Old SCVAL Door Prize Problem

; code in the process

; comments

; answer

; process of thinking

The Three Factors					Triangular Number		Notation	
TFTN#	**1st**	**2nd**	**3rd**					
1	1	2	3	Π=	6	=	T	3
2	**4**	**5**	**6**	Π=	**120**	=	T	**15**
3	**5**	**6**	**7**	Π=	**210**	=	T	**20**
4	**9**	**10**	**11**	Π=	**990**	=	T	**44**
5	**56**	**57**	**58**	Π=	**185136**	=	T	**608**
6	636	637	638	Π=	258474216	=	T	22736

Notes for understanding

- Notation, first of the three factors, triangular num

My idea

- Create a list for all the possible consecutive sums
- Create a list for all the possible products of three nums
- Compare the numbers in each and if it matches, return it

Process of thinking

- Writing the list for product

```
(define (triplet x)
  (* x (+ x 1) (+ x 2)))
```

The input to x should be 0

```
(define (prod1 x)
  (define (iter num result)
    (if (> num 636)
        result
        (iter (+ num 1) (cons (triplet num) result))))
  (iter x '()))
```

- This returns a backward list, starting with greatest number

- Writing the list for sum

```
The input to x should be 0
(define (sum1 x)
  (define (iter num list-result total-sum)
    (if (= num 22736)
        list-result
        (iter (+ num 1) (cons (+ num total-sum) list-result) (+
num total-sum))))
  (iter x '() 0))
```

- Again, this returns a backward list

- Reversing both the lists

```
; input starting with 636
(define (prod x)
  (define (iter num result)
    (if (= num 0)
        result
        (iter (- num 1) (cons (triplet num) result))))
  (iter x '()))

; input starting with one
(define (sum x)
  (define (iter num list-result total-sum)
    (if (> num 22736)
        list-result
        (iter (+ num 1) (append list-result (list (+ num
total-sum))) (+ num total-sum))))
  (iter x '() 0))
```

- Original idea: use list-ref and then compare numbers

```
(define (list-of-sums x)
  (list-ref (sum 1) x))

(define (list-of-prod x)
  (list-ref (prod 636) x))
```

```
(define (func2 x y)
  (define (iter n1 n2 result)
    (cond ((or (= n1 22735) (= n2 635)) result)
          ((= (list-of-sums n1) (list-of-prod n2))
           (iter (+ n1 1) (+ n2 1) (cons (list-of-prod n2)
result)))
          ((< (list-of-sums n1) (list-of-prod n2))
           (iter (+ 1 n1) n2 result))
          ((> (list-of-sums n1) (list-of-prod n2))
           (iter n1 (+ 1 n2) result))
          (else result)))
  (iter x y '()))
```

- This takes too long

- Instead, use let to define the computed lists, so we do not have to compute it every time
- Both lists advance at the same time to reduce time
- Return number of the list of sum (highest notation), number of the list of prod (first factor), list-ref of prod (triangular number)

```
(define (func)
  (let ((list-sum (sum 1))
        (list-prod (prod 636)))
    (define (iter n1 n2 result)
      (cond ((or (= n1 22736) (= n2 636)) result)
            ((= (list-ref list-sum n1) (list-ref list-prod n2))
             (iter (+ n1 1) (+ n2 1) (append result (list (list
(+ n1 1) (+ n2 1) (list-ref list-prod n2))))))
            ((< (list-ref list-sum n1) (list-ref list-prod n2))
             (iter (+ n1 1) n2 result))
            ((> (list-ref list-sum n1) (list-ref list-prod n2))
             (iter n1 (+ n2 1) result))
            (else result)))
    (iter 0 0 '())))
```

Returns: '((3 1 6) (15 4 120) (20 5 210) (44 9 990) (608 56 185136) (22736 636 258474216))

```
; takes about 20 seconds
```