




## Ejercicios para resolver en papel


### 6 Naturales

En los ejercicios de esta práctica **NO UTILICE** equal? ni operadores numéricos como +, -, \*, /, =, >=, <=, etc.

 **Ejercicio 1\*.** Considere la función intervalo de la **Práctica 6.Ejercicio. 6.**

De una nueva definición de esta función **SIN USAR** reverse.


```
;intervalo: Natural -> List(Natural)
(check-expect (intervalo 0) empty)
(check-expect (intervalo 3) (list 1 2 3))
```

 **Ejercicio 2.** Complete el diseño de la función igualNat?.


```
;igualNat?: Natural Natural -> Boolean
;Esta función implementa la igualdad entre naturales.
(check-expect (igualNat? 0 0) #t)
(check-expect (igualNat? 0 4) #f)
(check-expect (igualNat? 5 0) #f)
(check-expect (igualNat? 7 3) #f)
(check-expect (igualNat? 7 7) #t)
```

 **Ejercicio 3.** Complete el diseño de la función mayorNat?.

```
;mayorNat?: Natural Natural -> Boolean
;Esta función implementa la relación de orden '>' entre dos naturales.
(check-expect (mayorNat? 0 0) #f)
(check-expect (mayorNat? 3 3) #f)
(check-expect (mayorNat? 5 3) #t)
(check-expect (mayorNat? 3 5) #f)
```



 **Ejercicio 4.** Complete el diseño de la función difNat. *Sugerencia: utilice las funciones definidas previamente.*

```
;difNat: Natural Natural -> Natural
;Esta función recibe dos naturales y devuelve la diferencia entre ellos.
(check-expect (difNat 0 0) 0)
(check-expect (difNat 3 3) 0)
(check-expect (difNat 11 5) 6)
(check-expect (difNat 5 11) 6)
```

 **Ejercicio 5.** Complete el diseño de la función contarElem.


```
;contarElem: List(Any) -> Natural
;Esta función recibe una lista y devuelve la cantidad de sus elementos.
(check-expect (contarElem (list "a" "b" 5 #t)) 4)
(check-expect (contarElem '()) 0)
```

---


 **Ejercicio 6.** Considere la función `contarElem` ( **Ejercicio. 5**). ¿Se podría definir la función del siguiente modo?

```
(define (contarElem l)
  (foldr add1 0 l))
```

En caso de no ser posible, justifique su respuesta y de su versión de la función usando `foldr`.

 **Ejercicio 7.** Complete el diseño de la función `nElementos?`.

```
;nElementos?: Natural List(Any) -> Boolean
;Esta función recibe un natural n y una lista, y devuelve #t si la lista tiene
; exactamente n elementos; en caso contrario, devuelve #f.
(check-expect (nElementos? 3 empty) #f)
(check-expect (nElementos? 0 empty) #t)
(check-expect (nElementos? 3 (list "a" #t 8)) #t)
(check-expect (nElementos? 5 (list "a" #t 8)) #f)
(check-expect (nElementos? 2 (list "a" #t 8)) #f)
(check-expect (nElementos? 0 (list "a" #t 8)) #f)
```

 **Ejercicio 8.** Complete el diseño de la función `transformaN`.

```
;transformaN: Natural (X -> Y) List(X) -> List(X|Y)
;Esta función recibe un natural n, una función (X -> Y) y una lista de tipo X
; y devuelve la lista habiendo aplicado la función a los primeros n elementos.
(check-expect (transformaN 3 number->string (list 3 4 5 6 7))
  (list "3" "4" "5" 6 7))
(check-expect (transformaN 0 sqr (list 3 4 5)) (list 3 4 5))
(check-expect (transformaN 2 sqr (list 3 4 5 )) (list 9 16 5))
(check-expect (transformaN 3 sqr (list 3 4 5 )) (list 9 16 25))
(check-expect (transformaN 5 sqr (list 3 4 5)) (list 9 16 25))
(check-expect (transformaN 5 sqr empty) empty)
```