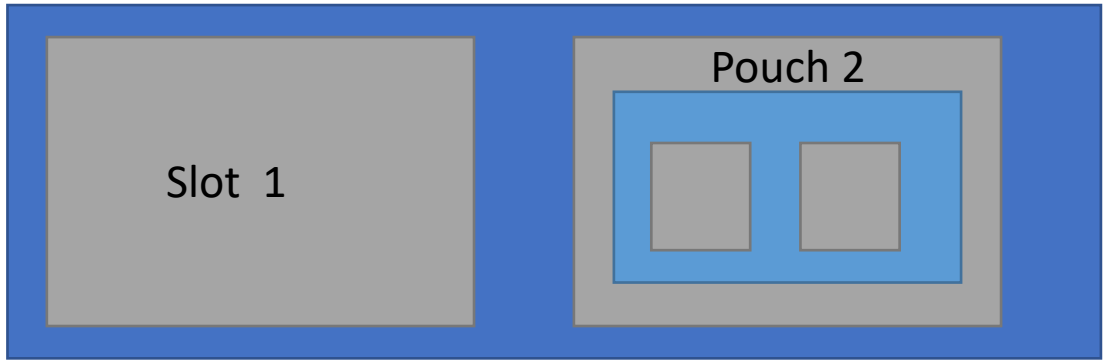
- Do the following:

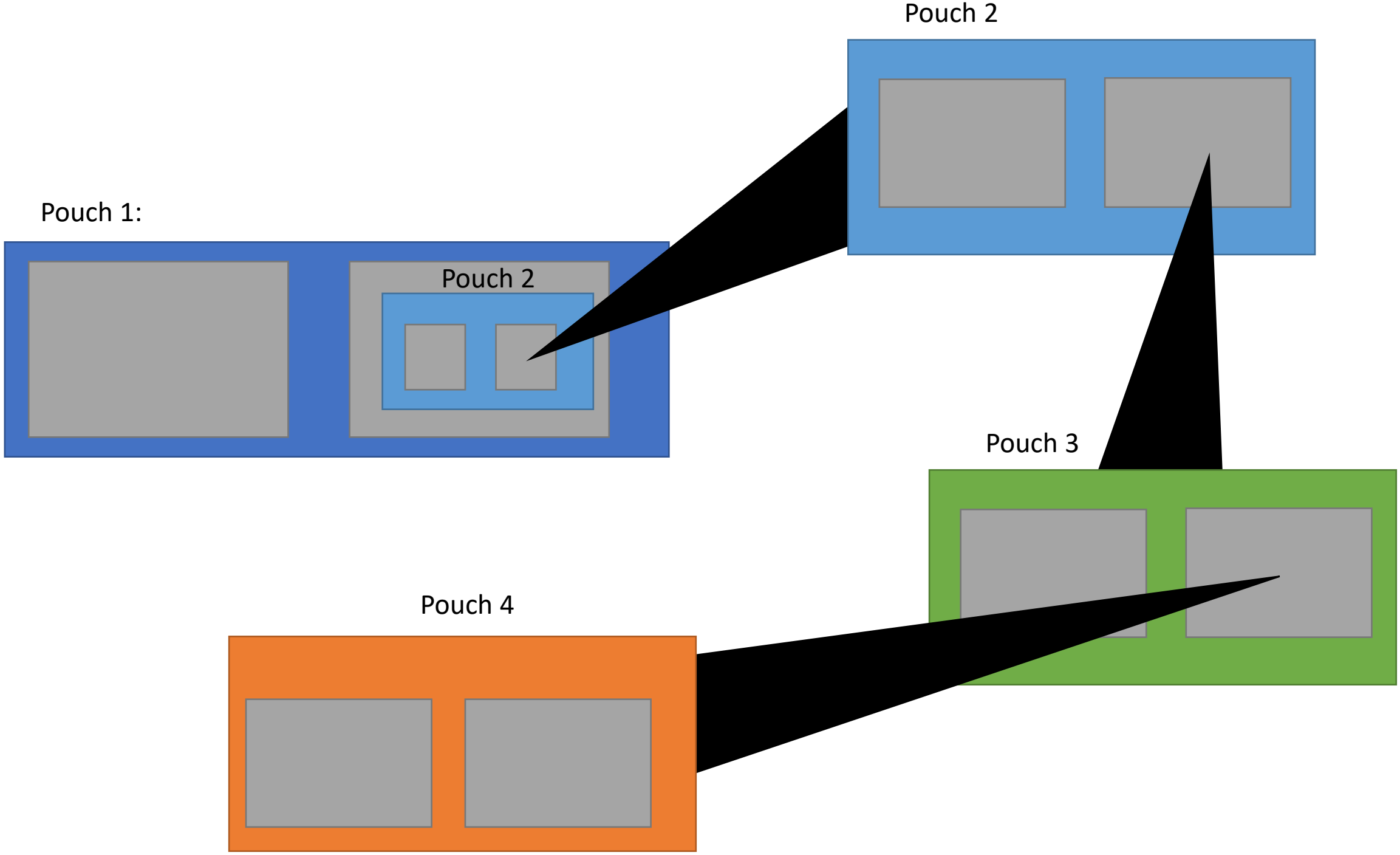
1. Ask the ghost how many pieces of treasure they are guarding.
2. Get your first pouch. Keep the first slot empty. Put a SECOND pouch in the second slot of the first pouch.
3. *In the second pouch, second slot, put another pouch.*
4. *Repeat this process until you have a number empty slots equal to the amount of treasured items in the cave.*
5. *Profit.*

Pouch 1:



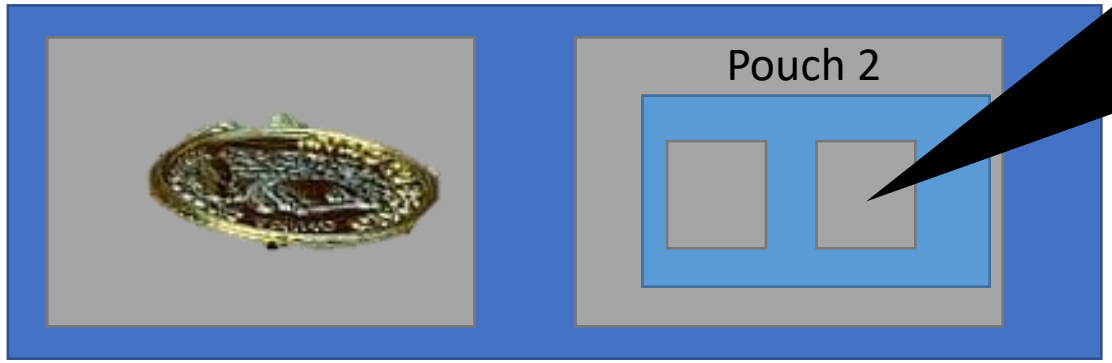
Pouch 1:



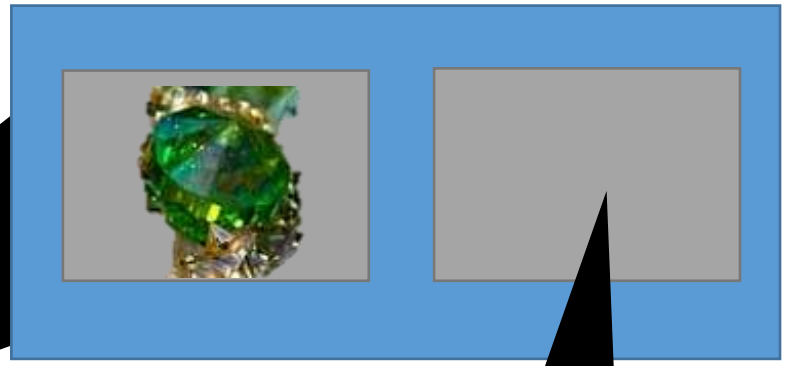

Pouch 2
Pouch 1:
Pouch 2
Pouch 3
Pouch 4

Example: 4 pieces of treasure are in the trove

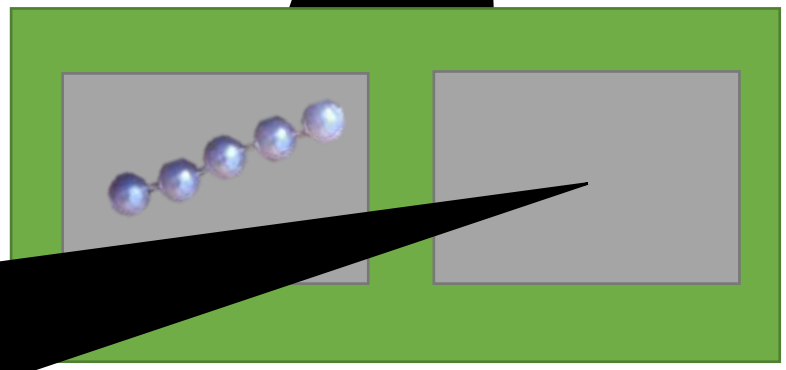
Pouch 1:



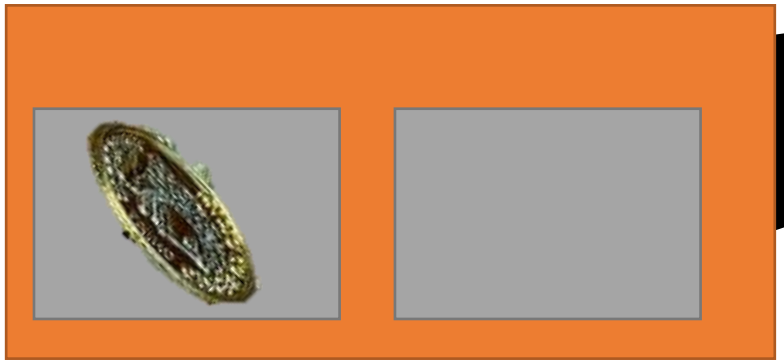
Pouch 2



Pouch 3



Pouch 4



Question: *In Scheme we learned pairs (an object that can hold two items). Can we build objects that can hold more than two items out of pairs?*

Lists in Scheme

- A *list* is an extremely flexible data structure that maintains an ordered list of objects, for example:
  - *Ceres*, *Pluto*, *Makemake*, *Haumea*, *Eris*, a list of 5 extrasolar planets.
- Scheme implements lists **in terms of the pair structure** you have already met.
  - However, pairs have only 2 slots, so we need a mechanism for using pairs to represent lists of arbitrary length.
- Roughly, Scheme uses the following recursive convention: the list of k objects $a_1, \ldots, a_k$ is represented as a pair where...
  - The first element of the pair is the first element of the list a_1.
  - The second element of the list is...*a list containing the rest of the elements*.

BUILDING UP LISTS WITH PAIRS

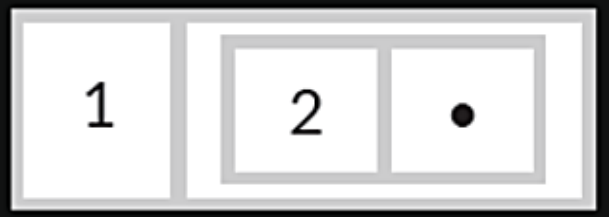
- To be more precise: A *list* is either
  - the *empty list*, or
  - a *pair*, whose first coordinate is *the first element of the list*, and whose second coordinate is *a list containing the remainder of the elements*.
- Note: *the second element of the pair must be a list*.

- For example, if • denotes the empty list, then...

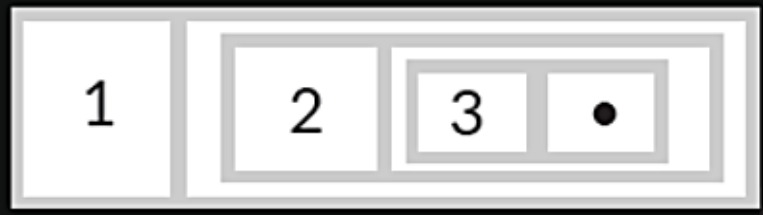
( )

(1)

(1 2)



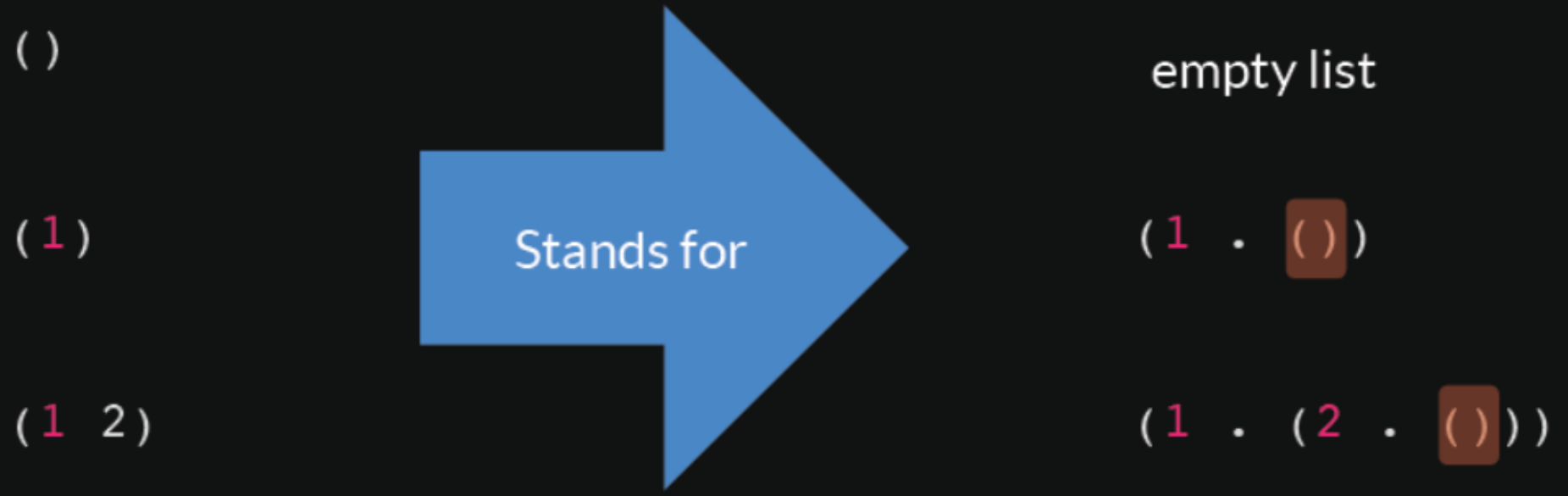
(1 2 3)



A GENERAL LIST; SCHEME NOTATION

- Thus, a list has the form:

- Since lists are used so frequently, Scheme provides special notation for them:



Note: In Scheme, **lists are always terminated with the empty list.**

QUOTATION; ENTERING LISTS IN THE SCHEME INTERPRETER

- Recall the Scheme evaluation rule for compound (list!) objects.
- This means that the natural way to enter a list doesn't work: Scheme wants to apply evaluation:

```
> ()
. #%app: missing procedure expression; probably originally (), which is an illegal empty application in: (#%app)
> (1 2)
. . procedure application: expected procedure, given: 1; arguments were: 2
```

- Scheme provides the `(quote <expr>)` (or `'<expr>`) form, which evaluates to <expr> without further evaluation:

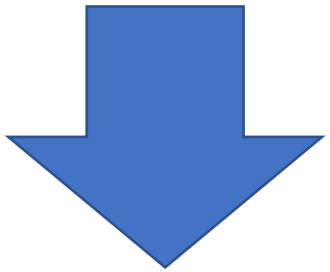
```
> (quote ())
()
> (quote (1 . ()))
(1)
> (quote (1))
(1)
> '(1)
(1)
```

Note how Scheme denotes these identical structures

'<expr> is shorthand for (quote <expr>)

List Syntax

```
1 | (define x (list 1 2 3 ))
2 | x
```

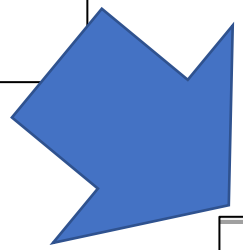


```
Welcome to DrRacket, version 8.3 [cs].
Language: R5RS; memory limit: 128 MB.
(1 2 3)
```

How do we know a list is also technically a pair?

Try using the car and cdr functions on the list.

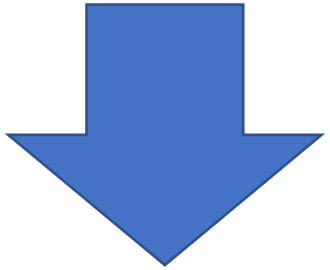
```
1 | (define x (list 1 2 3))
2 | (car x)
3 | (cdr x)
```



```
Welcome to DrRacket, version 8.3 [cs].
Language: R5RS; memory limit: 128 MB.
1
(2 3)
```

Can you put a list in a list?


```
1 (define y (list 1 (list 2 3) 4 5 ))
2 y
```

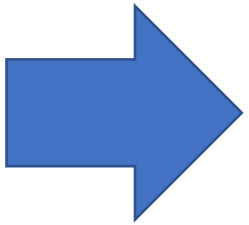


```
Welcome to DrRacket, version 8.3 [cs].
Language: R5RS; memory limit: 128 MB.
(1 (2 3) 4 5)
```

How do you check if a list is empty?

- Just ask!

```
1  ;Empty List
2  (define x (list ))
3  ;Non-empty List
4  (define y (list 1 2 3 ))
5
6  (null? x)
7  (null? y)
```



```
Welcome to DrRacket, version 8.3 [cs].
Language: R5RS; memory limit: 128 MB.
#t
#f
```

LIST PROCESSING:

HANDLE THE FIRST ELEMENTS AND, THEN,...HANDLE THE REST

- (null? x) returns #t if x is the empty list.
- list processing:
  - handle the first element (the car) and, then,
  - handle the remaining list (the cdr).
  - Notice that these have different "types."
- Computing the length, for example...

```
(define (nlength xyz)
  (if (null? xyz)
      0
      (+ 1 (nlength (cdr xyz)))))
```

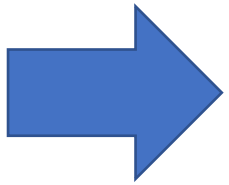
Then ...

```
> (nlength '(1 2 3))
3
> (nlength '())
0
> (nlength '((1 2) (3 4)))
2
```

How do we find the end of an arbitrary list?

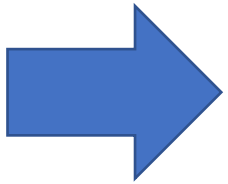
- What if we repeatedly use the cdr function to reduce the list…

```
1 ;Non-empty List
2 (define x (list 1 2 3 ))
3 (cdr x)
```



```
Welcome to DrRacket, version 8.3 [cs].
Language: R5RS; memory limit: 128 MB.
(2 3)
```

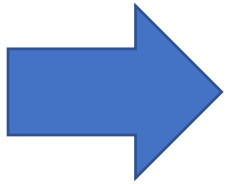
```
1 ;Non-empty List
2 (define x (list 1 2 3 ))
3 (cdr (cdr x))
```



```
Welcome to DrRacket, version 8.3 [cs].
Language: R5RS; memory limit: 128 MB.
(3)
```

Calling cdr one more time…

```
1 ;Non-empty List
2 (define x (list 1 2 3 ))
3 (cdr (cdr (cdr x)))
```



```
Welcome to DrRacket, version 8.3 [cs].
Language: R5RS; memory limit: 128 MB.
()
```

What happened? Scheme uses the empty list to indicate termination (that you final element of a list)

Getting to the last element in a list…

- If list is a list, it is easy to get to the first element: (car list).
- The last element, however, takes more work to find.
  - This is an inherent feature (and, sometimes, shortcoming) of this "data structure."

```
(define (last-element l)
  (if (null? (cdr l))
      (car l)
      (last-element (cdr l))))

> (last-element '(5 4 3 2 1))
1
```

A side note…How did we get the last element of a list in Python?

```
(define (last-element l)
  (if (null? (cdr l))
      (car l)
      (last-element (cdr l))))
```

```
1  l = [1 ,2, 3]
2  lastIndex = len(l)-1
3  print(l[lastIndex])
```

ANOTHER EXAMPLE: SUMMING THE NUMBERS OF A LIST

- Adding the elements of a list:

```
1  (define (sum-list list)
2    (if (null? list)
3        0
4        (+ (car list)
5           (sum-list (cdr list)))))
```

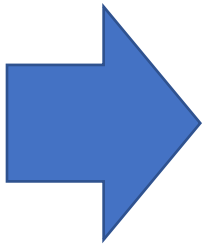
- Then...

```
1  > (sum-list '())
2  0
3  > (sum-list '(1 3 5 7))
4  16
```

How do we combine lists?

- Use the built in append function…

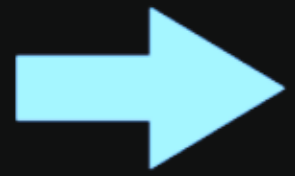
```
1  (define x (list 1 2 3))
2  (define y (list 4 5 6))
3
4  (define z (append x y))
5  z
```



```
Welcome to DrRacket, version 8.3 [cs].
Language: R5RS; memory limit: 128 MB.
(1 2 3 4 5 6)
```

Append: What is going on under the hood?

- Basic operation on lists: place one after the other:

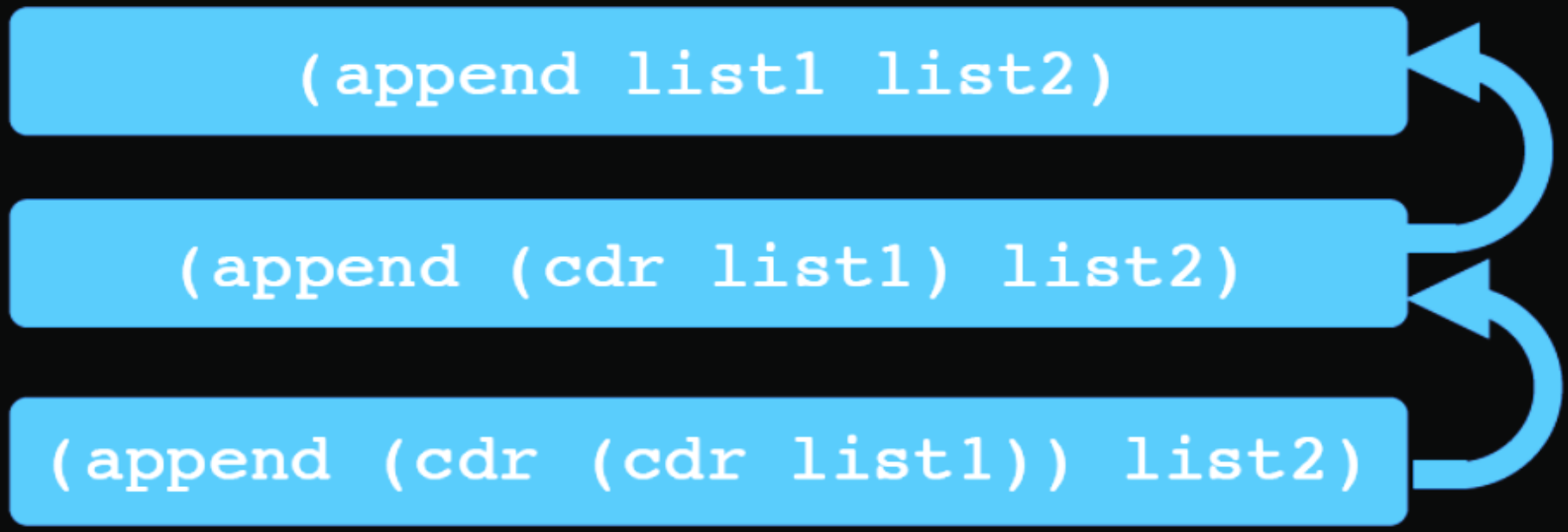
`(1 2 3)` **append** `(11 12 13)`  `(1 2 3 11 12 13)`

- It's easy:

```
(define (append list1 list2)
  (if (null? list1)
      list2
      (cons (car list1)
            (append (cdr list1) list2))))
```

HOW LONG DOES THIS TAKE?

- A good measure of the "time taken" by a `Scheme` function (without looping constructs, which we will discuss later) is simply the number of recursive calls it generates.
- `(append list1 list2)` involves a total of `length(lista)` recursive calls. (Why? It needs to find the end of the list.)

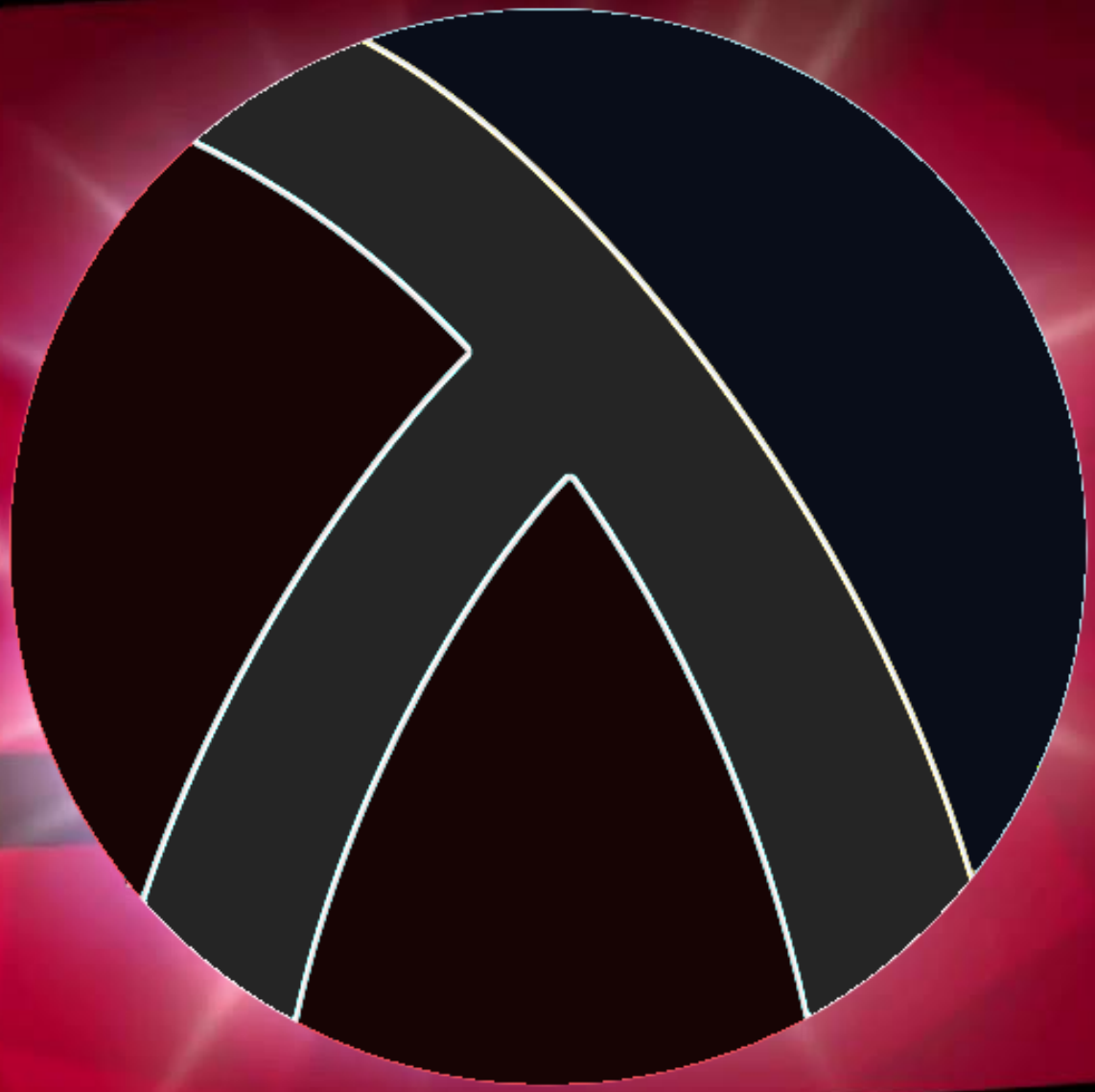


A LIST USER'S GUIDE...

- Suppose that `L` is a list in Scheme;
  - then you can tell if it is empty by testing `(null? L)`; if not...
  - its first element is `(car L)`;
  - the "rest" of the elements are `(cdr L)` (this is a list, and might be empty).

- Suppose that L is a list in Scheme and x is a value;
  - `'()` or `(list)` is the empty list.
  - `(cons x L)` is a new list—its first element is `x`; the rest of the elements are those of `L`.
  - The list containing only the value `x`? Same idea, but use the empty list for `L`: `(cons x '())` or `(list x)`.

A new foe has appeared!



CHALLENGER APPROACHING

The Ordered Square List Challenge

```
(define (square-list k)
  (if (= k 0)
      (list 0)
      (cons (* k k)
            (square-list (- k 1)))))
```

- Hmm... it lists them backwards!

```
> (square-list 4)
(16 9 4 1 0)
```

SQUARES IN THE RIGHT ORDER

- It's easy if both ends of a range are given: (why did this make it easy?)

```
(define (squares start finish)
  (define (square x) (* x x))
  (if (> start finish) '()
      (cons (square start)
            (squares (+ start 1) finish))))
```

- We can wrap this in a definition that starts at zero:

```
(define (forward-squares k)
  (define (square x) (* x x))
  (define (squares start finish)
    (if (> start finish) '()
        (cons (square start) (squares (+ start 1) finish))))
  (squares 0 k))
```

Figure Sources

- https://cdn.vox-cdn.com/thumbor/pr3jD5sfTRKpPinnYym_4A0gJaQ=/0x27:4415x3338/1400x1400/filters:focal(0x27:4415x3338):format(jpeg)/cdn.vox-cdn.com/uploads/chorus_image/image/43170476/ghosts.0.0.jpg
- https://www.worldhistory.org/img/r/p/500x600/14661.jpg?v=1639573202
- https://imgflip.com/memegenerator/Yo-Dawg-Heard-You
- https://i.imgur.com/rGQN1Em.png
- https://storage.googleapis.com/kapwing/final_6023fd031a86500094fcebb3_722930.png