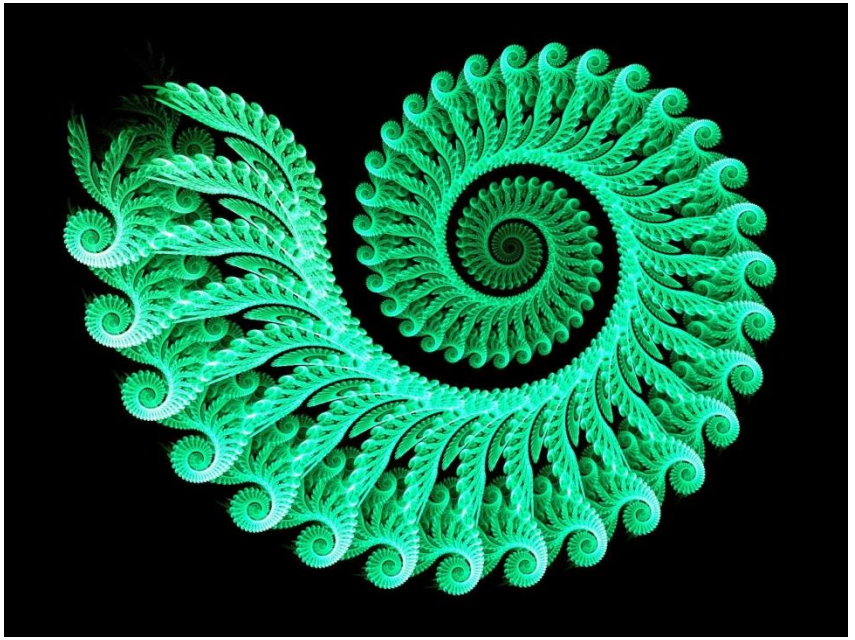
Listas en Racket

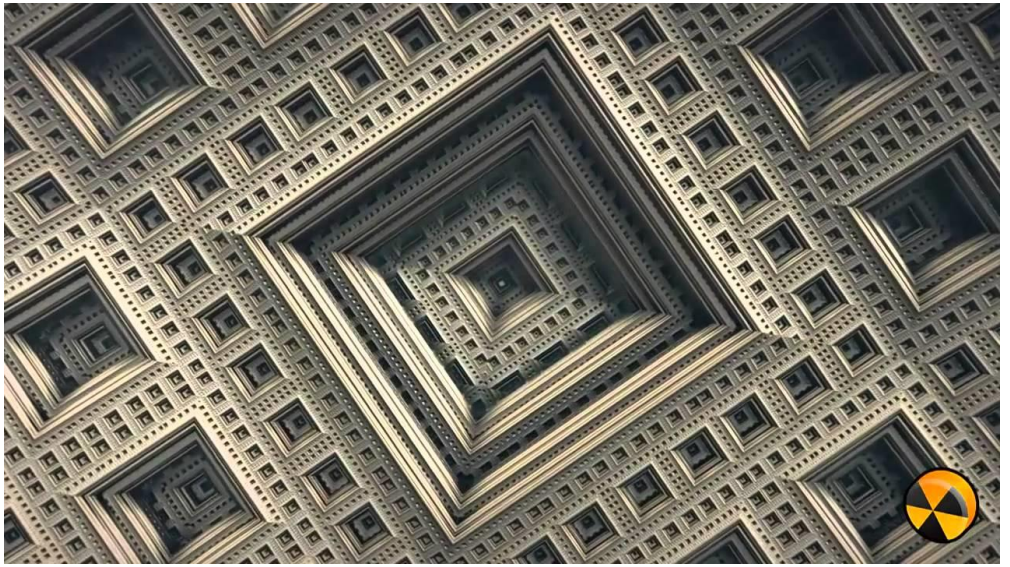
DATOS AUTO REFERENCIADOS EN LA VIDA REAL

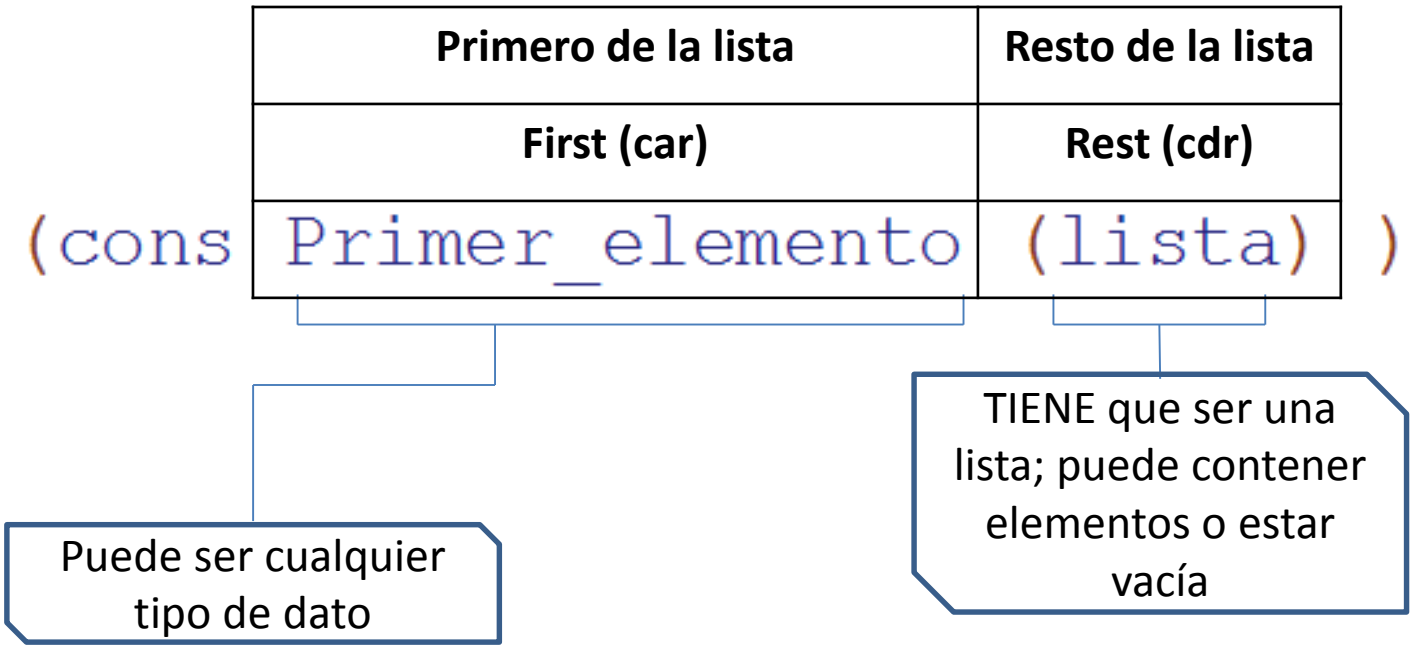
- Matrioskas o Muñecas Rusas

DATOS AUTO REFERENCIADOS EN LA VIDA REAL

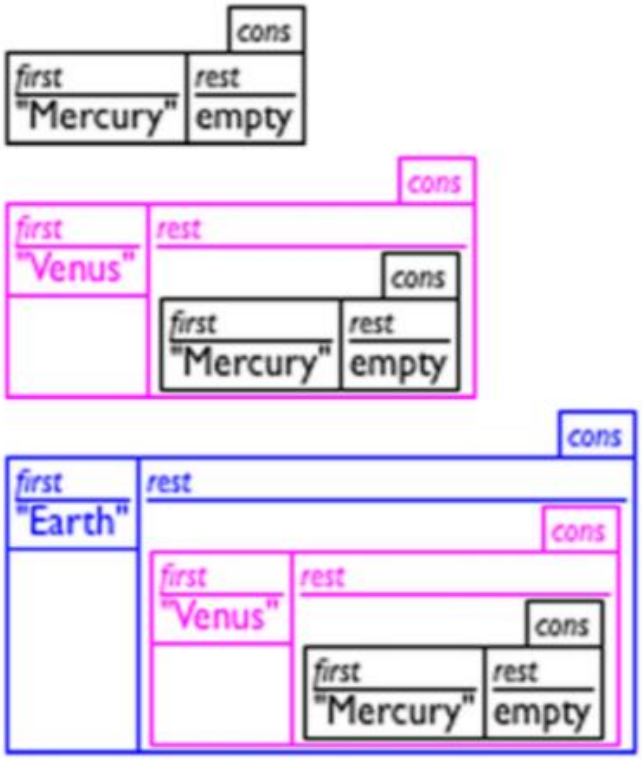
- Fractales






Primero de la lista
Resto de la lista
First (car)
Rest (cdr)
(cons Primer_elemento (lista))
Puede ser cualquier tipo de dato
TIENE que ser una lista; puede contener elementos o estar vacía

```
(cons "Mercury"
      empty)

(cons "Venus"
      (cons "Mercury"
            empty))

(cons "Earth"
      (cons "Venus"
            (cons "Mercury"
                  empty)))
```



```
(cons "Earth"
      (cons "Venus"
            (cons "Mercury"
                  empty)
            )
      )

   '("Earth"
>    "Venus"
     "Mercury")
```

OPERACIONES CON LISTAS

- Podemos preguntar si una lista está vacía usando el predicado ***empty?***

```
> (empty? empty)
#t
> (empty? 5)
#f
> (empty? "Hello world")
#f
> (empty? (cons 1 empty))
#f
```

OPERACIONES CON LISTAS

- Podemos crear una lista con la palabra reservada ***cons***

```
> (cons 1 (cons 2 (cons 3 empty)))
'(1 2 3)
> (cons (vector "a" "b" "c") empty)
'(#("a" "b" "c"))
> (cons empty (cons "hola" empty))
'(() "hola")
```

OPERACIONES CON LISTAS

- Podemos crear una lista con la palabra reservada ***cons***

```
> (cons 1 (cons 2 (cons 3 empty)))
'(1 2 3)
> (cons (vector "a" "b" "c") empty)
'(#("a" "b" "c"))
> (cons empty (cons "hola" empty))
'(() "hola")
```

OPERACIONES CON LISTAS

- Podemos crear una lista de manera abreviada con la palabra reservada ***list*** o ***‘***

```
> (list 1 2 3)
'(1 2 3)
> (list (make-vector 3) 1)
'(#(0 0 0) 1)
> (list empty)
'(())
> '('a 'b 'c)
'('a 'b 'c)
> '("c" "a" "n" "c" "e" "l" "a" "r")
'("c" "a" "n" "c" "e" "l" "a" "r")
> '(empty)
'(empty)
```

OPERACIONES CON LISTAS

- Podemos crear una lista de manera abreviada con la palabra reservada ***list*** o ***‘***

```
> (list 1 2 3)
'(1 2 3)
> (list (make-vector 3) 1)
'(#(0 0 0) 1)
> (list empty)
'(())
> '('a 'b 'c)
'('a 'b 'c)
> '("c" "a" "n" "c" "e" "l" "a" "r")
'("c" "a" "n" "c" "e" "l" "a" "r")
> '(empty)
'(empty)
```

OPERACIONES CON LISTAS

- Podemos sacar el primer elemento de una lista con la palabra reservada ***first*** o ***car***

```
> (first (cons empty (cons "hola" empty)))
'()
> (car (cons empty (cons "hola" empty)))
'()
> (first (list 1 2 3))
1
> (car (list 1 2 3))
1
> (first '('a 'b 'c))
''a
> (car '('a 'b 'c))
''a
```

OPERACIONES CON LISTAS

- Podemos sacar el segundo elemento de una lista con la palabra reservada ***rest*** o ***dcr***

```
> (rest (cons empty (cons "hola" empty)))
'("hola")
> (cdr (cons empty (cons "hola" empty)))
'("hola")
> (rest (list 1 2 3))
'(2 3)
> (cdr (list 1 2 3))
'(2 3)
> (rest '('a 'b 'c))
'('b 'c)
> (cdr '('a 'b 'c))
'('b 'c)
```

OPERACIONES CON LISTAS

- Podemos juntar dos o más listas en una sola con la palabra reservada ***append***

```
> (append '('a 'b 'c) (list 1 2 3) (cons empty (cons "hola" empty)))
'('a 'b 'c 1 2 3 () "hola")
> (append (list (make-vector 3) 1) (cons (vector "a" "b" "c") empty))
'(#(0 0 0) 1 #("a" "b" "c"))
> (append '(empty) (cons 1 (cons 2 (cons 3 empty))))
'(empty 1 2 3)
```

RESUMEN DE PALABRAS RESERVADAS PARA LISTAS

- empty -> Lista vacía
- empty? -> Predicado que sólo reconoce ***empty***
- cons -> Constructor de listas (dos campos)
- cons? -> Predicado que sólo reconoce una instancia de ***cons***
- first (car) -> Selector que toma el primer elemento de una lista
- rest(cdr) -> Selector que toma el último elemento de una lista
- append -> Función para concatenar dos o más listas en una sola
- list ‘() -> Construye una lista de manera abreviada

CONSEJOS PARA MANEJAR LISTAS: DISEÑOS POR COMPOSICIÓN

- Las funciones auxiliares son útiles para resolver problemas que no pueden ser resueltos directamente a través de una sola función.
- Es posible realizar composición de funciones para que la función principal utilice funciones auxiliares que faciliten la solución del problema.
- En algunos casos, es necesario diseñar una función más general y definir la función principal como un uso específico de la función general.

ALGUNOS EJEMPLOS BÁSICOS CON LISTAS

Dada la siguiente lista de alumnos, construir una función que permita retornar el estudiante que hay en el puesto número 5.

```
(define lista_alumnos
(list "Pepito" "Juanito" "Jorgito" "Carlitos" "Josesito" "Perensejito"))
```

Ejemplo:

```
(quita_posición lista_alumnos)
> “Josesito”
```

ALGUNOS EJEMPLOS BÁSICOS CON LISTAS

```
1  #lang racket
2
3  (define lista_alumnos (list "Pepito" "Juanito" "Jorgito" "Carlitos" "Josesito" "Perensejito"))
4
5  ;Función principal
6
7  (define (quinta_posición lista_alumnos)
8    (aux1 lista_alumnos 1)
9    )
10
11 ;Función auxiliar
12
13 (define (aux1 lista n)
14   (if (= n 5)
15       (first lista)
16       (aux1 (rest lista) (+ 1 n))
17       )
18   )
19
20 (quinta_posición lista_alumnos)
21
```

```
Welcome to DrRacket, version 6.10 [3m].
Language: racket, with debugging; memory limit: 128 MB.
"Josesito"
>
```

ALGUNOS EJEMPLOS BÁSICOS CON LISTAS

Dada una lista de números, escribir una función que los sume y retorne la respuesta.
Ejemplo:

```
(define lista_num (list 1 2 3 4 5))

(sumar lista_num)
> 15
```

ALGUNOS EJEMPLOS BÁSICOS CON LISTAS

```
1  #lang racket
2
3  (define lista_num (list 1 2 3 4 5))
4
5  (define (sumar lista_num)
6    (if (empty? lista_num)
7        0
8        (+ (first lista_num)
9           (sumar (rest lista_num))
10          )
11       )
12   )
13
14 (sumar lista_num)
15
```

```
Welcome to DrRacket, version 6.10 [3m].
Language: racket, with debugging; memory limit: 128 MB.
15
>
```

ALGUNOS EJEMPLOS BÁSICOS CON LISTAS

Dada una lista de números, escribir una función que permita sumar los números pares de dicha lista

Ejemplo:

```
(define lista_num2 (list 1 2 3 4 5 6))

(sumar_pares lista_num2)
> 12
```

ALGUNOS EJEMPLOS BÁSICOS CON LISTAS

```
1  #lang racket
2
3  (define lista_num2 (list 1 2 3 4 5 6))
4
5  ;Función principal
6
7  (define (sumar_pares lista_num)
8    (cond
9      [(empty? lista_num) 0]
10     [(par (first lista_num)) (+ (first lista_num)
11                                 (sumar_pares (rest lista_num)))]
12     [else (sumar_pares (rest lista_num))]
13     )
14   )
15
16 ;Función auxiliar para hallar los pares
17
18 (define (par x)
19   (if (= 0 (remainder x 2))
20       #t
21       #f
22       )
23   )
24
25 (sumar_pares lista_num2)
```

```
Welcome to DrRacket, version 6.10 [3m].
Language: racket, with debugging; memory limit: 128 MB.
12
>
```

ALGUNOS EJEMPLOS BÁSICOS CON LISTAS

Una función puede retornar una lista como resultado.

Ejemplo:

Se desea calcular el salario semanal de todos los empleados de una empresa, el salario semanal de un empleado de dicha empresa se calcula mediante la multiplicación del número de horas semanales por $12.000, que es el valor de la hora.

Construir una función en Racket que reciba una lista con el número de horas de los empleados y retorne una lista con el salario a pagar por cada elemento de la lista.

```
(define lista_horas (list 2 3 5 1 7 9))
(salario lista_horas)
> '(24000 36000 60000 12000 84000 108000)
```

ALGUNOS EJEMPLOS BÁSICOS CON LISTAS

```
 1  #lang racket
 2
 3  (define lista_horas (list 2 3 5 1 7 9))
 4
 5  (define (salario lista_horas)
 6    (if (empty? lista_horas)
 7        empty
 8        (cons (* (first lista_horas) 12000)
 9              (salario (rest lista_horas)))
10        )
11    )
12
13  (salario lista_horas)
14
```

```
Welcome to DrRacket, version 6.10 [3m].
Language: racket, with debugging; memory limit: 128 MB.
'(24000 36000 60000 12000 84000 108000)
>
```

EJERCICIOS

- Escriba una función que reciba una lista y retorne su tamaño.
- Escribir una función que permita obtener el mayor entre una secuencia de números.
- Escribir una función que reciba una lista de números y retorne otra lista donde cada elemento es el triple de cada elemento en la lista original.
- Escribir una función que reciba un entero positivo N y retorne una lista con los valores 1!, 2!, . . . , N!.
- Escribir una función que reciba dos listas de igual tamaño y retorne una lista donde cada posición corresponde a la suma de los elementos en dicha posición en las listas recibidas como parámetro.

HALLAR EL IMC DE N PERSONAS CON LISTAS

```
;Estructura LISTA persona: (list peso altura)

(define (IMC3 lista)
  (if (empty? lista)
      empty
      (cons (/ (first (first lista)) (sqr (first (rest (first lista)))))
            (IMC3 (rest lista)))
      )
  )

;Lista con propósito de ejemplo
(define ejemplo (list (list 73 1.74) (list 80 1.70)
                      (list 65 1.58) (list 51 1.45)
                      (list 49 1.51) (list 44 1.52)))

(IMC3 ejemplo)
```

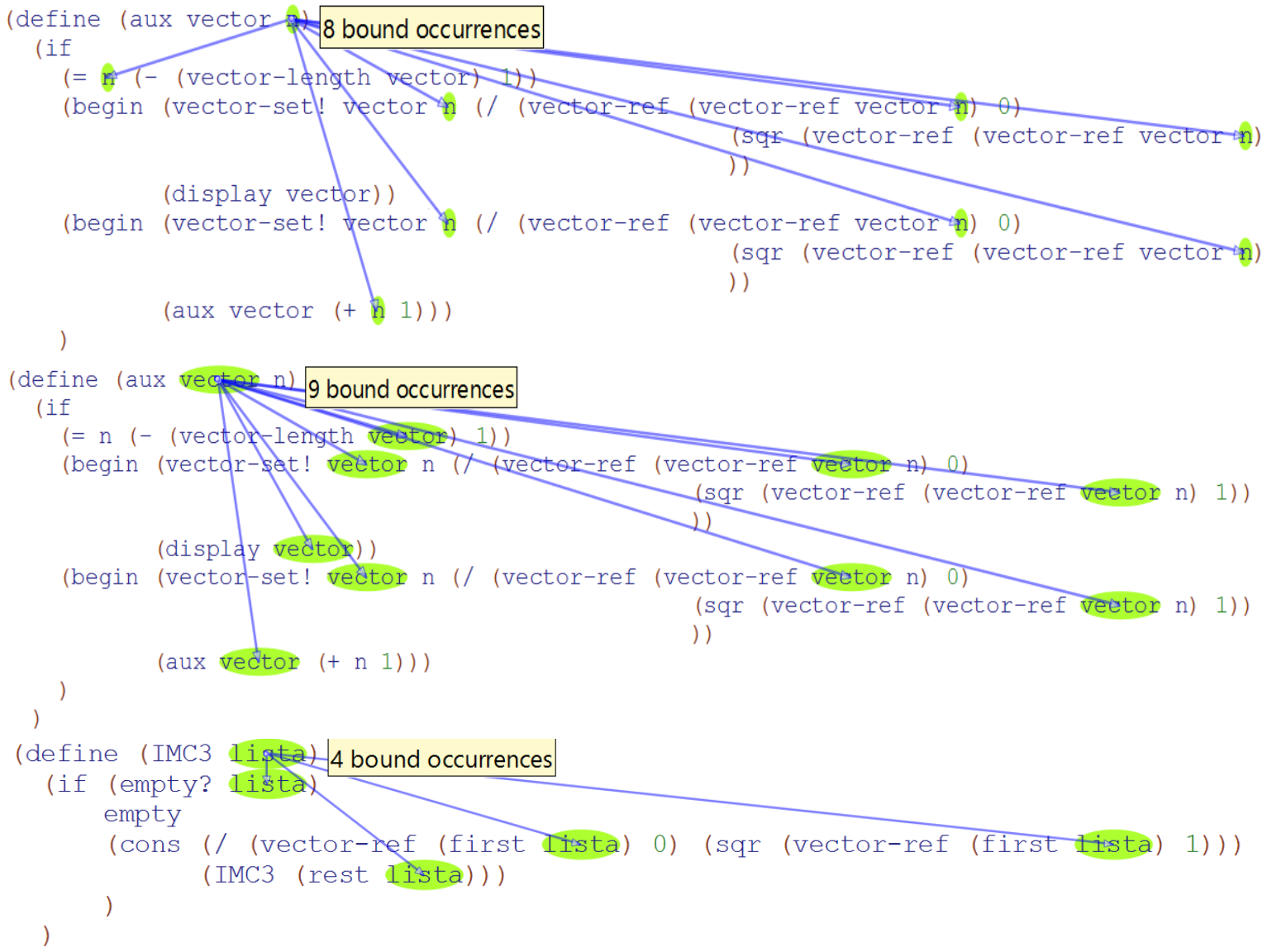
```
> '(24.111507464658473
  27.68166089965398
  26.037493991347535
  24.25683709869203
  21.490285513793253
  19.04432132963989)
```

OTRA FORMA DE HACERLO

```
;Estructura LISTA persona: (vector peso altura)

(define (IMC3 lista)
  (if (empty? lista)
      empty
      (cons (/ (vector-ref (first lista) 0) (sqr (vector-ref (first lista) 1)))
            (IMC3 (rest lista)))
      )
  )

;Lista con propósito de ejemplo
(define ejemplo (list (vector 73 1.74) (vector 80 1.70)
                      (vector 65 1.58) (vector 51 1.45)
                      (vector 49 1.51) (vector 44 1.52)))

(IMC3 ejemplo)
```

```
> '(24.111507464658473
  27.68166089965398
  26.037493991347535
  24.25683709869203
  21.490285513793253
  19.04432132963989)
```

COMPARACIÓN VECTORES - LISTAS



ORDENAMIENTO DE LISTAS

Una de las tareas más recurrentes a la hora de trabajar con listas es ordenarlas, para ello existen muchos métodos computacionales, tales como:

- Ordenamiento por inserción (Insertion sort)
- Ordenamiento burbuja (Bubblesort)
- Ordenamiento por mezcla (Merge sort)
- Ordenamiento con árbol binario (Binary three sort)
- Ordenamiento rápido (Quicksort)

EJEMPLO ORDENAMIENTO DE LISTAS: POR INSERCIÓN (INSERTION SORT) - DESCENDENTE

```
1  #lang racket
2  (define lista1 (list 1 4 3 5 2 7        ))
3
4  ;Función principal
5
6  ;; ordenar : lista-de-numero -> lista-de-numero
7  (define (ordenar lista)
8    (cond
9      [(empty? lista) empty]
10     [else (insertar (first lista)
11                     (ordenar (rest lista)))]
12     )
13   )
14
15 ; insertar : numero lista -> lista
16 ; crea una lista de números con número y lista
17 ; ordenada de mayor a menor, considerar que la
18 ; lista debe estar ordenada previamente.
19
20 (define (insertar numero lista)
21   (cond
22     [(empty? lista) (cons numero empty)]
23     [else (if (>= numero (first lista))
24               (cons numero lista)
25               (cons (first lista)
26                     (insertar numero (rest lista))))]
27     )
28   )
29
30 (ordenar lista1)
```

Language: racket, with debugging; memory limit: 128 MB.

EJEMPLO ORDENAMIENTO DE LISTAS: POR INSERCIÓN (INSERTION SORT) - ASCENDENTE

```
1  #lang racket
2  (define lista1 (list 1 4 3 5 2 7        ))
3
4  ;Función principal
5
6  ;; ordenar : lista-de-numero -> lista-de-numero
7  (define (ordenar lista)
8    (cond
9      [(empty? lista) empty]
10     [else (insertar (first lista)
11                     (ordenar (rest lista)))]
12     )
13   )
14
15 ; insertar : numero lista -> lista
16 ; crea una lista de números con número y lista
17 ; ordenada de mayor a menor, considerar que la
18 ; lista debe estar ordenada previamente.
19
20 (define (insertar numero lista)
21   (cond
22     [(empty? lista) (cons numero empty)]
23     [else (if (<= numero (first lista))
24               (cons numero lista)
25               (cons (first lista)
26                     (insertar numero (rest lista))
27     )
28   )
29
```

Welcome to DrRacket, version 6.10 [3m].
Language: racket, with debugging; memory limit: 128 MB.

¿PREGUNTAS?