Auxiliar 2

Profesor: Patricio Inostroza
Auxiliares: Analía Bannura, Martín Cornejo

P1.- Exponenciación eficiente

Dada la siguiente definición recursiva de la función :

$$f(a,b) = a^b = \begin{cases} 1 & \text{si } b = 0 \\ a \cdot a^{b-1} & \text{si b} > 0 \end{cases} \quad (1)$$

se obtiene el siguiente código:

```
# potencia : num int -> num
# calcula la potencia de base elevado a exp
# ejemplo : potencia (2 , 4) devuelve 16
def potencia ( base , exp ):
  if exp == 0:
    # caso base
    return 1
  else :
    # caso recursivo
    return base * potencia ( base , exp - 1)

# Test
assert potencia (2 , 4) == 16
assert potencia ( -1 , 5) == -1
assert potencia (3 , 0) == 1
```

Su amig@ le dice que la implementación de esta función está correcta, mas no es eficiente. Con algunas modificaciones al programa se puede crear una nueva función recursiva que obtiene los mismos resultados pero en una cantidad menor de pasos recursivos. Progame dicha solución.

Hint: Vea como reducir el exponente de forma más ”rápida”.

P2.- Triángulo de Pascal

Como se observa en la figura 1, las diagonales externas de este triángulo son siempre 1, en cambio los demás números se obtienen como la suma de sus ”padres”. Programe una función recursiva que calcule cualquier número en este triángulo, cuyos argumentos sean la fila y la columna en la que se

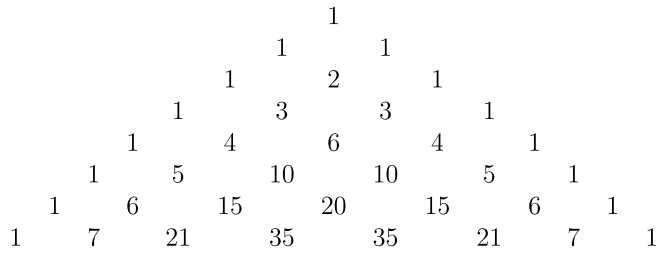

Figura 1: Triángulo de Pascal

encuentra dicho número, en ese orden. Ej: $f(4, 2) = 3$

P3.- **Calcular el seno de un ángulo numéricamente** [1]

Sabemos que para un ángulo x suficientemente pequeño, se cumple la aproximación $\sin(x) \approx x$. Además, se tiene la siguiente identidad trigonométrica:

$$\sin(x) = 3\sin\left(\frac{x}{3}\right) - 4\sin^3\left(\frac{x}{3}\right)$$

Usando recursivamente esta identidad se puede reducir el ángulo lo suficiente para que sea considerado pequeño (≤ 0.1 radianes) y poder aproximar la función $\sin(x)$.

a) Escriba la definición matemática recursiva del método.

b) Programe la función $\sin(x)$ y no olvide la receta de diseño con los tests. Utilice funciones auxiliares donde sea necesario.

Hint: Recuerde que cuando los resultados no son exactos debe usar la siguiente función para validar:

```
# cerca: num num num -> bool
# retorna True si x es igual a y con
# precision epsilon
def cerca(x, y, epsilon):
  diff = x - y
  return abs(diff) < epsilon

assert cerca(0.0, 0.0, 0.0001) == True
assert cerca(3.1416, math.pi, 0.0001) == True
```

[1] Modificación del ejercicio 1.15 del libro *Structure and Interpretation of Computer Programs*