Recursión, parte II

Ejercicio 4. Escriba una función que calcule el valor de b^n.

👀 Observaciones:
- si n es par, $b^n = b\text{^}(n/2)\cdot b\text{^}(n/2)$
- si n es impar, $b^n = b\text{^}((n\text{-}1)/2)\cdot b\text{^}((n\text{-}1)/2)\cdot b$
- si $n = 0$, el resultado es 1

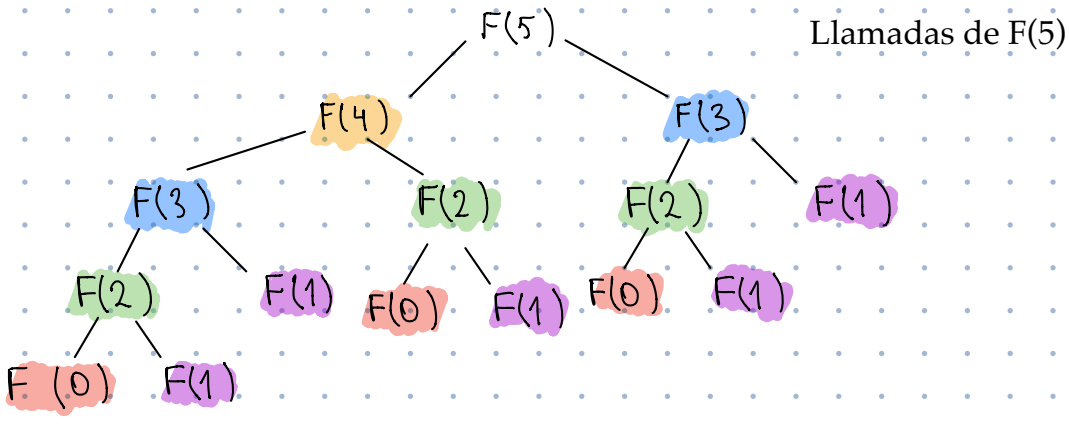
```
def potencia(b, n):
    #caso base
    if n==0:
        return 1
    elif: n%2==0:
        return potencia(b,int(n/2))*potencia(b,int(n/2))
    else:
        return potencia(b,int((n-1)/2))*potencia(b,int((n-1)/2))*b
```

Ejercicio 5. Alguien compra una pareja de conejos. Luego de un mes esos conejos son adultos. Después de dos meses esa pareja de conejos da a luz a otra pareja de conejos. Al tercer mes la primera pareja da a luz a otra pareja de conejos y sus primeros hijos se vuelven adultos. Cada mes qué pasa, cada pareja de conejos adultos da a luz a una nueva pareja de conejos y una pareja de conejos tarda un mes en crecer. Escriba una función que regrese cuántos conejos adultos se tienen pasados n meses.

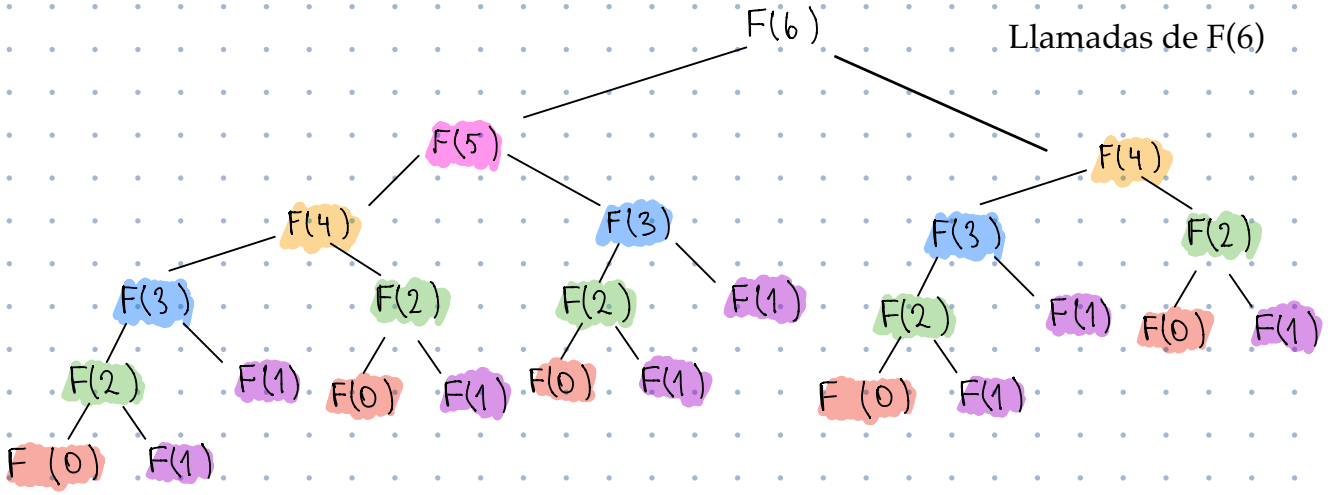
👀 Observaciones:
- si $n = 0$, entonces hay 0 parejas de conejos adultos
- si $n = 1$, entonces hay 1 pareja de conejos adultos
- el número de conejos adultos luego de n meses, es el número de conejos adultos luego de n - 1 meses más el número de conejos jóvenes en el mes n - 2.

```
def F(n):
    #caso base
    if n==0:
        return 0
    #caso recursivo
    elif n==1:
        return 1
    else:
        return F(n-1)+F(n-2)
```

⚠️ La función recursivas de Fibonacci es extremadamente lenta. ¿Qué la hace lenta? Por ejemplo, para calcular F(5) se hacen las siguientes llamadas:



Llamadas de F(5)



Definición. Una secuencia / sucesión numérica es una *función* que va de los números naturales hacia los números reales. En símbolos:

$$a\colon N \to R,\ a = f(n)$$

Definición. Una relación de recurrencia es una relación matemática entre el término n-ésimo de una secuencia numérica y uno o varios términos anteriores al término n-ésimo.

Ejemplo 1. Buscamos dar respuesta a la pregunta *¿cuántas llamadas recursivas hace la función de Fibonacci?*

Vamos a plantear este problema como una relación de recurrencia.

Para F(0), realizamos 1 llamada. Para F(1), realizamos 1 llamada. Para F(2), realizamos 2 llamadas. Para F(3), realizamos 3 llamadas.

En general podemos definir una secuencia numérica que describe la cantidad de llamadas recursivas al calcular F(n), es decir, una secuencia $a = \{a_0, a_1, a_2, ...\}$ cuyos elementos son las cantidades de llamadas recursivas al calcular F(n).

$$a = \{1, 1, 2, 3, ...\}$$
F(0), F(1), F(2), F(3), ...

⚠️ Para F(4), requerimos F(3) y F(2), es decir, 3 y 2 llamadas, respectivamente. Entonces para F(4), realizamos 2 + 3 = 5 llamadas.

En general, vemos que el número de llamadas recursivas para calcular F(n) es igual a la suma del número de llamadas para calcular F(n-1) y el número de llamadas para calcular F(n-2).

Entonces, la secuencia numérica $a = \{1, 1, 2, 3, ...\}$ puede definirse mediante la relación de recurrencia:

$$a(n) = a(n\text{-}1) + a(n\text{-}2)$$

🔍 Dejaremos planteado el problema y aprenderemos a resolver una ecuación en diferencias finitas, — EDF— en las próximas sesiones de clase para obtener una «*fórmula*» que permita averiguar el número exacto de llamadas a la función F(n) para un n específico sin tener que calcular los números de llamadas de F(n-1) y F(n-2).

Finalmente, ¿cómo podemos mejorar la ejecución de la función de Fibonacci?

Utilizamos recursión con memoria, un método para evitar que una misma función recursive varias veces ejecutándose bajo las mismas condiciones, al tener una estructura para guardar resultados ya calculados.

En esencia, cada vez que se desee calcular F(n) se verificará si dicho valor ya fue calculado. El siguiente código de Python es una implementación de esta solución.

```
m=numpy.zeros(50, dtype='int')
def memory_F(n):
    global m
    #memoria del programa
    if m[n]!=0:
        return m[n]
    #caso base
    if n==0:
        return 0
    if n==1:
        return 1
    m[n]=memory_F(n-1+memory_F(n-2)
    return m[n]
```