

RECURSIVIDAD

Hasta ahora hemos visto formas de estructurar tanto datos como programas, mediante funciones. Las funciones resuelven cometidos concretos y llevan parámetros que las hacen ser útiles para distintas situaciones. Las funciones vistas hasta ahora se resuelven por sí mismas o, a lo sumo se apoyan en otras funciones auxiliares que resuelven el problema.

Sin embargo, hay veces en que una función solo se puede resolver volviéndose a llamar a sí misma. Se trata de **funciones recursivas**. ¿Cuándo tiene esto sentido? En este tema se introduce el concepto de recursividad, se examinan los posibles casos, se muestra como un computador, que es fundamentalmente iterativo, no recursivo, puede emular la recursividad, se presentan algunas reglas para comprobar que las soluciones recursivas que se construyan tengan final y, se presentan algunos ejemplos importantes de recursión.

Definición y Tipos de recursividad

Se dice que un proceso es recursivo si se resuelve llamándose a sí mismo. Para que la recursión tenga sentido (un final útil) cada vez que se llame internamente a sí misma deberá hacerlo de una manera “menos recursiva” hasta que finalmente se llame a sí misma de una manera no recursiva.

La recursión infinita es inútil.

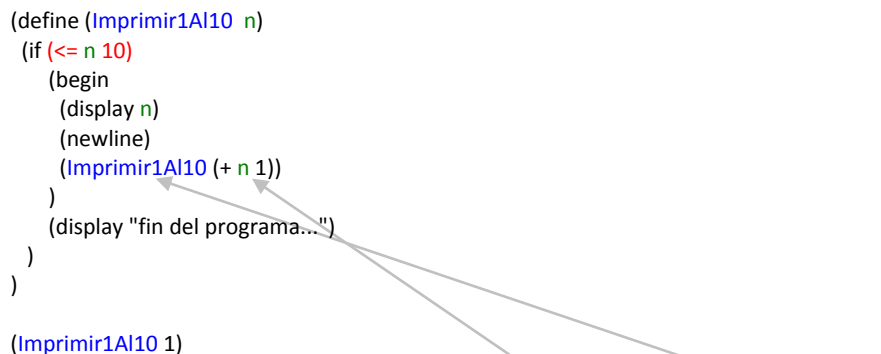
Lo único que determina el comportamiento de una función son los parámetros que son los que dan idea del “tamaño del problema”. La función no es recursiva para algunos valores del parámetro que se denominan casos base. Toda función recursiva, pues, debe tener algún caso base y toda llamada recursiva dentro de ella debe tender hacia el caso base.

Ejemplo1:

Imprimir los números del 1 al 10.

```
(define (Imprimir1Al10 n)
  (if (<= n 10)
      (begin
        (display n)
        (newline)
        (Imprimir1Al10 (+ n 1))
      )
      (display "fin del programa..."))
  )
)

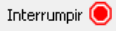
(Imprimir1Al10 1)
```



La característica principal de la recursividad es que una función SE LLAMA A SI MISMA para incrementar o restar el valor de un parámetro (en este caso se incrementa el valor de **n** para que llegue a ser igual a 10. Cuando **n** sea igual a 11, la condición **(<= n 10)** se evaluará como **falsa** y se terminará de ejecutar la función y así lograr que el procedimiento tenga un final, es decir que no se quede en un ciclo infinito.

Debemos siempre poner una condición para que se ejecute el proceso. Para saber que condición debemos poner hay que hacernos la pregunta: ¿Hasta cuando queremos que se repita el procedimiento?

Si miramos nuestro ejemplo, vemos que el primer valor que entra a la función es 1 y lo recibe el parámetro **n**. Como queremos mostrar los números del 1 al 10, la condición nos debe permitir que se evalúe como **verdadera** hasta que **n** tenga el valor de 10. Por eso se pone la condición **(<= n 10)**. Examinemos esta expresión: Cuando **n=1** (primer caso), la condición es **verdadera** porque 1 es menor o igual que 10 y se muestra el 1 en la pantalla. Luego se llama de nuevo a la función y se incrementa el valor de **n**, es decir que **n** ahora vale 2. Cuando **n=2** la condición también es **verdadera** y se muestra el 2 en la pantalla, y así se repite el proceso hasta que **n=11** y la condición sea **falsa** y se sale de la función, sin mostrar más números en pantalla.

Hacer esta prueba: cambiar el llamado (+ n 1) por (- n 1), y vuelva a Ejecutar. ¡OJO! Usted puede parar un programa que se quedó en un ciclo infinito usando 

¿Por qué el programa se queda en un ciclo infinito? Para comprender mejor lo que sucede puede hacer DEBUG. Esta opción le permite ejecutar el programa paso a paso o expresión por expresión.



En archivo adjunto, encontrará otros ejemplos básicos de recursividad, pruébelos, se sugiere usar DEBUG para analizar y comprender mejor el concepto de RECURSIVIDAD.

Ejercicios:

1. Hacer una función que permita imprimir los números del 10 al 1.
2. Hacer una función que permita imprimir los cuadrados de los números del 1 al 10.

Ej: 1
4
9
.
.
.
100

3. Hacer una función que imprima los números que hay del 10 al 100. Frente a cada número debe aparecer su cuadrado, su cubo, su raíz cuadrada y su raíz cúbica. *Ejemplo:*

| Número | Cuadrado | Cubo | Raíz cuadrada | Raíz cúbica |
|--------|----------|---------|---------------|-------------|
| 10 | 100 | 1000 | 3.166.... | 2.154.... |
| | | | | |
| 100 | 10000 | 1000000 | 10 | 4.6415... |

No tienen que hacer la tabla. Tan solo mostrar los números como está en el ejemplo.

4. Hacer una función que permita imprimir la suma total de los números del 1 al 10. *Deben mostrar la suma total de: 1+2+3+...+10 (Usar un parámetro que lleve la suma). En este ejemplo debe mostrar: 55.* Los ejercicios siguientes de sumas deben mostrar también la suma total.
5. Hacer una función que permita imprimir la suma de los cuadrados de los números del 1 al 10. *Deben mostrar la suma total de: 1+4+9+...+100 (Usar un parámetro que lleve la suma)*
6. Construir una función que reciba como parámetro un número N, y calcule la suma de todos los enteros menores que el número recibido. *Ej: Si la función recibe 4, la función debe devolver la suma=1+2+3. En este caso deben mostrar=6.*
7. Construir un programa que dados dos enteros M y N diferentes, calcule la suma de los cuadrados de los números que hay entre ellos, sin incluirlos. *Ej: Si la función recibe un 1 y un 6, la función debe devolver la suma total de: 4+9+16+25*
8. Escriba un programa que calcule la suma de los números que existen entre dos números dados. Debe incluir ambos números. *Ej: si se entran los números 3 y 8 la función debe devolver la suma total de: 3+4+5+6+7+8*

9. Escriba una función que calcule cuantos números naturales hay entre 2 números dados, incluyéndolos. *Por ejemplo si se entran los números 3 y 8, la función debe devolver: 6, pues hay 6 números del 3 al 8 así: 3,4,5,6,7,8.*
10. Escriba un programa que calcule el promedio de los números naturales que existen entre dos números dados. Debe considerar ambos números. *(Puede usar las dos funciones anteriores)*
11. Hacer un programa que lea 10 números e imprima el cuadrado y el cubo de cada número. Los números se deben leer dentro de la función. *(Usar la función (read))*
12. Hacer un programa que reciba un número.
Si el número es menor o igual que 5: imprime los números del 1 al 5.
Si el número es mayor que 5 y menor o igual que 10: imprime los números del 5 al 10.
Si el número es mayor que 10: imprime los números del 10 al 20.