```python
from math import sqrt

def est_premier(n):
    if n < 2:
        return False
    max = int(sqrt(n))
    for k in range(2, 1 + max):
        if n % k == 0:
            return False
    return True


#  Q1
from math import log, sqrt, floor, ceil
print(log(0.5))

# Q2
def sont_proches(x, y):
    atol, rtol = 1e-5, 1e-8
    return abs(x - y) <= atol + rtol * abs(y)

# Q3  3

# Q4  si 0 < x < b : 0, sinon
# mystere(x, b) = floor(log(x, b))

# Q5 essayer aussi avec 5*10**4
# à la place de 10**5

# Q6  environ N = 10**9

# Q7   32 10**9

# Q8

def erato_iter(N):
    liste_bool = [True for k in range(N+1)]
    liste_bool[0], liste_bool[1] = False, False
    for i in range(2, int(sqrt(N)) + 1):
        if liste_bool[i]:
            for k in range(2, 1 + N//i):
                liste_bool[i*k] = False
    return liste_bool

# Q 9  N.ln(ln N)
# Q 10  2**n.(ln n)

# Q 11  2**N - 1

# Q 12
```

```
import time

def graine():
    t = time.time()
    t = t - int(t)
    t = int(t * 10**7)
    return t

def bbs(N):
    p1 = 24375763
    p2 = 28972763
    M = p1 * p2
    xi = graine()
    A = 0
    for i in range(N):
        if xi % 2 == 1:
            A = A + 2**i
        xi = (xi**2) % M
    return A

def test(p):
    return p > 1 and \
           (2**(p-1)) % p == 1 and \
           (3**(p-1)) % p == 1 and \
           (5**(p-1)) % p == 1 and \
           (7**(p-1)) % p == 1

# Q 13
def premier_rapide(n_max):
    N = int(log(n_max, 2))
    p = bbs(N)
    while not test(p):
        p = bbs(N)
    return(p)

def essai():
    for k in range(10):
        print(premier_rapide(65))
        time.sleep(0.01)

# Q 14
def stats_bbs_fermat(n_max, nb):
    liste_bool = erato_iter(n_max)
    faux = []
    for k in range(nb):
        p = premier_rapide(n_max)
        if not erato_iter(p):
            faux.append(p)
    return len(faux) / nb, faux
```

```
def premier_des_faux():
    k = 2
    while not test(k) or est_premier(k):
        k += 1
    return k

# Q 15
def Pi(N):
    liste_bool = erato_iter(N)
    resultat = [[1, 0]]
    compteur = 0
    for k in range(2, N+1):
        if liste_bool[k]:
            compteur += 1
        resultat.append([k, compteur])
    return resultat

# Q 16   [[0,0]] pour corriger
# la malveillance de l'énoncé
def verif_Pi(N):
    Pi_N = [[0, 0]] + Pi(N)
    resultat = True
    for n in range(5393, N+1):
        if n >= (log(n) - 1) * Pi_N[n][1]:
            return False
    return True

# Q 17, 18   complexité en O(n) où n est le nombre de points :
n = (b-a)/pas

# Q 19
def inv_ln_rect_d(a, b, pas):
    n = round((b-a) / pas)
    s = sum(1 / log(a + k*pas) for k in range(1, n+1))
    return s*pas

# Q 20
def li_d(x, pas):
    if x < 1:
        return inv_ln_rect_d(0, x, pas)
    else:
        return inv_l_rect(0, 1 - pas) + inv_ln_rect_d(1 + pas,
x, pas)

# Q 21  Au voisinage du point où la fonction s'annule,
#  pour une même erreur absolue,
#  l'erreur relative tend vers l'infini

# Q 25 Plusieurs lignes avec le même nom
```

```
# Q 26

# 1  SELECT COUNT(*), AVG(ram) FROM ordinateurs

# 2  SELECT nom FROM ordinateurs
#    WHERE nom NOT IN
#    SELECT teste_sur FROM fonctions
#    WHERE algorithme = restangles AND nom = li

# 3 SELECT algorithme, teste_sur, ram
#   FROM ordinateurs JOIN fonctions
#   ON ordinateurs.nom = fonctions.teste_sur
#   WHERE fonctions.nom = Ei
#   ORDER BY temps_exec DESC
```