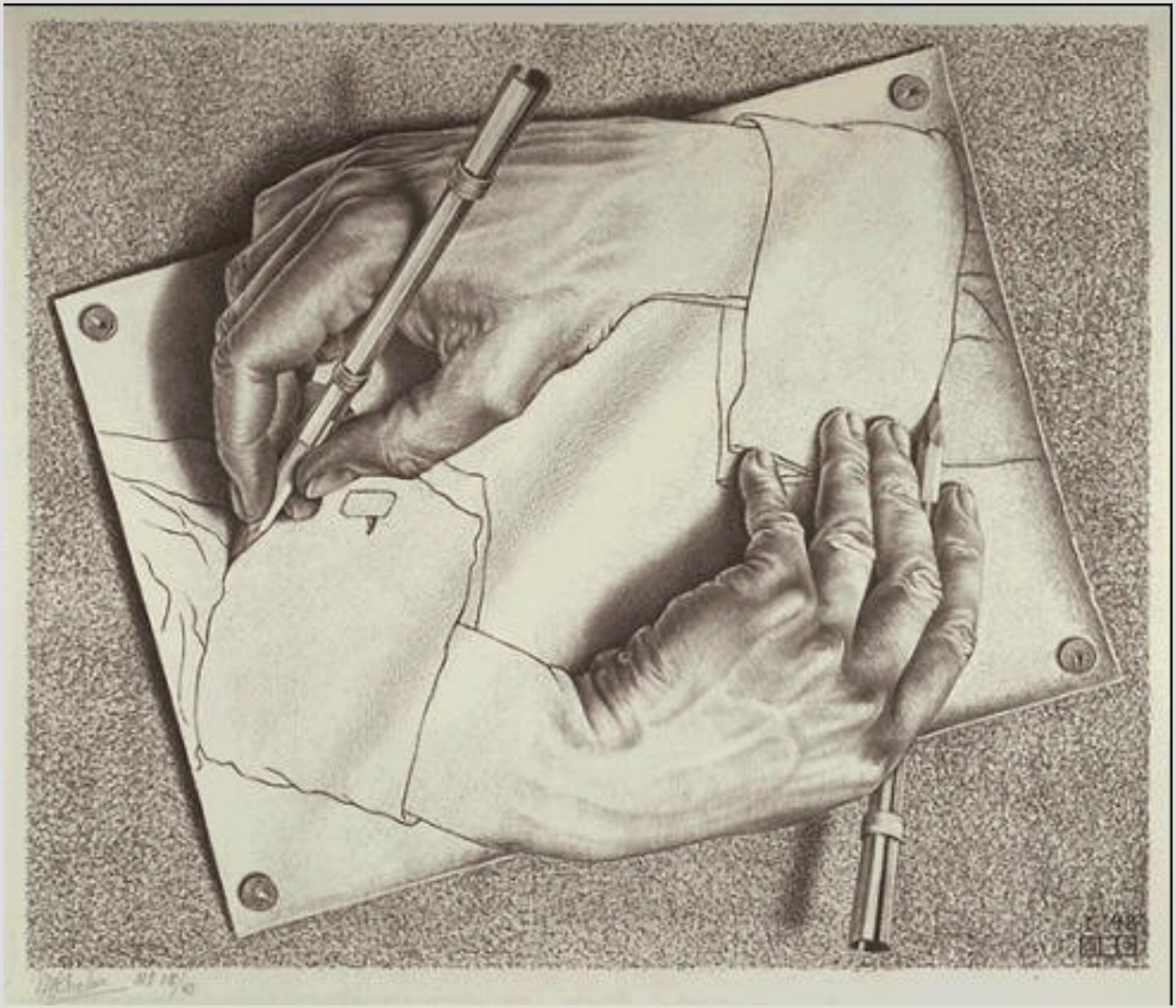
Fundamentos de programación

Funciones recursivas y listas

Fundamentos de programación
E.I.S.C.

Recursión



Introducción

- ***¿****qué es* ***Recursión****?* cierto diccionario mal intencionado dice lo siguiente:

Recursión (sustantivo): ver recursión

- ¿se imagina entonces qué es la ***recursión***?

  Es definir algo en términos de si mismo

Veamos algunos ejemplos: Fractales

Fractales

Sabia que....

- Un fractal es un objeto semi geométrico cuya estructura básica se repite a diferentes escalas. El término fue propuesto por el matemático Benoît Mandelbrot en 1975 y deriva del Latín fractus, que significa quebrado o fracturado. Muchas estructuras naturales son de tipo fractal.
- Dado que en un fractal “su estructura básica se repite a diferentes escalas”, es recursivo porque está definido en términos de sí mismo.

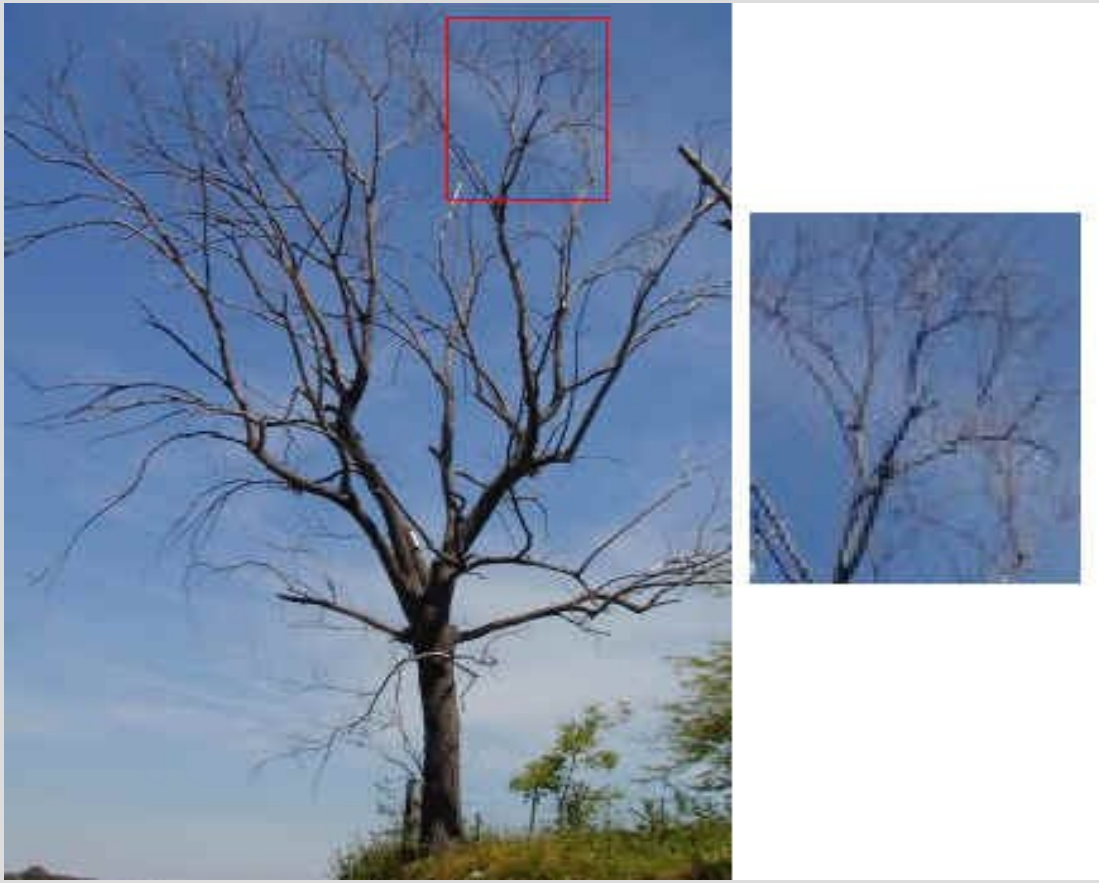
Fractales

Fractales en la naturaleza:

el “delicioso” brócoli

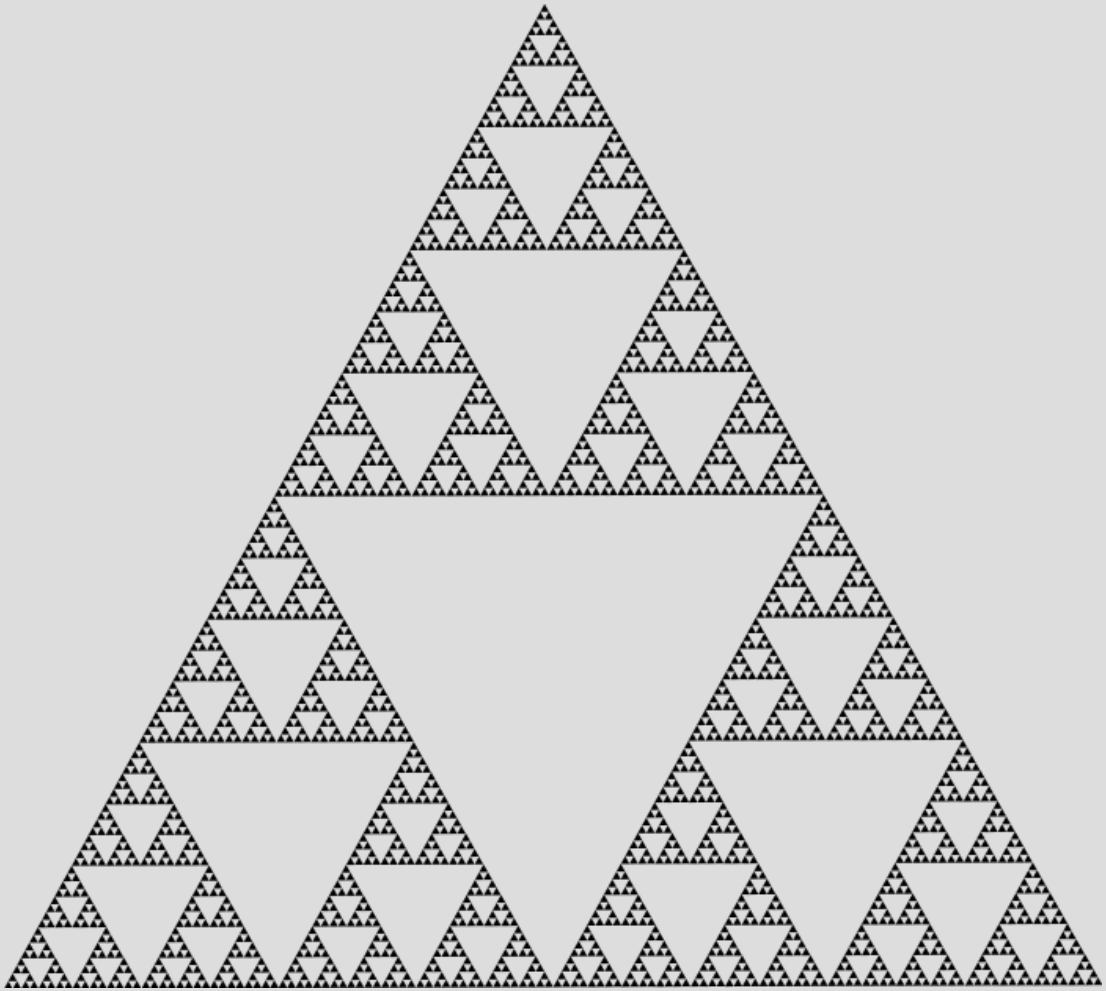
Fractales

Fractales en la naturaleza:



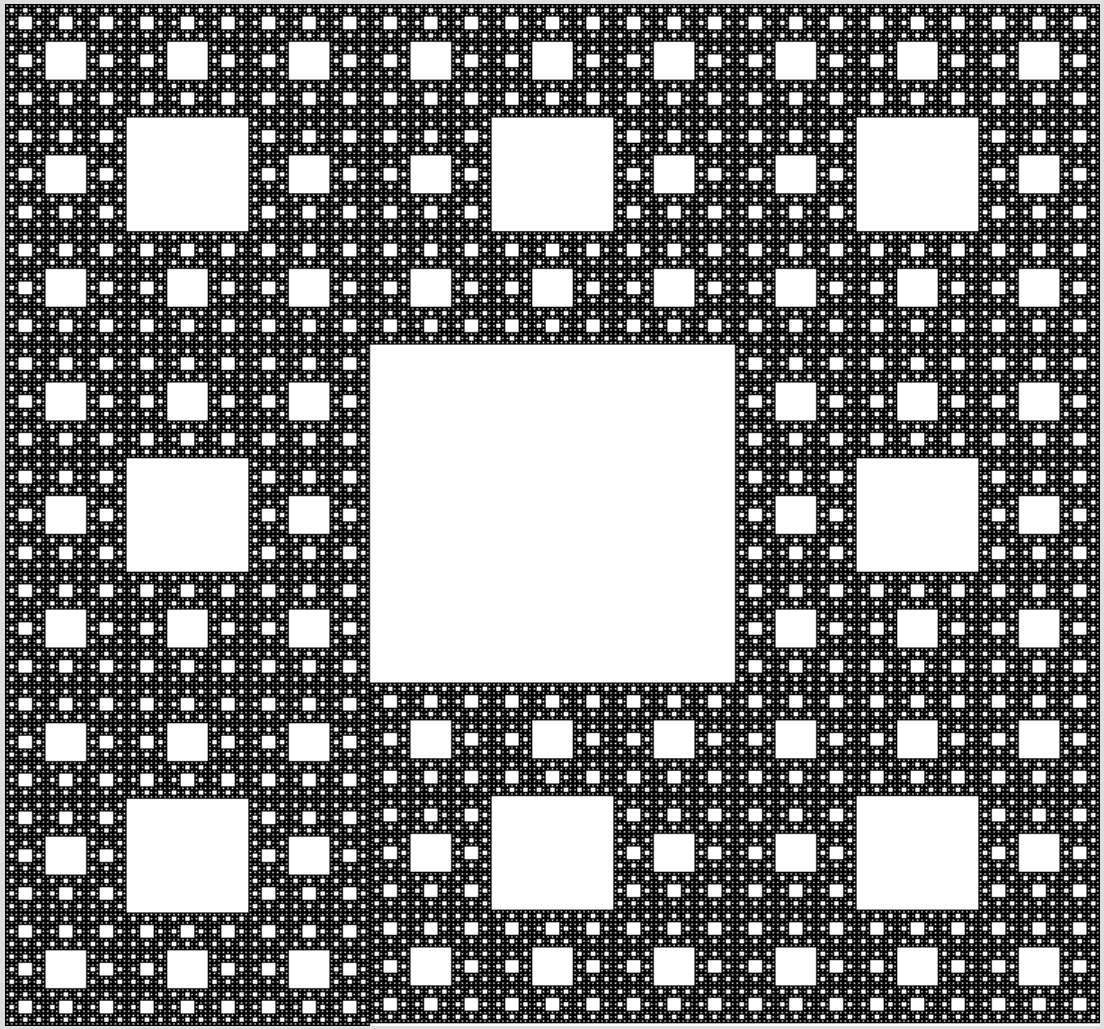
Fractales

- El triangulo de Sierpinski es un fractal que puede construirse a partir de cualquier triangulo

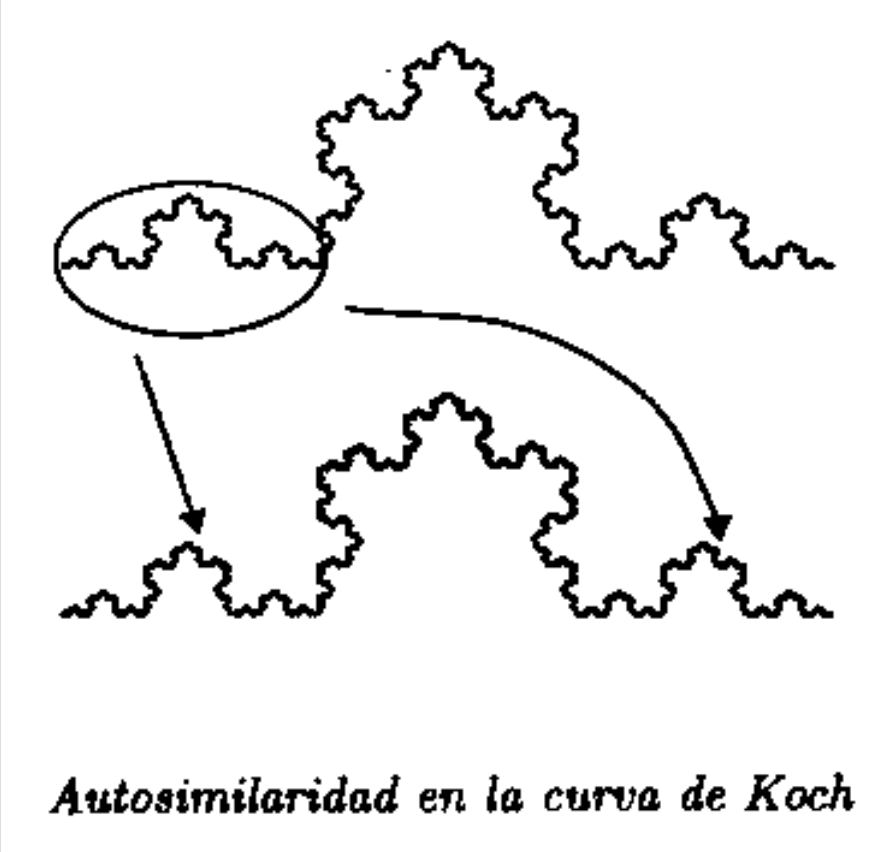


Fractales

- La alfombre de Sierpinski, hecha a base de cuadrados



Fractales



Autosimilaridad en la curva de Koch

- La curva de Koch es una de las más sencillas figuras fractales, y una de las primeras. Fue inventada por el matemático sueco Helge von Koch en 1906.

Bibliografía

- Más sobre Fractales:
  - http://es.wikipedia.org/wiki/Fractal
  - http://www.cientec.or.cr/matematica/fractales.html
  - http://www.public.asu.edu/~starlite/
- Litografías de M. C. Escher: waterfall, drawing hands:
  - http://www.mcescher.com/
- Imágenes tomadas de:
  - la naturaleza: http://vida-serendipity.blogspot.com/2007/07/desde-el-brocoli-la-luna.html
  - Curva y copo de koch: http://es.wikipedia.org/wiki/Copo_de_nieve_de_Koch

Recursividad

- En computación, la **recursión** se presenta en funciones y en tipos de datos
- Una **función es recursiva** si en su definición tiene al menos un llamado a sí misma
- Un **tipo de datos es recursivo** si está definido en términos de sí mismo.

Ejemplo:

el conjunto de potencias de 2: 1 2 4 8 16 32 64 ...

podemos definirlas como p(0) = 1 y p(n+1) = p(n)*2

Ejemplos de funciones recursivas

- Potencias de dos
- Factorial
- Serie de fibonacci
- Triangulo de Sierpinski

Ejemplos de funciones recursivas: potencias de dos

- Definición:

$$F(n) = \begin{cases} 1 & \text{si n=0} \\ 2 \cdot F(n-1) & \text{para n>0 en los naturales} \end{cases}$$

- En scheme:

```
(define (potencias-dos n)
  (cond
    [(= n 0) 1]
    [else
     (* 2 (potencias-dos (- n 1) ) )]))
```

Ejemplos de funciones recursivas: factorial

- Definición:

$$n! = \begin{cases} 1 & \text{si n=0} \\ n \cdot (n-1)! & \text{para n>0 en los naturales} \end{cases}$$

- en scheme:

```
(define (factorial n)
  (cond
    [(= n 0) 1]
    [else
     (* n (factorial (- n 1)))]))
```

Ejemplos de funciones recursivas: fibonacci

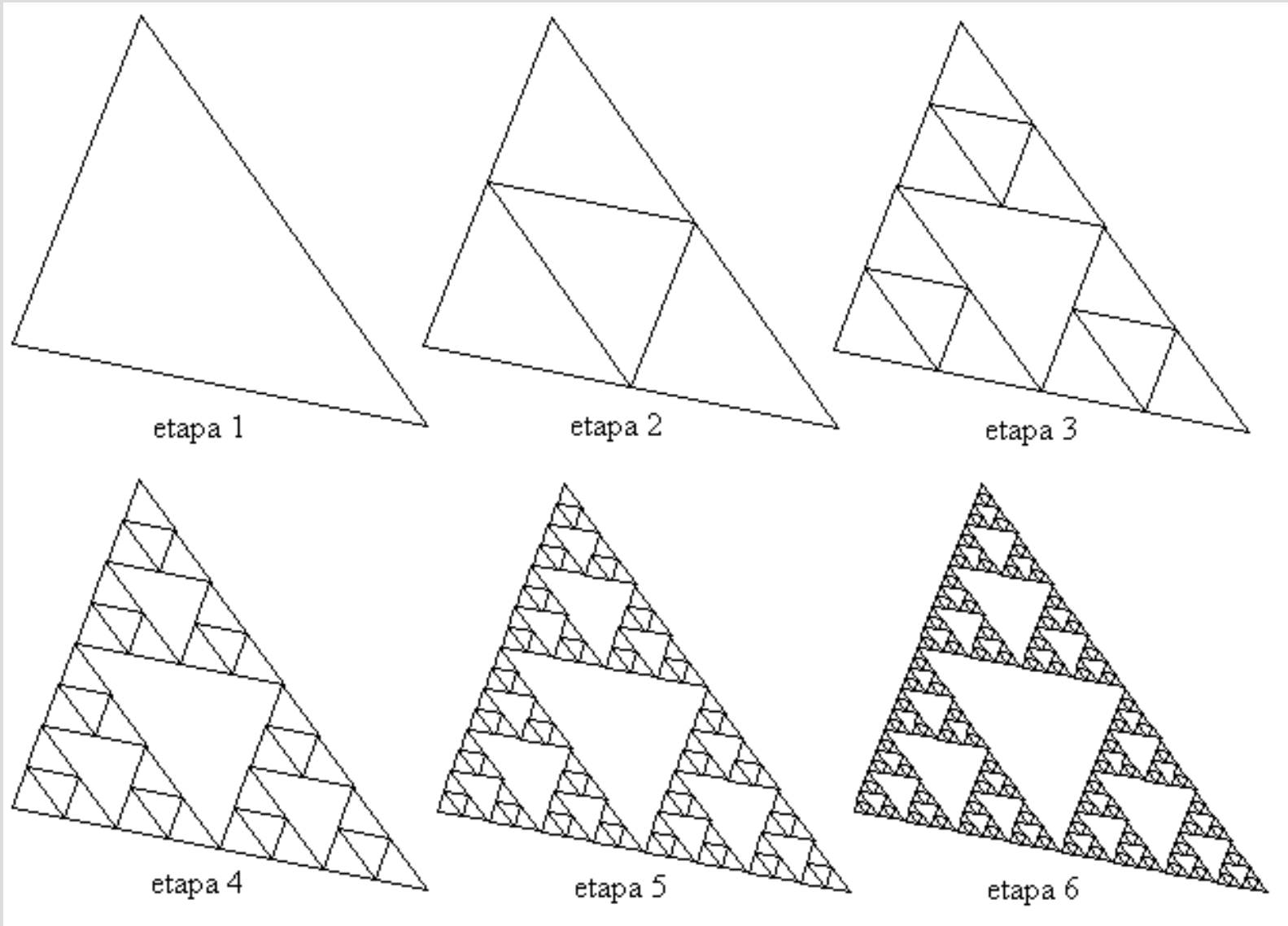
- Definición

$$F(n) = \begin{cases} 0 & \text{si } n = 0; \\ 1 & \text{si } n = 1; \\ F(n-1) + F(n-2) & \text{si } n > 1. \end{cases}$$

- En scheme:

```
(define (fibo n)
  (cond
    [(= n 0) 0]
    [(= n 1) 1]
    [else
     (+ (fibo (- n 1))
        (fibo (- n 2)))]))
```

Ejemplos de funciones recursivas: Triangulo



Funciones Recursivas

- Todas las funciones recursivas tienen las siguientes características:
  - tienen uno o más casos base: criterio de parada.
  - tienen un caso recursivo: se llama a la misma función

  Los ejemplos anteriores tienen caso base y caso recursivo?
  - Potencias de 2
  - Factorial
  - Fibonacci
  - Triangulo

Funciones Recursivas

- Cómo todas las funciones recursivas tienen la misma estructura, el cuerpo de la función (en scheme) será un condicional.
- ¿Cuantas condiciones debo poner?
  - Una por cada caso base
  - Una por el caso recursivo Ver ejemplos
- Se debe probar primero el caso base, porque si este tiene errores (lógicos) es posible que la función se quede haciendo un ciclo infinito

Ejercicio

- Desarrolle una función recursiva llamada *sumatoria* usando la siguiente definición:

$$\sum(n) = \begin{cases} n & \text{si n=0} \\ n + \sum(n-1) & \text{para n>0 en los naturales} \end{cases}$$

```
;sumatoria: numero (entero) -> numero
;calcula la suma de 1+2+3+4+5+....+n n es la entrada
;plantilla:
;(define (sumatoria n)
  (cond
    [(<caso base>) respuesta]
    [else (<llamado recursivo>)]) )
```

Ejercicio

- Análisis de datos:
  - caso base: ¿Cuál es la respuesta del programa cuando n=0?

    retorna cero
  - caso recursivo: ¿si NO se cumple que n=0, cual es la respuesta?

    se suman: n + el resultado de calcular la sumatoria de n-1

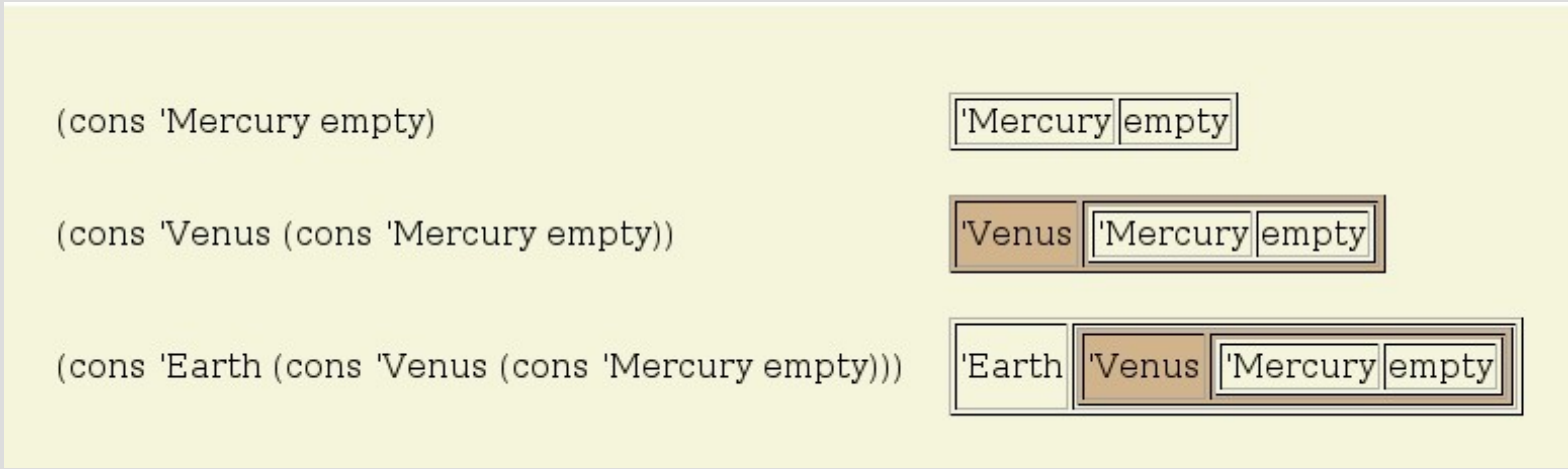
```
(+ n (sumatoria (- n 1)))
```

Ejercicio

- Ejemplos:
  - si n=0 el programa debe retornar 0
  - si n=3 el programa debe calcular 3+2+1+0
  - si n=154.567 el programa debe calcular.....???

Datos Recursivos

- Listas: una lista es un tipo de datos recursivo porque se define en términos de sí misma:
  - Una lista es (una de las dos):
    - empty (lista vacía)
    - (cons elemento lis) lista construida en donde elemento puede ser de cualquier tipo y lis es una lista.

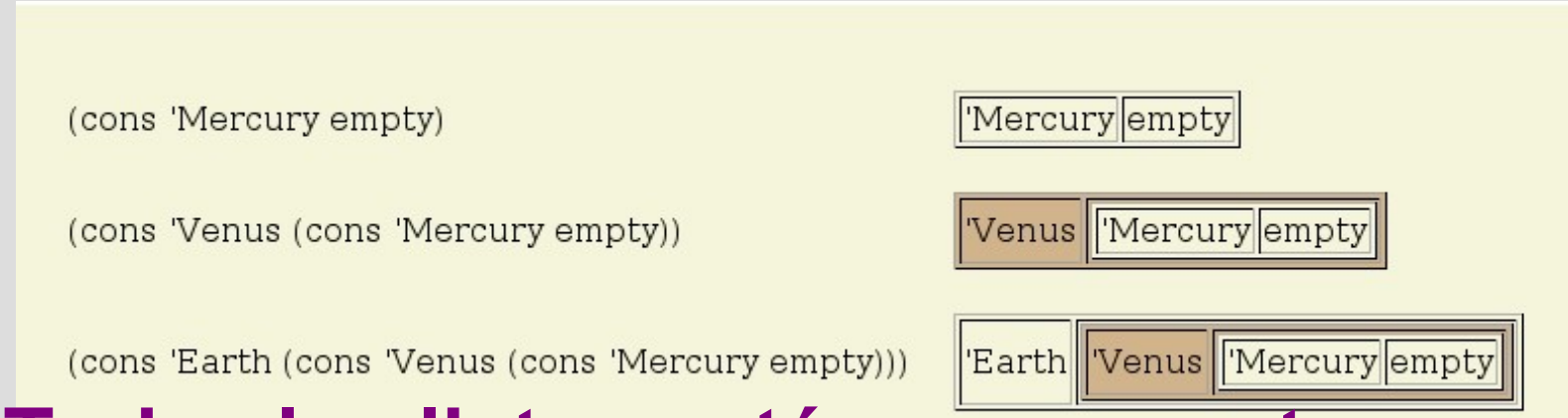


El operador CONS

- El operador cons nos permite construir listas. el operador cons se utiliza de la siguiente forma:

(cons cabeza cola)

En donde cabeza es un elemento y cola es una lista



Todas las listas están compuestas por cabeza y cola, excepto la lista vacía

Ejemplos

- lista vacía: empty
- lista de los números del 20 al 25: 20, 21, 22, 23, 24

```
(cons 20 (cons 21 (cons 22(cons 23 (cons 24 (cons 25
                                                empty))))))
```

- ¿Cómo se representa en scheme la lista con las vocales?

Operadores first y rest

- Similar a los selectores de las estructuras, las listas tienen dos operaciones que permiten consultar el contenido de la cabeza y de la cola
- first: retorna el primer elemento de la lista
- rest: retorna la cola de una lista (que es una lista)
- Ejemplo
  - (first (cons 'venus (cons 'mercury empty))) retorna 'venus
  - (rest (cons 'venus (cons 'mercury empty))) retorna (cons 'mercury empty)

Ejemplos:

- (first empty) ¿qué retorna?
- (rest empty) ¿qué retorna?
- (first `(cons 20 (cons 21 (cons 22(cons 23 (cons 24 (cons 25 empty))))))`) ¿qué retorna?
- (rest `(cons 25 empty)`)
- (first (rest `(cons 21 (cons 22(cons 23 (cons 24 (cons 25 empty)))))`)) ¿qué retorna?

Los números naturales

- Los números naturales son un caso especial de dato recursivo.
- Se pueden definir de la siguiente forma:
- Un número natural es (uno de los dos):
  - cero
  - (n+1) si n es natural
- Los números naturales nos permiten contar.

Ejemplos

- Desarrolle la función saludos, la cual tiene como entrada un número natural **n** y produce una lista que tienen **n** veces el símbolo 'hola
- Pregunta: ¿cuántas veces hay que repetir el símbolo?
- ¿cómo resolvería usted el problema usando lápiz y papel?
  - Contando, ¿cierto?
- Entonces debemos hacer en scheme un programa que repita una operación, o que cuente hasta n

Ejemplos

- Contrato:

```
saludos: n -> lista
```

- Propósito

; generar una lista que tenga el símbolo 'hola n veces

- Análisis de los datos:
  - Cuando n es cero, el programa retorna una lista vacía
  - De lo contrario???

Ejemplos

- ¿Qué es lo que deseamos hacer?
  - Hacer una lista
  - ponerle 'hola como elemento n veces.
- Ejemplos:
  - Si n es 1, el programa debe retornar la lista con un elemento:

```
(cons 'hola empty)
```

  - Si n es 3, el programa debe retornar la lista con 3 elementos:

```
(cons 'hola (cons 'hola (cons hola empty)))
```

Ejemplos

- Como no sabemos el valor de n, necesitamos un conjunto de instrucciones que se repitan hasta que se cumpla una condición.
- La condición es el criterio de parada, o el caso base.
- Las instrucciones que deben repetirse, deben dar solución al problema.
- La forma de repetirlas es por medio de llamados recursivos a la función

Ejemplos

```
(define (saludos n)
  (cond
    [(=n 0) empty]
    [else
      (cons 'hola (saludos (- n 1)))]))
```

¿Qué retorna este programa cuando n=0?

¿y cuando n=1?

¿y cuando n=3?

funciones que toman listas como entrada

- La mayoría de las funciones que tienen listas como entrada deben recorrer la lista.
  - sumar los elementos de una lista.
  - buscar un elemento en la lista.
  - contar los elementos de una lista.
- Y por cada elemento de la lista, se hace una operación
  - sumarlo, compararlo, o contarlo.
- Por eso debemos tener muy claro cuál es la operación y luego aplicarla a cada uno de los elementos de la lista

Funciones que toman listas como entrada

- Ejemplo: sea la lista
- `(cons 11 (cons 12 (cons 13 (cons 14 empty))))`
  - sumar todos los elementos:
    - (+ 11 (+ 12 (+ 13 (+ 14 (0)))) porque empty no cuenta!!
  - buscar si 16 está en la lista:
    - 11 es 16?, 12 es 16?....14 es 16?, vacío es 16? no está en la lista
  - contar cuantos elementos tiene
  - 11 (1), 12 (2), 13 (3), 14 (4), empty (no cuenta) , la lista tiene 4 elementos
- empty es un elemento de la lista, pero un elemento que no cuenta.

funciones que toman listas como entrada

- El recorrer una lista y aplicarle una operación se logra con first y rest.
- Con first se toma el primer elemento.
- Con rest se hace el llamado recursivo de la función con el resto de la lista.
- El caso recursivo tiene entonces:
  - la aplicación de la operación entre (first lista) con el resultado que retorna el llamado a la misma función pero con el resto de la lista.

funciones que toman listas como entrada

- Ejemplos:

```
(define lista
  (cons 11 (cons 12 (cons 13 (cons 14 empty)))))
```

  - sumar todos los elementos:
    - (+ (first lista) (suma (rest lista)))
  - buscar si 16 está en la lista:

```
(cond
[(= 16 (first lista) 'si está)]
[else (buscar 16 (rest lista))])
```

  - contar cuantos elementos tiene
  - (+ 1 (contar (rest lista)))

Funciones que toman listas en la entrada

- empty es la lista vacía, la función debe tener una respuesta en el caso en que la lista sea vacía
  - si la lista es vacía la suma es cero.
  - si la lista es vacía, ningún elemento está en la lista.
  - si la lista es vacía tiene cero elementos.
- El caso base tiene la respuesta del programa en el caso en que la lista sea vacía.

funciones que retornan listas

- ```
  (cond
   [(empty? lista) 0]
   [else <caso recursivo>])
  ```
- ```
  (cond
   [(empty? lista) 'no está]
   [else <caso recursivo>])
  ```
- ```
  (cond
   [(empty? lista) 0]
   [else <caso recursivo>]])
  ```
  ```

Ejemplos

- Función que suma todos los elementos de una lista:

```
(define (sum-lista lis)
  (cond
   [(empty? lis) 0]
   [else (+ (first lis)
            (sum-lista (rest lis)))]))
```

Ejemplos

- Función que cuenta cuantos elementos de una lista:

```
(define (contar lis)
  (cond
   [(empty? lis) 0]
   [else (+ 1 (contar (rest lis)))] ))
```

Ejemplo

- función que busca un elemento en una lista:

```
(define (buscar elem lis)
 (cond
   [(empty? lis) 'no_está]
   [else
    (cond
     [(= elem (first lis)) 'si_está]
     [else (buscar elem (rest lis))])]))
```

Funciones que retornan listas

- Las funciones que retornan listas tienen como operación principal cons
- y en el caso base retornan la lista vacía.
- Ejemplo: desarrolle la función que toma como entrada una lista de números y retorna la lista de cuadrados de estos números.

Ejemplo

- Contrato

```
;cuadrados: lista_de_numeros->
            lista_de_numeros
```

- Propósito:

```
; elevar al cuadrado cada elemento de una
  lista
```

- Ejemplos:
  - (cons 2 (cons 3 (cons 4 empty))) “debe retornar (cons 4 (cons 9 (cons 16 empty)))”

Ejemplos

- Plantilla:

```
;(define (cuadrados lista)
;  (cond
;   [(empty? lista) empty]
;   [else
;     <caso recursivo>   ]))
```

- Programa:

```
(define (cuadrados lista)
  (cond
   [(empty? lista) empty]
   [else
     (cons (expt (first lista) 2))
           (cuadrados (rest lista))   ]))
```