Exercici 5: Algorismes numèrics.

Lliurament:

UN ÚNIC FITXER (exercici5.py) QUE CONTINGUI EL CONJUNT DE FUNCIONS QUE S'HAN IMPLEMENTAT.

Criteris d'avaluació (per ordre d'importància):
1.- El programa ha de donar un resultat correcte: 50% de la nota.
2.- Ús adequat del llenguatge (fer servir if/while correctament, fer servir la col·lecció adequada, etc.): 30% de la nota.
3.- Bon estil de programació (fer una interfase d'usuari adequada, comentaris, etc.): 20% de la nota.

AQUESTA LLISTA DE PRÀCTIQUES S'HA DE REALITZAR EN DUES SESSIONS (4 HORES)!

- El Màxim Comú Divisor (MCD) de dos valors es pot calcular amb l'algorisme d'Euclides. Començant amb els valors `m` i `n`, apliquem repetidament la fórmula `n, m=m, n%m` fins que `m` és 0. Llavors, `n` és el MCD de `m` i `n`. Escriviu una funció, **mcd**, que calcula el MCD de dos nombres enters entrats per l'usuari.

- El sedàs d'Eratòstenes és un algorisme antic per cercar tots els nombres primers fins a un determinat enter. Va ser creat per Eratòstenes (276-194 aC) un matemàtic de l'Antiga Grècia.
  Referència: http://ca.wikipedia.org/wiki/Sedàs_d'Eratòstenes

```
Algorisme:

1. Escriviu una llista A amb els números des del 2 fins a l'enter més gran N que
vulgueu calcular.
2. El primer nombre de la llista és un nombre primer. Anoteu-lo en una llista de
nombres primers, B.
3. Esborreu de la llista A el primer nombre i els seus múltiples.
4. Si el primer nombre de la llista A és més petit que l'arrel quadrada de N, torneu
al punt 2.
5. Els nombres de la llista B i els que queden a la llista A són tots els nombres
primers cercats.
```

  Escriu una funció, **eratostenes1**, que demani a l'usuari un nombre `n` i llavors usi aquest algorisme per imprimir tots els nombres primers menors o iguals que `n`.

- La forma empírica més corrent per a saber quan triga un determinat algorisme és usar les funcions de Python que ens permeten consultar el rellotge de l'ordinador. Per exemple, si volguéssim saber el temps empleat per calcular la suma dels factorials de tots els enters fins a 200 ho podríem fer amb la funció següent:

```
def temps():
    import time
    import math
    t1 = time.clock()
    a = 0
    for i in range(1,200):
         a = a + math.factorial(i)
    t2 = time.clock()
    print "\n"
    print "El temps de proces ha estat %0.3f ms" % ((t2-t1)*1000)
```

  Escriu una funció, **eratostenes2**, que imprimeixi el temps que es triga a calcular mitjançant el sedàs d'Eratòstenes els nombres primers menors que 10.000.000 i quants nombres primers hi ha.

- L'any 1640, Fermat va definir el següent teorema:

  Si P és un nombre primer, llavors, per a qualsevol nombre a,
  de forma que 1 <= a < P, es compleix que mod(a^{P-1},P) = 1.

  Escriu una funció, **fermat1**, que comprovi si es compleix el teorema petit de Fermat pels nombres primers menors que 1.000. La funció ha d'escriure al final de tots els tests el resultat (cert o fals) i el temps que ha trigat.

- Escriu una funció, **fermat2**, que comprovi si es compleix el teorema de Fermat pels nombres compostos, i només pels nombres compostos, entre 2 i 1000. La funció ha d'escriure al final de tots els tests el resultat (cert o fals), quants nombres compostos hi ha i el temps que ha trigat.

- Entre tots els possibles tests de primalitat, un dels que més freqüentment s'usen és el *test de Miller-Rabin*. L'algorisme és el següent (http://en.wikipedia.org/wiki/Miller–Rabin_primality_test):

Entrada: Un nombre natural n>2, el nombre k de vegades que s'executa el test (factor que ens determina la fiabilitat del test!).

Sortida: COMPOST si n és compost i PROBABLE PRIMER si n és un probable primer.

Definir r y s tal que r és senar i $(n-1) = r \cdot 2^s$ (per exemple, si fem el test per 221, calculem $200=2^2 \cdot 55$)
Per j des de 1 fins k fer:
  $a \leftarrow$ Genera un nombre aleatori a l'interval $[2, n-2]$
  Si $(y \neq 1) \wedge (y \neq n-1)$ llavors:
    $j \leftarrow 1$
    Mentre $j \leq (s-1) \wedge y \neq (n-1)$ fer:
      $y \leftarrow y^2 \bmod n$
      Si $y = 1$ llavors:
        Retornar COMPOST
      $j \leftarrow j + 1$
    Si $y \neq (n-1)$ llavors:
      Retornar COMPOST
Retornar PROBABLE PRIMER

Implementa una funció, **miller_rabin1**, que comprovi (amb k=100) si un nombre entrat per l'usuari és probable primer o compost i el temps que ha trigat. La primera instrucció de l'algorisme es pot implementar així:

```
def sr(n):
    factors = []
    s = 0
    while n > 0:
        if n % 2 == 0:
            factors.append(0)
            n = n / 2
            s += 1
        else:
            factors.append(n)
            r = n
            break
    return s, r
```

Implementa una funció, **miller_rabin2**, que busqui amb aquest test els possibles nombres primers entre 2 i 10.000.000 fent servir k=10. La funció ha d'escriure al final el nombre de possibles primers que ha trobat i el temps que ha trigat.