; Lists, Part II: Number Ranges, and Indexing
; -----------------------------------------------

; There are eight unary functions to retrieve the first eight elements.
;  first: list -> any
;  second: list -> any
;  third: list -> any
;  ...
;  eighth: list -> any

(define our-list (list 104 "hello there" true 0 1 2 3))
(check-expect (length our-list) 4)
; So only ‘first’ ‘second’ ‘third’ and ‘fourth’ will work on ‘our-list’.
(check-expect (first our-list) (list-ref our-list 0))
(check-expect (second our-list) (list-ref our-list 1))
(check-expect (third our-list) (list-ref our-list 2))
(check-expect (fourth our-list) (list-ref our-list 3))
; (seventh our-list): error

; Retrieving an Element by Index
; -----------------------------
; List elements in CS are usually indexed starting from 0.

(list 104 "hello there" true 0 1 2 3)
; 0 1 2 3

; The binary function ‘list-ref’ takes a list and an index.
;  list-ref: list number -> any
;  It produces the element of the list at that index.

(check-expect (list-ref our-list 0) 104)
(check-expect (list-ref our-list 0) (first our-list))
(check-expect (list-ref our-list 1) "hello there")
(check-expect (list-ref our-list 1) (second our-list))
(check-expect (list-ref our-list 2) true)
(check-expect (list-ref our-list 2) (third our-list))
(check-expect (list-ref our-list 3) )
(check-expect (list-ref our-list 3) (fourth our-list))
; Producing a Sequence of Numbers
; -----------------------------
; The ternary function `range` takes three numbers, and produces a
; list of numbers.
;
; range : number number number -> list{of numbers}
;
; The list of numbers it produces:
; • starts from the first number
; • counts by repeatedly adding the third number to it
; • goes up to but NOT including the second number

; List of numbers 1 up to but not including 27, counting by 2.
(check-expect (range 1 27 2)
  (list 1 3 5 7 9 11 13 15 17 19 21 23 25))
;
; List of numbers it produces:
; • starts from the first number
; • counts by repeatedly adding the third number to it
; • goes up to but NOT including the second number

; Making a Function to Produce the Last Element of a List
; -----------------------------------------------

; Let's make a unary function `last`, with contract
; last: list -> any
; that produces the last element of a non-empty list.

(check-expect (last (list 104 "hello there" true ))
  (list-ref (list 104 "hello there" true ) 3))

; The last element of a list has index one less than the length of the list.

; Record the expected behaviour on a concrete example we can see.
; Depending on one's point of view [the creator of the function, or a
; reader/user of it] this can be considered a correctness check and/or
; checked example documentation.

(check-expect (last (list 104 "hello there" true ))
  (list-ref (list 104 "hello there" true ) 3))
; Express what we did, more explicitly, more generically/abstractly:

(check-expect (last (list 104 "hello there" true)) )

(list-ref (list 104 "hello there" true) )

(- (length (list 104 "hello there" true )) 1)))

; An example encouraging the use of racket to help us as much as possible.
; List of numbers from 123, counting by 7, up to but not including 456.
(define a-range (range 123 456 7))
(check-expect (last a-range)
  (list-ref a-range (- (length a-range) 1)))

; Now we have an expression for the body of our function.
; But use a more generic name than ‘a-range’ for the place-holder/parameter,
; for the sake of people reading and using the function.

; last : list -> any
; The last element of non-empty list a-list.
(define (last a-list)
  ; From the ‘check-expect’: (list-ref a-range (- (length a-range) 1)) .
  ; But replace ‘a-range’ with the place-holder name ‘a-list’ we chose.
  (list-ref a-list (- (length a-list) 1)))

(last a-range)

; People in CS, and half of Mathematicians, count from 0.
; Also, people in CS like to specify ranges excluding the end of the range.
; It has some good reasons in more complex situations.

; One property is that [when counting by 1] the number of numbers is the
; difference between the end and the start of the range:
(check-expect (length (range 123 456 1))
  (- 456 123))

; Another property [which doesn't seem as worthwhile initially, and maybe
; even odd] is that matching the end of one to the start of the next doesn't
; duplicate a number when appending ranges.

; The function ‘append’ takes lists, and produces a single list containing
; all the elements.
; append : lists -> list

(append (list "cat" false) (list 104 "hello there" true) (list 123))

(check-expect (append (range 1 10 1) (range 10 20 1)) (range 1 20 1))