; Recursion: Sierpinski Triangle, part II.

(require picturing-programs)

; stack: image -> image
; an-image centred above two copies of an-image beside each other.
(define (stack an-image)
  (above an-image (beside an-image an-image)))

(check-expect (stack (star 25 "outline" "red"))
  (star 25 "outline" "red"))

; Sierpinski: number -> image
(define (Sierpinski n)
  (cond [(zero? n) ▲]
       [else (stack (Sierpinski (- n 1))))]))

(check-expect (Sierpinski 0)
  (stack (Sierpinski 0)))

(check-expect (Sierpinski 0)
  (cond [(zero? 0) ▲]
       [else (stack (Sierpinski (- 0 1))))])

(check-expect (Sierpinski 0)
  (cond [true ▲]
       [else (stack (Sierpinski (- 0 1))))])

(check-expect (Sierpinski 0)
  ▲)

; Steps for (Sierpinski 1). Don't want to distract with check-expects.
(Sierpinski 1)

(check-expect (Sierpinski 1)
  (cond [(zero? 1) ▲]
       [else (stack (Sierpinski (- 1 1))))])

(check-expect (Sierpinski 1)
  (cond [false ▲]
       [else (stack (Sierpinski (- 1 1))))])

(stack (Sierpinski (- 1 1)))
(stack (Sierpinski 0))
(stack ▲); From our work on (Sierpinski 0).

(above ▲ (beside ▲ ▲))

; So:

(check-expect (Sierpinski 1) ▲)

And if that was enough practice, we can jump to:
(check-expect (Sierpinski 2)

(above ▲ (beside ▲ ▲ ▲)))

; So what does (Sierpinski 2) look like?