C.R. T.P. 8 début

1 Autour de l'algorithme d'Euclide

1.1 L'algorithme d'Euclide usuel :

```
def Euclide(a,b):
    b=abs(b)# important pour le test b>0
    while b>0:
        a,b=b,a%b
    return a
```

1.2 L'algorithme d'Euclide étendu

a) *Au brouillon, on trouve la relation de récurrence en remplaçant dans les égalités... on y reviendra peut-être de manière plus conceptuelle matriciellement et géométriquement..*

On définit des suites qui vont toutes être finies (r_k), (q_k), (u_k) et (v_k) récurrentes d'ordre deux, (sauf q_k) comme suit :

- **Initialisation (double sauf pour q_k) :** $\begin{cases} r_0 = a,\ q_0 = 0,\ u_0 = 1,\ v_0 = 0, \\ r_1 = b, u_1 = 0, v_1 = 1. \end{cases}$
- **Formule de récurrence :** pour tout $k \geq 1$, *tant que* $r_k \neq 0$
  a) (q_k, r_{k+1}) est le couple quotient-reste de la division euclidienne de r_{k-1} par r_k,
  b) $u_{k+1} = u_{k-1} - u_k q_k$, et $v_{k+1} = v_{k-1} - v_k q_k$.

Propriété : $\boxed{\textit{Pour chaque } k,\ au_k + bv_k = r_k.}$

Démonstration : notons P_k : $au_k + bv_k = r_k$. Preuve par récurrence double :
• Par l'initialisation donnée : P_0 et P_1 sont vraies.
• H.R. Supposons que pour un $k \geq 1$, P_k et P_{k-1} sont vraies.
Alors $au_{k+1} + bv_{k+1} = a(u_{k-1} - u_k q_k) + b(v_{k-1} - v_k q_k) = (au_{k-1} + bv_{k-1}) - q_k(au_k + bv_k)$.
Donc par l'H.R., $au_{k+1} + bv_{k+1} = r_{k-1} - q_k r_k$ et $r_{k-1} - q_k r_k = r_{k+1}$ par déf. de r_{k+1}. Ceci prouve P_{k+1}. La réc. est établie. □

b) Il est important de comprendre que l'algorithme d'Euclide étendu contient l'algorithme d'Euclide usuel. C'est important car cela garantit déjà sa terminaison, et pour ce qui est du calcul du pgcd, sa correction. Ainsi la condition dans le `while` ne change pas, et l'évolution du couple `a,b` non plus. Le reste de sa correction consiste dans la démonstration par récurrence faite au a).

Dans le code ci-dessous, on utilise des variables informatiques `u,v,up,vp` où le `p` veut signifier « prime ». Avec l'affectation en tuple en Python, ces quatre variables suffisent.

```
def AEE(a,b):
    u,v=1,0 # initialisation u_(i-1),v_(i-1)
    up,vp=0,1 # initialisation de u_i,v_i
    while b>0:
        q=a//b # valeur q_i
        a,b=b,a%b # a,b codent deux restes succ.
        u,v,up,vp=up,vp,u-q*up,v-q*vp
    # noter la puissance de l'affectation en tuple
    return  a,u,v# pour i=N, on retourne u_N,v_N
```

2 Tests naifs de primalité

a)
```
def estpremier(n):
    n=abs(n) # pour se ramener aux nombres positifs
    if n<=1:
        return False
    else:
        N=int(n**(1/2))
        for i in range(2,N+1):
```

```
            if n%i==0:
                return False
    return True
```

b) Le crible d'Eratosthène consiste à enlever les multiples successifs des nombres qui ne sont pas rayés, en commençant à 2 qui n'est pas rayé. Informatiquement, pour rayer un nombre on le remplace par un 0. Comme précédemment, il suffit de s'arrêter à la partie entière de $\sqrt{n}$.
Ensuite on écrit une autre fonction qui enlève les 0 de la liste. Il est important de séparer ces deux fonctions, car enlever directement les nombres de la liste dans la première fonction fausserait les indices !

```
def erato0(n):
    """renvoie la liste des entiers de 0 à 1 où les nombres  non premiers
     sont remplacés par des 0"""
    L=list(range(n+1))
    L[1]=0
    N=int(n**(1/2))
    for i in range(2,N+1):
        if L[i]!=0:
            j=2
            while j*i<=n:
                L[j*i]=0
                j=j+1
    return L

def enlevezero(L):
    M=[]
    for valeur in L:
        if valeur!=0:
            M.append(valeur)
    return M

def Eratosthene(n):
    return enlevezero(erato0(n))
```

c) Il est plus commode d'utiliser notre fonction `erato0` puisqu'on sait qu'à la fin du tableau `erato0(n)` la dernière entrée vaudra 0 ssi n n'est pas premier. D'où le code :

```
def premierErato(n):
    L=erato0(n)
    return L[n]!=0
```

d) On se dit quand même qu'`erato0`, en générant une liste, doit être plus couteuse que `estpremier`. En voici une confirmation expérimentale :

```
 from time import clock
T1=0
T2=0
N=100 # nombre d'essais
debut=100000 # premier nombre à essayer.
for i in range(debut,debut+N):
    t0=clock()
    estpremier(i)
    t1=clock()
    T1=T1+(t1-t0)
    t0=clock()
    premierErato(i)
    t1=clock()
    T2=T2+(t1-t0)
print("Temps avec EstPremier",T1)
print("Temps avec premierErato",T2)
 Temps avec estpremier 0.0018909999999792149
Temps avec premierErato 5.802627000000019
```