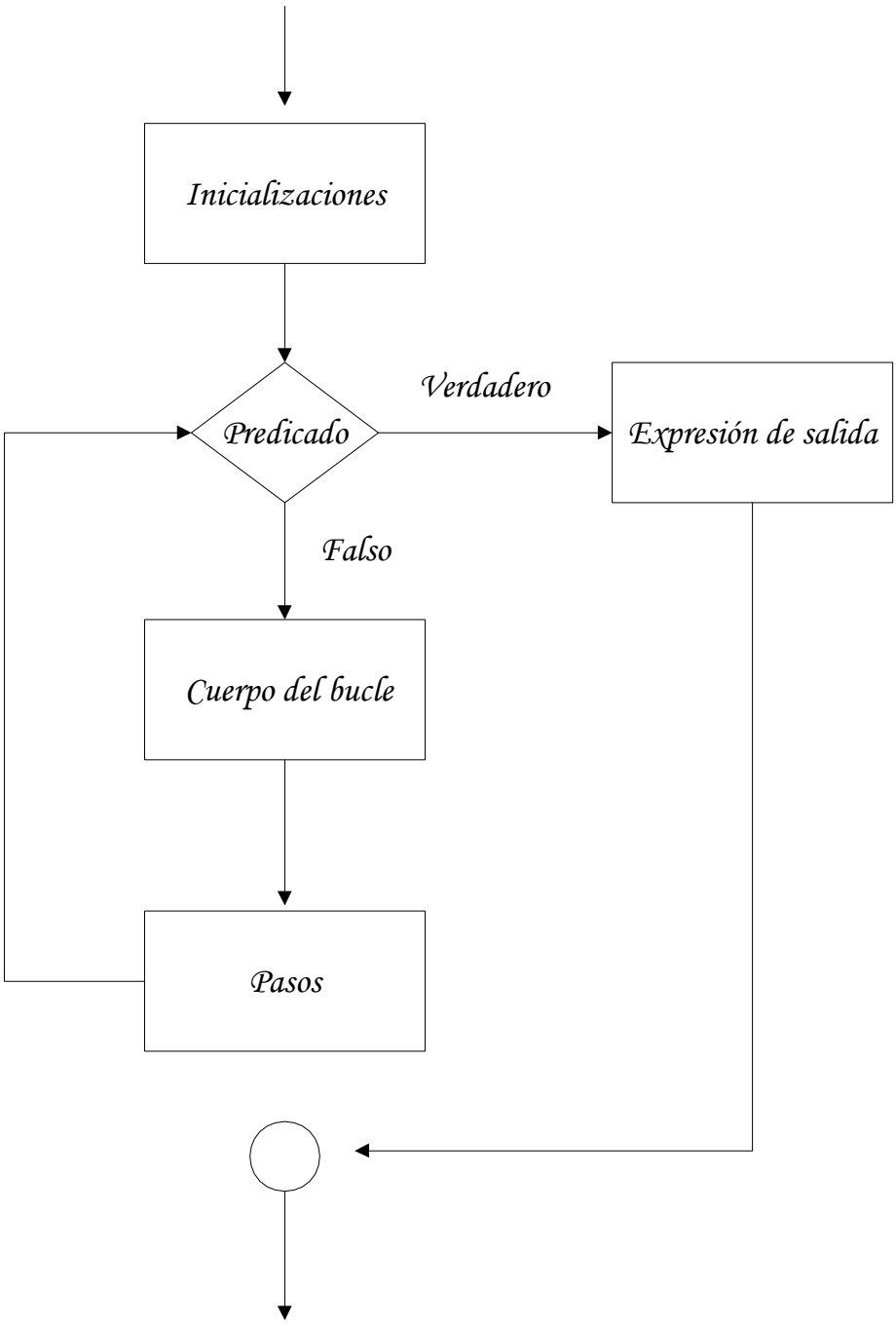
Sintaxis de la forma especial iterativa "do"

```
(do
  ;; Inicializaciones
   (
    (<variable 1> <inicialización 1> <paso 1>)
    (<variable 2> <inicialización 2> <paso 2>)
     …
    (<variable n> <inicialización n> <paso n>)
   )
  ;; Condición de salida y expresiones asociadas
  ( <predicado> <expresión> …)

  ;; Cuerpo del bucle
  <expresión> …
)
```

Diagrama de flujo de la forma especial "do"



Ejemplos de la forma especial "do"

- Factorial de un número: versiones iterativas

```
(define (fac-iter n)
        (if (or (= n 0) (= n 1))  1
                (do   ;; Inicializaciones
                        (
                         (i            n   (- i 1))
                        (resultado  1)          ;; no tiene "paso"
                        )
                       ;; Condición de salida
                       ((= i 1)    resultado)
                       ;;  Cuerpo del bucle
                       (set! resultado (* resultado i))
                )
        )
)
```

La siguiente función es equivalente a la anterior

```
(define (fac-iter-2 n)
        (if (or (= n 0) (= n 1))  1
                (do      ;; Inicializaciones
                        (
                         (i              n   (- i 1))
                        (resultado  1   (* resultado i))
                        )
                        ;; Condición de salida
                        ((= i 1)    resultado)
                        ;; no hay cuerpo del bucle
                )
        )
)
```

- Término de la sucesión de Fibonacci: versión iterativa

```
(define (fib-iter n)
    (if (or (= n 0) (= n 1))  1
        (do
            ;; Inicializaciones
            (
             (actual     1   (+ actual anterior))
             (anterior   1    actual)
             (i          n   (- i 1)
            )
            ;; Condición de salida
            ((= i 1)  actual)
            ;; no hay cuerpo del bucle
        )
    )
)
```

Recursión simple

- Factorial: versión recursiva

$$f(n) = n! = \begin{cases} 1 & Si \quad n = 0 \vee n = 1 \\ n \times (n-1)! = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1 & Si \quad n \geq 2 \end{cases}$$

```
(define (factorial n)
        (if (or (= n 0) (= n 1))
                        1
                        (* n (factorial (- n 1)))
        )
)

(factorial 4)                   ==> 24
```

- Potencia

$$a^b = \begin{cases} 1 & Si\ b = 0 \\ a \times a^{b-1} & Si\ b \geq 1 \end{cases}$$

```
(define (potencia a b)
        (if (= b 0)
                        1
                        (* a (potencia a (- b 1)))
        )
)

  (potencia  2 3)               ==> 8
```

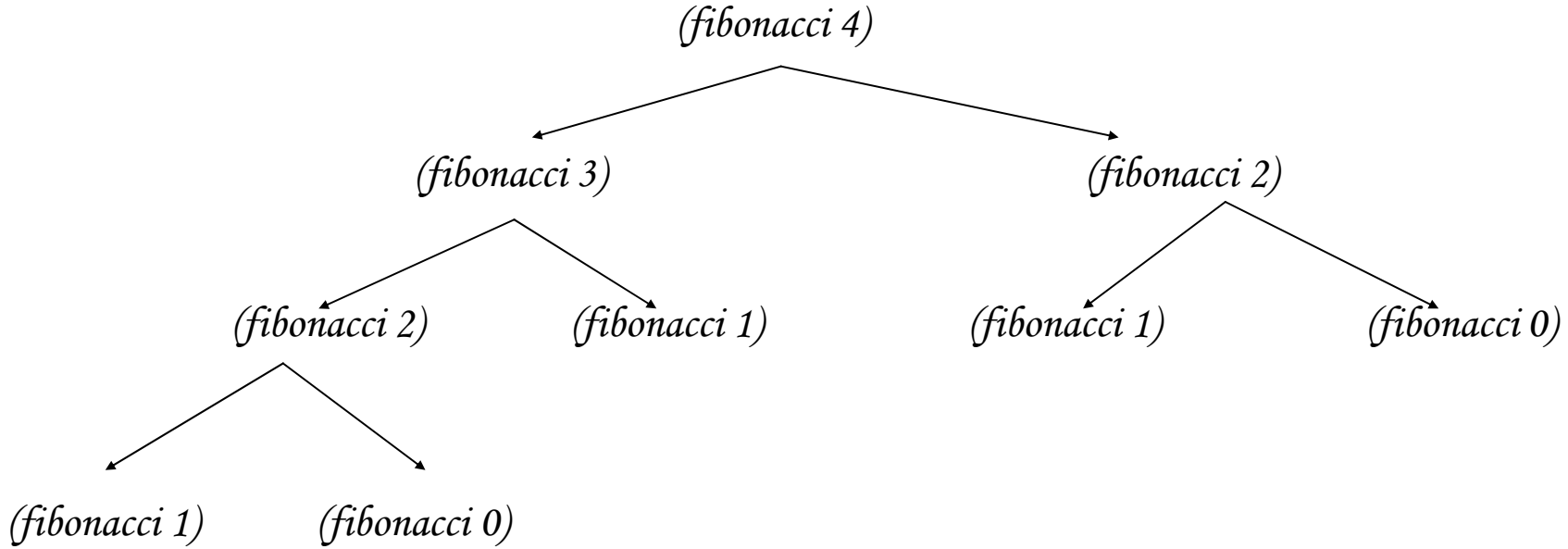
Recursión doble

- Término de la sucesión de Fibonacci: versión recursiva doble

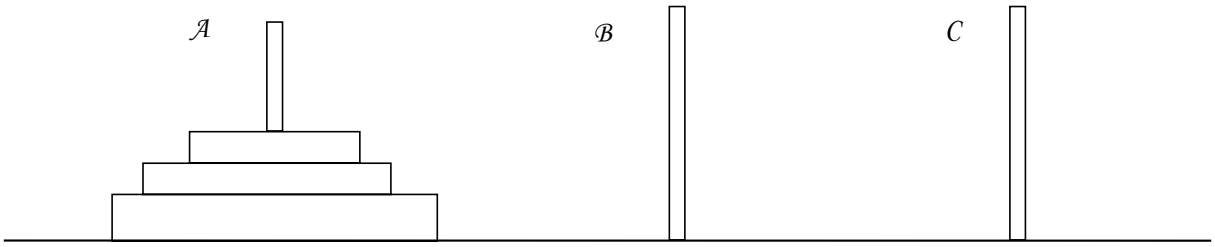
$$f(n)=\begin{cases} 1 & Si \;\; n=0 \vee n=1 \\ f(n-1)+f(n-2) & Si \;\; n\geq 2 \end{cases}$$

```
(define (fibonacci n)
        (if (or (= n 0) (= n 1))
                        1
                        (+
                            (fibonacci (- n 1))
                           (fibonacci (- n 2))
                        )
        )
)

(fibonacci 4)              ==> 5
```

- *Árbol de activación de (fibonacci 4)*

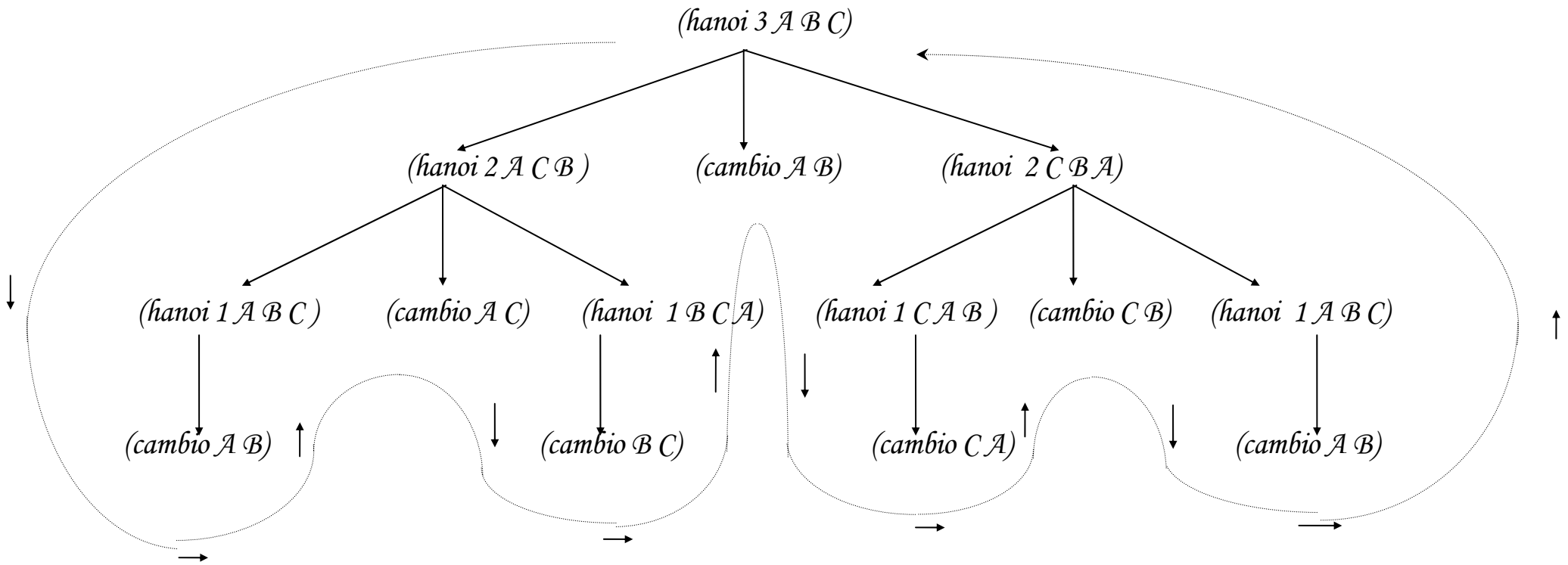


- *Las torres de Hanoi:*




```
(define (hanoi n a b c)
        (define (cambio x y)
                (display x)
                (display " --> ")
                (display y)
                (display "; ")
                1
        )
;; cuerpo de "hanoi"
        (if (= n 1) (cambio a b)
                (+    (hanoi (- n 1) a c b)
                      (cambio a b)
                      (hanoi (- n 1) c b a)
                )
        )
)
```

- *Árbol de activación de (hanoi 3 A B C)*



El recorrido en profundidad del árbol de activación genera la siguiente sucesión de cambios:

1.- A --> B 2.- A --> C 3.- B --> C 4.- A --> B

5.- C --> A 6.- C --> B 7.- A --> B Número de cambios: 7

Recursión de cola

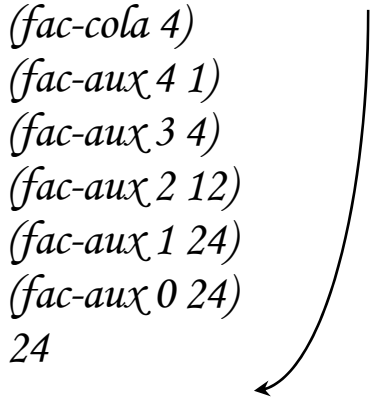
- Factorial de un número

```
(define (fac-cola n)
        (define (fac-aux n resultado)
                (if (= n 0)   resultado
                              (fac-aux (- n 1) (* n resultado))
                )
        )
        ; cuerpo de "fac-cola"
        (fac-aux n 1)
)
```

```
(fac-cola 4)            ==> 24
```

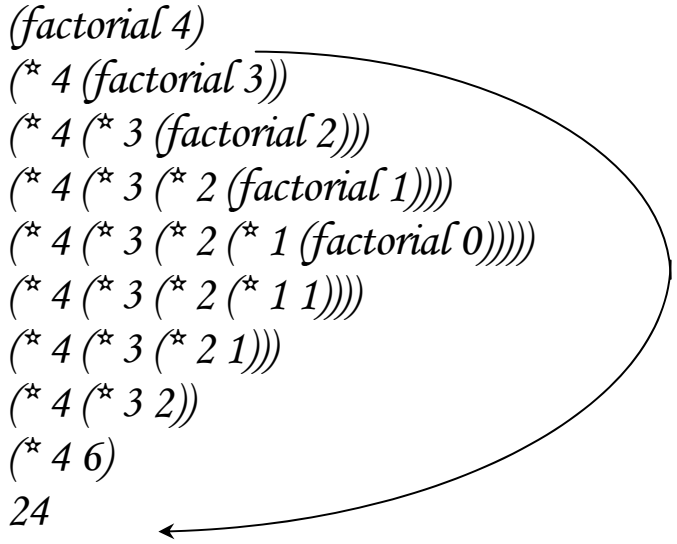
La ejecución de (fac-cola 4) desencadena los siguientes procesos:

Proceso "recursivo de cola"



La ejecución de (factorial 4) es la siguiente:

Proceso "recursivo lineal"



- *Raíz Cuadrada por el método de Newton:*

$$x_0 = 1, \quad x_{n+1} = \frac{\frac{x}{x_n} + x_n}{2} \quad si \ n \geq 1$$

$$\lim_{n \to \infty} x_n = y = \sqrt{x}$$

V.g.: $y = \sqrt{2}$ $x_0 = 1$

$$x_1 = \frac{\frac{2}{1} + 1}{2} = 1.5$$

$$x_2 = \frac{\frac{2}{1.5} + 1.5}{2} = 1.41\widehat{6}$$

$$x_3 = \frac{\frac{2}{1.41\widehat{6}} + 1.41\widehat{6}}{2} = 1.41421568$$

```
(define (raíz x)
        (define cota-error 0.001)
        (define (siguiente x y)
                (/ (+ (/ x y) y) 2. )
        )
        (define (raíz-aux x y)
           (if (< (abs (- (* y y) x)) cota-error)
                        y
                        (raíz-aux x (siguiente x y))
                )
        )
        ; cuerpo de "raíz"
        (raíz-aux x 1)
)
(raíz 2)                ==>  1.41421568
(raíz 256)              ==> 16.000003532
```

Forma especial "let con nombre"

La forma especial ***let*** *posee una variante que se denomina* ***let con nombre*** *(named let).*

Su sintaxis es la siguiente:

```
(let  <nombre-función>
        ;; Asignación inicial a los parámetros
        (
          (<parámetro 1>  <inicialización 1>)
          (<parámetro 2>  <inicialización 2>)
            …
          (<parámetro n>  <inicialización n>)
        )
     ;; cuerpo del "let con nombre",
     ;; donde se puede llamar  recursivamente a
     ;; <nombre-función>

      <expresión> …
)
```

- *Ejemplo: Factorial de un número.*

```
(define (factorial-let-nombre n)
    (let  fact-let
        ;; Asignación inicial a los parámetros
          (
           (i   n)
          ( resultado 1)
          )
          ;; cuerpo de let con nombre
          (if (or (= i 0) (= i 1))
              resultado
              ;; llamada recursiva
              (fact-let (- i 1) (* resultado i))
             )
    )
)
```

Funciones utilizadas como parámetros

- *Considérense las dos funciones siguientes:*
  - *Función que calcula la suma de los cubos de los enteros comprendidos entre "a" y "b":*

$$\sum_{a}^{b} n^3 = a^3 + (a+1)^3 + (a+2)^3 + \ldots + (b-1)^3 + b^3$$

```
(define (suma-cubos a b)
   (define (cubo x) (* x x x))
      (if (> a b)
            0
             (+ (cubo a)
                 (suma-cubos  (+ a 1) b)
             )
      )
)
```

(suma-cubos 1 3) ==> 36

- *Función que calcula la suma de una secuencia de términos de la siguiente serie que converge "muy lentamente" a* $\pi/8$

$$\frac{\pi}{8} = \sum_{1}^{\infty} f(n) = \frac{1}{1\times 3} + \frac{1}{5\times 7} + \frac{1}{9\times 11} + \dots$$

$$f(n) = \frac{1}{(4\,(n-1)+1) \times (4\,(n-1)+3)}$$

```
(define (suma-pi a b)
      (if (> a b)
            0
            (+ (/ 1.0 (* a (+ a 2.0)))
               (suma-pi (+ a 4.0) b)
            )
      )
)

(* 8.0 (suma-pi 1 10000))     ==> 3.14139265
```

- *Las dos funciones anteriores poseen el siguiente esquema subyacente:*

```
(define (<nombre> a b)
        (if (> a b)
                0
                 (+
                    (<término> a)
                    (<nombre> (<siguiente> a) b)
                 )
        )
)
```

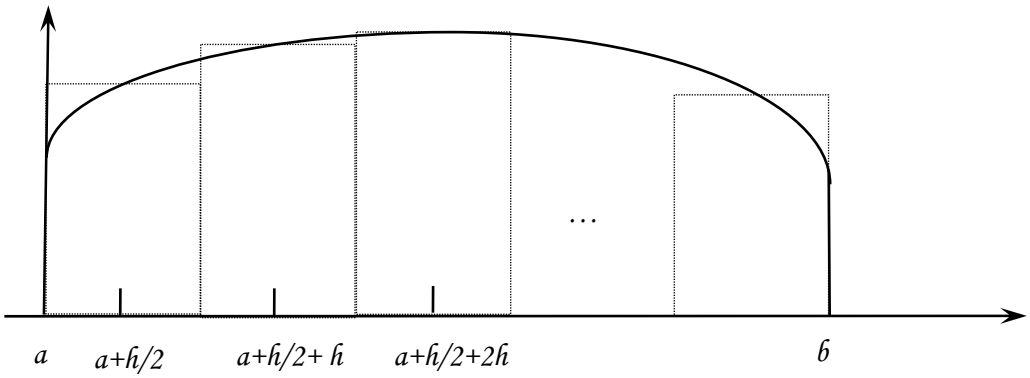
- *Se puede definir una nueva función que reciba como parámetros a las funciones "término" y "siguiente":*

```
(define (suma término siguiente a b)
        (if (> a b)    0
            (+
               (término a)
               (suma término siguiente  (siguiente a) b)
            )
        )
)
```

- Haciendo uso de la función **"suma"**, podemos calcular la suma de las series anteriores:

```
(define (suma-cubos-bis a b)
  ;; función que calcula el término de la serie
     (define (cubo x) (* x x x))

  ; ; función para obtener el "siguiente" elemento
     (define (suma-uno x)  (+ x 1.0))

  ;; cuerpo de suma-cubos-bis
     (suma  cubo  suma-uno  a  b)
)

(define (suma-pi-bis a b)
     ;; función que calcula el término de la serie
     (define (termino-pi  x) (/ 1.0  (* x (+ x 2.0))))
     )
   ; función para obtener el "siguiente" elemento
     (define (siguiente-pi  x)  (+ x 4))

   ; cuerpo de la función suma-pi-bis
     (suma termino-pi siguiente-pi a b)
)


;; Uso de la función suma-pi-bis

(* 8 (suma-pi-bis 1 10000))  ==> 3.14139265
```

- *Uso de la función "**suma**" para calcular el área contenida debajo de una curva mediante la aproximación por rectángulos centrados en los puntos medios de los intervalos.*

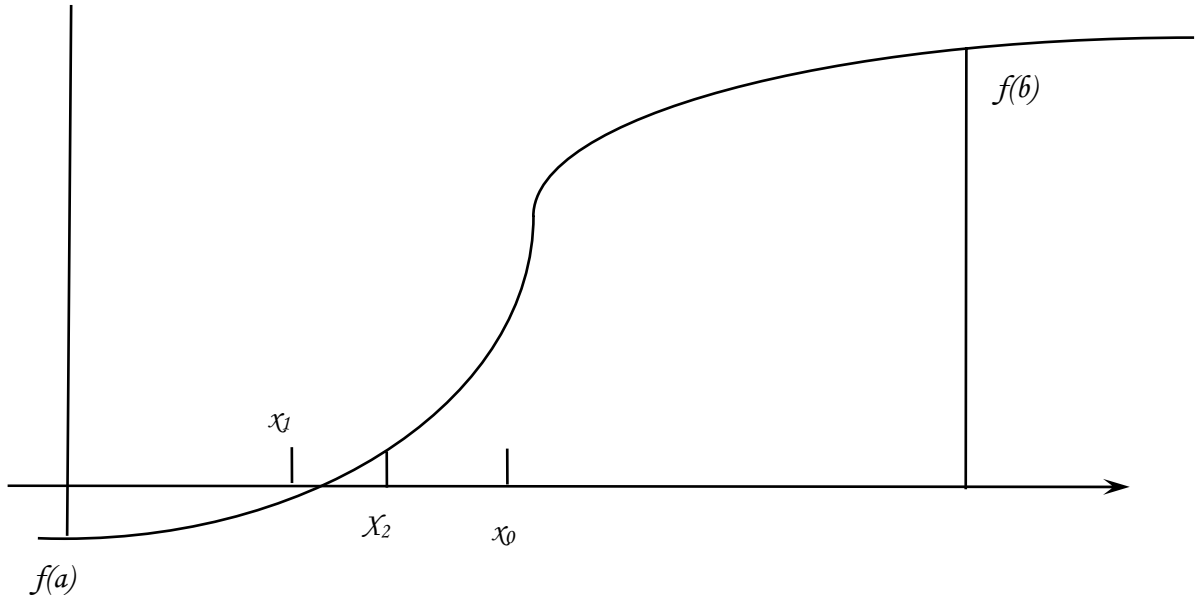


$$\int_a^b f(x) = \left(f(a+h/2) + f(a+h/2+h) + f(a+h/2+2h) + \ldots \right) h$$

"h" debe ser suficientemente pequeño y, además, se ha de cumplir que $a + h\,k + h/2 \geq b$

```
(define (integral f a b h)
 ;; función para obtener el siguiente elemento
 (define (sumar-h x) (+ x h))
 ;; cuerpo de integral
        (*
          (suma f sumar-h (+ a (/ h 2)) b)
          h
        )
 )

(define (f x) x)

(integral f 0 1 0.001)          ==> 0.50000000
      o
(integral (lambda (x) x) 0 1 0.001) ==> 0.50000000
```

- ***Bisección**: cálculo de la raíz de una función. Se exige que la función tome valores de diferente signo en los extremos del intervalo.*



```
(define (búsqueda f negativo positivo)
  (define (cercanos? x y)
        (< (abs (- x y)) 0.001)
  )
  (define (promedio x y)
              (/ (+ x y) 2.0)
  )
  (let
      ;; Asignación inicial a la variable
       ((mitad (promedio negativo positivo)))
       ;; Cuerpo de let
       (if (cercanos? negativo positivo)
          mitad
          (let   ((valor (f mitad)))
              (cond ((positive? valor)
                          (búsqueda f negativo mitad))
                    ((negative?  valor)
                          (búsqueda f mitad positivo))
                    (else mitad)
              )
          )
      )
  )

;; Ejemplo de llamada a la función búsqueda

(búsqueda (lambda (x) (- (* x x) 2)) 1 3)      ==> 1.41455078
```

```
(define (bisección f a b)
        (let ( (valor-a (f a))
               (valor-b (f b))
             )
          (cond ( (and (negative? valor-a)  (positive? valor-b))
                                              (búsqueda f a b)
                )
                ( (and (negative? valor-b)  (positive? valor-a) )
                                              (búsqueda f b a)
                )
                (else (display "los puntos tienen el mismo signo"))
          )
        )
)

;; Ejemplo de llamada a la función bisección

(bisección sin 2 4)                          ==> 3.14111328
```

Funciones devueltas como resultados

- **Derivada de una función**

$$f'(x) = Df(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

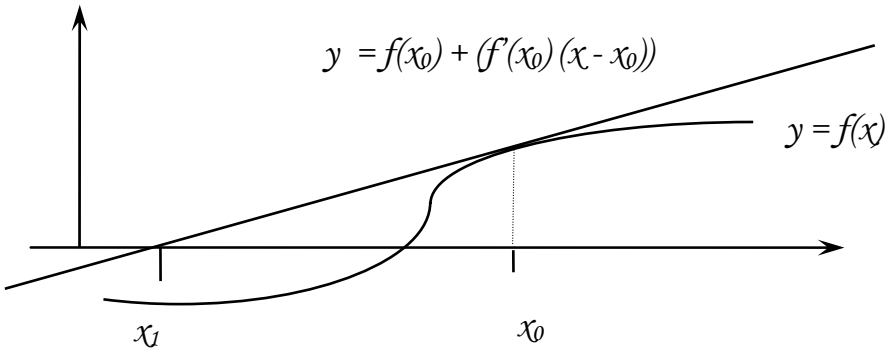
donde $\frac{f(x+h) - f(x)}{h}$

se denomina **"cociente incremental"**.

Se puede utilizar la forma especial **lambda** para obtener el cociente diferencial de una función

```
(define (cociente-incremental f h)
        (lambda (x) (/
                      (- (f (+ x h)) (f x))
                      h
                    )
        )
)

(define (cubo x) (* x x x))
((cociente-incremental cubo 0.001) 5)
        ==> 75.01500100
o
((cociente-incremental (lambda (x) (* x x x)) 0.001) 5)
        ==> 75.01500100
```

- **Método de Newton**: cálculo de la raíz de una función.



Sea x_0 un punto inicial. Si $f(x_0) \neq 0$ entonces se obtiene el siguiente punto de la sucesión intersectando la recta tangente a la curva $y = f(x)$ en el punto $(x_0, f(x_0))$ con el eje de abscisas.

$$y = 0$$
$$y = f(x_0) + f'(x_0)(x - x_0)$$

Despejando x, se obtiene $$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

En general $$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Si aplicamos el método de Newton a la función

$$y = f(x) = x^2 - a$$

cuya solución es $\sqrt{a}$, obtendremos la siguiente sucesión:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{x_n^2 - a}{2x_n} = \frac{x_n + \frac{a}{x_n}}{2}$$

que es equivalente a expresión la indicada anteriormente para calcular la raíz cuadrada de un número.

- ***Codificación del método de Newton:***

```
;; Funciones auxiliares
   (define (cociente-incremental f h)
         (lambda (x) (/
                        (- (f (+ x h)) (f x))
                        h
                     )
         )
   )

(define (bueno? x f) (< (abs (f x)) 0.001))
```

La función **siguiente** *permite obtener un nuevo elemento de la sucesión.*

```
   (define (siguiente x f)
         (-    x
               (/    (f x)
                     ((cociente-incremental f 0.001) x)
               )
         )
   )

;; Función que implementa el método de Newton.
   (define (newton f inicial)
         (if (bueno? inicial f)  inicial
                     (newton f (siguiente inicial f))
         )
   )

(newton  (lambda (x) (- (* x x) 2)) 1)       ==> 1.41421657
```