```
;; hi-lo:  number number → number
;; Purpose: consumes the endpoints of an interval and finds
;;              the number hidden by guess.  Uses a strategy
;;              called binary search to make this efficient.
(define (hi-lo   lo hi)
  (local [ (define midpoint  (truncate (/ (+ lo hi) 2)))
             (define answer      (guess midpoint))]
          (cond
             [(symbol=?  answer midpoint) midpoint]
             [(symbol=?  answer 'higher)  (hi-lo  midpoint  hi)]
             [(symbol=?  answer  'lower)  (hi-lo  lo  midpoint)] ))
```

```
;; hi-lo: natnum natnum → natnum
;; Purpose: given low & high, return the hidden number in
;;              the interval [low, high]
(define (hi-lo lo hi)
  (cond   [(symbol=? (guess hi) 'equal) hi]
          [else
              (local [ (define  mid   (truncate (/  (+ lo hi) 2)))
                       (define answer  (guess mid))]
                    (cond
                       [(symbol=?  answer  'equal)    mid]
                       [(symbol=?  answer  'higher)  (hi-lo mid hi)]
                       [(symbol=?  answer  'lower)   (hi-lo  lo mid)]
                       )
              )
             ]
      ))
```

```
;; hi-lo: int int -> int
;; Purpose: given low & high, return the hidden number
;;              in the interval [low, high]
(define (hi-lo lo hi)
  (local [  (define  mid   (truncate (/  (+ lo hi) 2)))
            (define answer  (guess mid))]
       (cond
          [(symbol=?  answer  'equal)    mid]
          [(symbol=?  answer  'higher)  (hi-lo  (add1 mid) hi)]
          [(symbol=?  answer  'lower)   (hi-lo  lo (sub1 mid) )]
          )
   ))
```

A city is a symbol.

```
;; The information for a city can be represented as a structure
;;  (make-city-info name dests)
;; where c is a city (symbol) and dests is a list of symbole
(define-struct city (name dests))


;; A route-map is a list of city-info
(define routes
  (list (make-city-info  'Houston   (list 'Dallas 'NewOrleans))
        (make-city-info 'Dallas   (list  'LittleRock  'Memphis))
        (make-city-info  'NewOrleans (list 'Memphis))
        (make-city-info 'Memphis  (list 'Nashville)) ))


;; find-flights: city city route-map → (list of city) or false
;; Purpose: create a path of flights from start to finish or return false
(define (find-flights start finish  rm) …)
```

Examples:

```
(find-flights  'Houston 'Houston routes)
= (list 'Houston)

(find-flights 'Houston 'Dallas)
= (list 'Houston 'Dallas)

(find-flights 'Dallas 'Nashville)
= (list 'Dallas 'LittleRock 'Memphis 'Nashville)
```

Original Version

```
;; find-flights: city city route-map → (list of city) or false
;; Purpose: create a path of flights from start to finish or return false
(define (find-flights start finish  rm)
  (cond
    [(symbol=? start finish) (list start)]
    [(else
       (local [(define possible-route
                    (find-flights-for-list (direct-cities start rm) finish rm))]
              (cond
                [(boolean? possible-route)  false]
                [else  (cons start possible-route)])) ] ))


;; direct-cities: city route-map → list-of-city
;; Purpose: return a list of all cities in the route map with direct flights
;;       from the city given as an argument
(define (direct-cities  from-city  rm)
   (local [(define  from-city-info
              (filter (lambda (c)(symbol=? (city-info-name c) from-city)) rm))]
         (cond
            [(empty? from-city-info)  empty]
            [else (city-info-dests (first (from-city-info))])))

;; find-flights-for-list: list-of-city city route-map → list-of-city or false
;; Purpose: finds a flight route from some city in the input list to the
;;       destination, or returns false if no such route can be found.
(define (find-flights-for-list aloc finish rm)
  (cond
      [(empty? aloc)  false]
      [else
       (local [(define possible-route
                        (find-flights (first aloc) finish rm))]
              (cond
               [(boolean? possible-route)
                (find-flights-for-list (rest aloc) finish rm)]
              [else  possible-route]))]))
```

With Institutional Memory

```
;; find-flights: city city route-map (list of city) → (list of city) or false
;; Purpose: create a path of flights from start to finish or return false
(define (find-flights start finish rm visited)
  (cond
    [(symbol=? start finish) (list start)]
    [(memq start visited)   false]  ;; cut off this search path
    [(else
       (local [(define possible-route
                 (find-flights-for-list (direct-cities start rm) finish
                                        rm  (cons start visited)))]
         (cond
           [(boolean? possible-route)  false]
           [else  (cons start possible-route)])) ] ))


;; direct-cities: city route-map → list-of-city
;; Purpose: return a list of all cities in the route map with direct flights
;;       from the city given as an argument
(define (direct-cities  from-city  rm)
   (local [(define  from-city-info
             (filter (lambda (c)(symbol=? (city-info-name c) from-city))
                     rm))]
     (cond  [(empty? from-city-info)  empty]
            [else (city-info-dests (first (from-city-info))])))

;; find-flights-for-list: list-of-city city route-map (list of city)
;;                                → list-of-city or false
;; Purpose: finds a flight route from some city in the input list to the
;;       destination, or returns false if no such route can be found.
(define (find-flights-for-list aloc finish rm visited)
  (cond
     [(empty? aloc)  false]
     [else
      (local [(define possible-route
                (find-flights (first aloc) finish rm visited))]
        (cond
         [(boolean? possible-route)
          (find-flights-for-list (rest aloc) finish rm visited)]
         [else  possible-route]))]))
```

(find-flights ‘Houston ‘Memphis routes)

=> (find-flights 'Houston 'Memphis routes)
    => (direct-cities 'Houston)
    => (find-flights-for-list
        (list 'Dallas 'NewOrleans) 'Memphis routes)

=>(find-flights 'Dallas 'Memphis routes)
    => (direct-cities 'Dallas)
    => (find-flights-for-list
        (list 'LittleRock 'Memphis) 'Memphis routes)

Dead end !

=> (find-flights 'LittleRock 'Memphis routes)
    => (direct-cities 'LittleRock)
    => (find-flights-for-list empty 'Memphis routes)

=> (find-flights-for-list (list 'Memphis) 'Memphis routes)
    => (find-flights 'Memphis 'Memphis)

returns (list 'Houston 'Dallas 'Memphis)

Never got this far !!