La recursividad es la posibilidad de que una función se llame a sí misma, bien directa o indirectamente. Un ejemplo típico es el cálculo del factorial de un número, definido en la forma:

codeDivStart()N! = N * (N-1)! = N * (N-1) (N-2)! = N * (N-1)*(N-2)*...*2*1

La función factorial, escrita de forma recursiva, sería como sigue:

```
codeDivStart()unsigned long factorial(unsigned long numero)
{
   if ( numero == 1 || numero == 0 )
      return 1;

   /* El else no hace falta, ya que es obvio */
   return numero * factorial(numero-1);
}
```

Supóngase la llamada a esta función para N=4, es decir factorial(4). Cuando se llame por primera vez a la función, la variable numero valdrá 4, y por tanto devolverá el valor de
4*factorial(3);
pero
factorial(3)
devolverá
3*factorial(2);
factorial(2)
a su vez es
2*factorial(1)
y dado que factorial(1) es igual a 1 (es importante considerar que sin éste u otro caso particular, llamado caso base, la función recursiva no terminaría nunca de llamarse a sí misma), el resultado final será 4*(3*(2*1)).

Casi siempre es posible escribir una función recursiva en forma iterativa, es decir usando sólo ciclos for o while, por ejemplo la función factorial en forma iterativa quedaría como:

```
codeDivStart()unsigned long factorial(unsigned long numero)
{
   unsigned long f=1;
   while ( numero >1 )
   {
         f = f * numero;
         numero--;
   }
```

```
    return f;
}
```

Otro ejemplo clásico de recursión es la serie de fibonaci, cuya definición es la siguiente:

```
fibonaci(1) = 1;
fibonaci(2) = 1;
fibonaci(3) = fibonaci(2) + fibonaci(1);
fibonaci(4) = fibonaci(3) + fibonaci(2);
```

Es decir la número de fibonaci de 1 y 2 es 1 en ambos casos, el número fibonaci de 3 es la suma del fibonaci de 2 y 1, el número fibonaci de 4 es la suma del fibonaci de 3 y 2, y así en forma sucesiva, la definición sería:

```
fibonaci(n) = fibonaci(n-1) + fibonaci(n-2)
```

Implementado esto en forma recursiva quedaría como:

```
codeDivStart()unsigned long fibonaci(unsigned long numero)
{
   if ( numero == 1 || numero == 2 )
      return 1;

   /* El else no hace falta, ya que es obvio */
   return fibonaci(numero-1) + fibonaci(numero-2);
}
```

Y por supuesto la versión iterativa es la siguiente:

```
codeDivStart()unsigned long fibonaci(unsigned long numero)
{
   unsigned long f1=1, f2=1, T;
   unsigned long i;

    int i;
    for (i=2; i       {
         T=f1;
         f1=f1+f2;
         f2=T;
    }

    return f1;
}
```

Sobre el factorial, mencionar que se puede escribir en una sóla línea:

```
codeDivStart()unsigned long factorial(unsigned long numero)
{
    return numero==0 ? 1 : numero * fact(numero -  1);
}
```

Y eso es todo...

Mencionar que, por lo general la recursividad no ahorra memoria, pues ha de mantenerse una pila con los valores que están siendo procesados. Tampoco es más rápida, sino más bien todo lo contrario, pero el código recursivo es más compacto y a menudo más sencillo de escribir y comprender.