Exercício de programação proposto para resolver até 3 de Abril.

Escreva na linguagem de programação de sua preferência
(aconselha-se o Python ou o Haskell) as seguintes funções

a) mpot(a,b,m) calcula a^b mod m de forma eficiente (polinomial
   em |b| para |m| fixo)

b) ctest(n) calcula o número de a's tal que a^b mod m não é 1
   (número de testemunhas baseado no pequeno Teorema de Fermat)
   2 <= a < n

c) ftest(n) frequência das testemunhas = ctest(n) / (n-2)

d) Lista dos inteiros n <= 3000 tais que ftest(n)<0.5.

e) Teste de Rabin-Miller, rm(n) -> {False,True}

f) Lista dos inteiros n <= 1000 baseada numa repetição de 10 vezes
   (no máximo) do teste de Rabin-Miller.

```
#-------------------------------------------------------------
Possível solução
#-------------------------------------------------------------
#  a^b  (mod m)
def mpot(a,b,m):
    if b==0:
        return 1
    t=mpot(a,b/2,m)
    if b%2==0:
        return (t*t)%m
    return (t*t*a)%m
#-------------------------------------------------------------
#  n is primo?  Testa divisores 2,3,...,int(sqrt(n))
def primo(n):
    d=2
    while d*d<=n:
        if n%d==0:
            return False
        d=d+1
    return True
#-------------------------------------------------------------
# teste... primos ate' 1000
print "primos",
for i in range(1,1001):
    if primo(i):
        print i,
```

```
#-------------------------------------------------------------
# numero de testemunhos de composicionalidade (pequeno T. de Fermat)
def ctest(n):
    return ([a for a in range(2,n) if mpot(a,n-1,n)!=1])
#-------------------------------------------------------------
# percentagem de testemunhos de composicionalidade (pequeno T. de Fermat)
def ftest(n):
    c=0.0
    a=2
    for a in range(2,n):
        if mpot(a,n-1,n)!=1:
            c=c+1
    return c/(n-2)

#... teste ...
print
print "compostos e frequencia PTF"
for i in range(1,2000):
    if not primo(i) and ftest(i)<0.5:
        print i, ftest(i)

lista = [561,1105,1729,2465]
#-------------------------------------------------------------
#  teste de Rabin-Miller com "testemunha" a
from random import *
def rm(n,a):
    s=0
    d=n-1
    while d%2==0:
        s=s+1
        d=d/2
    if mpot(a,d,n)==1:
        return False     # primo
    for i in range(s):
        if mpot(a,d,n)==n-1:
            return False # primo
        d=2*d
    return True          # composto
#-------------------------------------------------------------
# percentagem de testemunhos de composicionalidade RM
def crm(n):
    c=0.0
    for a in range(2,n):
        if rm(n,a):
            c=c+1
```

```
    print c,
    return c/(n-2)

#... teste ...
print
print "compostos e frequencia RM"
for i in range(2,200):
    if not primo(i):
        print i,crm(i)

#-------------------------------------------------------------
# teste de primalidade RM; r tentativas...
def primorm(n,r):
    for i in range(r):
        a=randint(2,n-1)
        if rm(n,a):
            return False
    return True

#... teste ...
print
print "primos ate' 1000: RM (10x), deteccao de erros"
for n in range(3,1001):
    if primorm(n,5) and not primo(n):
        print n,
```

```
Resultados
------------------------------------------------------------------

primos 1 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79
83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173
179 181 191 193 197 199 211 223 227 229 233 239 241 251 257 263 269
271 277 281 283 293 307 311 313 317 331 337 347 349 353 359 367 373
379 383 389 397 401 409 419 421 431 433 439 443 449 457 461 463 467
479 487 491 499 503 509 521 523 541 547 557 563 569 571 577 587 593
599 601 607 613 617 619 631 641 643 647 653 659 661 673 677 683 691
701 709 719 727 733 739 743 751 757 761 769 773 787 797 809 811 821
823 827 829 839 853 857 859 863 877 881 883 887 907 911 919 929 937
941 947 953 967 971 977 983 991 997

compostos e frequencia PTF
561  0.4293
1105 0.3046
1729 0.2501

compostos e frequencia RM
4   2.0   1.0
6   4.0   1.0
8   6.0   1.0
9   6.0   0.857
10  8.0   1.0
12  10.0  1.0
14  12.0  1.0
15  12.0  0.923
16  14.0  1.0
.......
189 186.0 0.994
190 180.0 0.957
192 190.0 1.0
194 192.0 1.0
195 192.0 0.994
196 192.0 0.989
198 196.0 1.0

primos ate' 1000: RM (10x), deteccao de erros
(nada!)
```