Idoufker
Aboubakr
Licence 2
Semestre 3

Informatique Appliquées
aux Mathématiques
Groupe 1

TRAVAIL PRATIQUE 4

Mr Cardinal

Institut Galilée
Université Paris 13
2017-2018

Introduction:

L'Arithmétique est une branche des Mathématiques très vaste. Dans ce TD, nous avons fait programmé beaucoup de choses dont le pgcd de deux entiers, trouver les coefficients a,b de l'identité de Bézout, remonter l'algorithme d'Euclide et il y même une partie sur le système RSA. Les réponses exigées et les fonctions sont dans deux fichiers différents nommés 'tp4.py' et 'fonctiontp4.py'.

Exercice 1:

Cet exercice tourne autour du sujet des nombres premiers. Nous n'avons pas trop rencontré de difficultés, si ce n'est dans la fonction `primes`.

a) Soient `a` et `b` deux entiers, on trouve le quotient de la division euclidienne de `a` par `b` à l'aide du signe `/` (`a/b`) et le reste de cette division `r` et donné par la relation : `r = a%b`.

b) Un **nombre premier** est un entier naturel qui admet exactement deux diviseurs distincts entiers et positifs (qui sont alors 1 et lui-même).

c) 1001 et 89999 ne sont pas premiers puisque 1001 = 7*143, et 89999 = 7*12857. En revanche 2017, 3001, 49999 sont premiers.

d) La fonction `is_prime` renvoie des booléens. C'est une nouvelle notion que nous avons appris dans ce TP. On insère avant tout les modules nécessaires (numpy, etc.) :

```
import random
import numpy as np
import math
```

La description de `is_prime` est la suivante :

```
def is_prime(n):
    if (n < 2):
        return False


    for p in range(2, (int(np.sqrt(n))+1)):
        if n%p == 0:
            return False

    return True
```

Si n = 0 ou n= 1 alors n n'est pas premier.

Si n > 1, on prend tous les entiers entre 2 et √n. (+1 car sinon il sort de la boucle)

si on en trouve un parmi eux multiple de n, on retourne faux

S'il n'en trouve aucun il sort de la boucle et il retourne vrai

e) On invente une fonction pour les nombres de Fermat, puis on utilise, la fonction `is_prime` sur le terminal afin de voir s'ils sont premiers ou non.

La fonction des nombres de Fermat :

```
def Fn(n):
    return 2**(2**n) + 1
```

L'éxecution de `Fn(1)`, `Fn(2)`, `Fn(3)`, `Fn(4)`, `Fn(5)`, avec `is_prime` :

```
le nombre de Fermat 5 est premier
le nombre de Fermat 17 est premier
le nombre de Fermat 257 est premier
le nombre de Fermat 65537 est premier
le nombre de Fermat 4294967297 n est pas premier
```

Exercice 2:

a) Voici le crible d'Ératosthène :

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120
121	122	123	124	125	126	127	128	129	130
131	132	133	134	135	136	137	138	139	140
141	142	143	144	145	146	147	148	149	150
151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170
171	172	173	174	175	176	177	178	179	180
181	182	183	184	185	186	187	188	189	190
191	192	193	194	195	196	197	198	199	200

Les nombres premiers de 1 à 200 sont :
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.

b) La fonction `primes` est définie de la manière suivante :

```
def primes(n):
    P = [k for k in range(2, n) if is_prime(k)]
    return P
```

On prend tous les nombres de 2 à n, on vérifie s'ils sont premiers (avec is_prime) si oui alors on les ajoute dans P.

c) On teste primes avec n = 1000, puis on enregistre le tableau dans 'primes.txt' comme ceci :

```
P = primes(1000)
with open('primes','w') as f:
    f.write('Tableau contenant tous les entiers de 1 a 1000\n')
    cpt = 0
    for p in P:
        cpt += 1
        f.write('{0}  '.format(p))
        if cpt%10 == 0:
            f.write('\n')
```

Entête

Compteur de nombres par ligne.

Pour tous les nombres de la liste P
On écrit les nombres dans 'primes.txt'

Quand le compteur est multiple de 10 on saute une ligne.

Finalement, le fichier ‘primes.txt’ sera envoyé au professeur, avec le compte rendu.

d) On définit deux fonctions π, et *n/ln(n)* comme demandé :

```
def pi(n):
    return np.shape(primes(n))[0]

def f(n):
    return n/(math.log(n))
```

La fonction shape du module numpy renvoie le cardinal d’une liste.

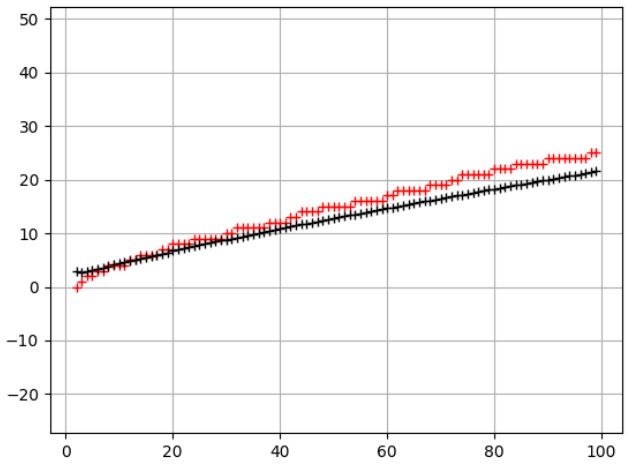
En l’occurence, ici, on compte le cardinal de la liste renvoyée par primes.
Or, shape renvoie une liste (à un élément), c’est pourquoi il y’a nécessité du [0]

Puis, on les représente graphiquement comme ceci :

```
plt.clf()
for i in range(2,100):
    plt.plot(i, pi(i) ,'+r',i,f(i),'+k')
plt.grid('on')
plt.axis('equal')
plt.savefig('Graphique')
```

On a pas utilisé la fonction linspace cette fois-ci puisque ce sont des suites qu’on cherche à afficher donc une boucle for suffit.

Ça donne :



On voit quand même que les deux suites sont proches.
(π, en rouge et $n/ln(n)$ en noir)

e) On représente le tebleau une seconde fois avec `with open` le tableau sera envoyé avec le compte-rendu.

Exercice 3:

a) Tout entier strictement positif peut être écrit comme un produit de nombres premiers d'une unique façon, à l'ordre près des facteurs.

b) 924 = 462*2 = 231*2² = 77*2²*3 = 2²*3*7*11

c) Voici la fonction `factors` :

```
def factors(n):
    P = primes(n)
    FF = []
    for p in P:
        while n%p == 0:
            n = n/p
            FF.append(p)
    return FF
```

Pour p entier dans P (liste des entiers premiers avec n), tant que p divise n, Inclure p dans FF.

Exercice 4:

a) Le pgcd de deux entiers a et b, est le plus grand diviseur qu'ils ont en commun.

b) Pour calculer le pgcd de 4864 et 3458. On utilise une fonction pgcd, en attente pour l'occasion :

```
def pgcd(a, b):
    r = a%b
    while r != 0:
        a = b
        b = r
        r = a%b
    return b
```

Dans le terminal :

```
In [2]: pgcd(4864,3458)
Out[2]: 38
```

c) **L'identité de Bézout** est un théorème de la théorie des nombres élémentaires : soit *a* et *b*, soient des entiers non-nuls et disons *d* leur plus grand commun diviseur. Alors il existe des entiers *x* et *y* tels que $xa + yb = d$. La réciproque n'est pas forcément vraie.

d) Dans une première étape, on écrit l'algorithme d'Euclide :
4864 = 1*3458 + 1406
3458 = 2*1406 + 646
1406 = 2*646 + 114
646 = 5 * 114 + 76
114 = 1*76 + **38**
76 = 38 * 2 + **0**

Dans une seconde étape, on le remonte comme ceci :
38 = 114 – 76
38 = 114 – (646 – 5*114)
38 = - 646 + 6 * 114
38 = -646 + 6*(1406 – 2 * 646)
38 = 6 * 1406 – 13 * 646
38 = 6 * 1406 – 13*(3458 – 2 * 1406)
38 = 32 * 1406 – 13 * 3458
38 = 32 * (4864 – 3458) – 13 * 3458
38 = **32** * 4864 – **45** * 3458

e) La fonction Euclide nous a causé pas mal de problèmes donc nous nous sommes aidés essentiellement de sites internet, cela donne :

```
def euclide(a,b):
    d = a
    dp = b
    x = 1
    y = 0
    xp = 0
    yp = 1
    while (dp != 0):
        q = d/dp
        ds = d
        xs = x
        ys = y
```

```
            d = dp
            x = xp
            y = yp
            dp = ds - q*dp
            xp = xs - q*xp
            yp = ys - q*yp
    return d,x,y
```

Exercice 5:

a) Comme le professeur a demandé, cet exercice ne sera pas rédigé ici. Néanmoins, toute les fonctions figurent dans le fichier python. (Nous avons étudié les questions a) à f)).

Conclusion:

Pour ce dernier travail pratique, on s'excuse par avance d'avoir été négligent dans certaines parties du tp. Comme il nous restait que deux semeines pour travailler sur ce tp, nous avons été un peu trop rapides dessus contrairement aux autres tp. Sinon, nous l'avons vraiment apprécié par rapport à tous les autres tp, car il est plus instructif, plus facile à comprendre, plus facile à programmer, et surtout plus facile à traiter. Nous remercions le professeur pour cela, et pour son aide.