TP1 – Euclide et consorts

Remarque générale.
Toutes les fonctions écrites doivent être testées. Par exemple, vous pouvez écrire, pour chaque fonction `fct`, une fonction de test `test_fct()` qui ne prend pas de paramètre et effectue des tests. Vous pouvez avoir ces fonctions de tests dans le même fichier que les fonctions du TP, ou avoir par exemple un fichier `TP1.py` et un second fichier `testsTP1.py`. D'autre part, vous pouvez inclure des tests avec des valeurs fixes, et d'autres tests aléatoires. Pour cela, vous pouvez utiliser la bibliothèque `random`. Par exemple

```
from random import randrange
a = randrange(10, 20)
```

permet de renvoyer un entier (pseudo[1]-)aléatoire de l'ensemble $\{10, \dots, 19\}$.

Exercice 1. *Modulo en Python*

1. En testant tous les possibilités, déterminer le comportement de l'opérateur `%` sur des entrées positives et négatives. *Définir mathématiquement le résultat renvoyé par* `%`.
2. Effectuer le même travail avec l'opérateur `//` de calcul de quotient.
3. Écrire une fonction `DivEucl(a,b)` qui calcule la division euclidienne de a par b, avec un reste positif, compris entre 0 et $|b|-1$. *Faire appel aux opérateurs* `%` *et* `//` *!*

Exercice 2. *Algorithme d'Euclide*

1. Implanter l'algorithme d'Euclide en versions récursive et itérative.
2. Comparer les temps de calcul pour des entiers aléatoires. L'une des deux fonctions est-elle plus rapide ? *Dans la console* `Ipython`*, les « commandes magiques »* `%time` *et* `%timeit` *permettent de mesurer des temps de calculs de manières très simple. Par exemple, il suffit d'écrire* `%time EuclideRec(a,b)` *pour mesurer le temps d'exécution de l'appel. La commande* `%timeit` *exécute plusieurs fois l'appel et renvoie une moyenne des temps de calcul (qui est donc moins sujette aux aléas).*
3. Implanter l'algorithme d'Euclide étendu en version récursive.

Exercice 3. *Entiers de Fibonacci*
La célèbrissime suite de Fibonacci $(F_n)_{n\geq 0}$ est définie par $F_0 = 0$, $F_1 = 1$ et $F_{n+2} = F_n + F_{n+1}$ pour $n \geq 0$. Elle a la propriété que les *pires* entrées pour l'algorithme d'Euclide sont lorsque les deux entiers sont deux éléments consécutifs de la suite de Fibonacci.

1. Écrire une fonction `FiboNaive(n)` qui calcule *naïvement* l'entier F_n. Vérifier que cette fonction est bien *naïve* : quelle est la plus grande valeur de n pour laquelle `FiboNaive` s'exécute en moins d'une seconde ?
2. Écrire une version itérative efficace `Fibo(n)` qui calcule le couple (F_n, F_{n+1}). *L'idée est de partir avec le couple* (F_0, F_1)*, et à chaque itération de remplacer le couple* (F_i, F_{i+1}) *par* (F_{i+1}, F_{i+2}).
3. Tester les trois algorithmes d'Euclide de l'exercice précédent sur des éléments consécutifs de la suite de Fibonacci. À partir de quelle valeur (approximative) de n les algorithmes récursifs *plantent*-ils quand ils sont appelés avec F_n et F_{n+1} ?

1. Le *vrai* aléatoire n'existe pas sur un ordinateur (classique). Le sujet de la génération de nombres *pseudo-aléatoires*, c'est-à-dire qui ont l'*air aléatoire*, est passionnant mais hors sujet dans ce cours !

Exercice 4. *Euclide étendu itératif*

Pour pouvoir calculer les coefficients de Bézout pour des grands entiers, il faut écrire une version itérative de l'algorithme d'Euclide étendu. Pour cela, on note $r_0 = a$, $r_1 = b$, et $r_{i+2} = r_i \bmod r_{i+1}$ pour tout $i \geq 0$ la suite des restes calculés par l'algorithme d'Euclide (on a alors $\text{PGCD}(a,b) = r_k$ où r_k est le dernier reste non nul). L'objectif est de calculer, pour tout i, des coefficients u_i et v_i tels que $r_i = u_i \cdot a + v_i \cdot b$. Les cas $i = 0$ et $i = 1$ peuvent être caclulés directement. Ensuite, si on sait que $r_i = u_i \cdot a + v_i \cdot b$ et $r_{i+1} = u_{i+1} \cdot a + v_{i+1} \cdot b$, en écrivant $r_{i+2} = r_i - q r_{i+1}$ où $q = r_i \operatorname{quo} r_{i+1}$, on en déduit des valeurs pour u_{i+2} et v_{i+2}.

1. Compléter l'algorithme EUCLIDEETENDUIT qui implante cette stratégie. *Attention, pour ne pas multiplier les variables r_i, u_i et v_i, on en utilise uniquement six : r^0, r^1, u^0, u^1, v^0 et v^1. À tout instant, si r^0 contient r_{i+1}, alors r^1 contient r_i.*

   EUCLIDEETENDUIT(a,b) :
   1. $(r^0, r^1) \leftarrow (a, b)$
   2. $(u^0, v^0) \leftarrow (\ldots, \ldots)$ # *on veut* $r^0 = u^0 \cdot a + v^0 \cdot b$
   3. $(u^1, v^1) \leftarrow (\ldots, \ldots)$ # *on veut* $r^1 = u^1 \cdot a + v^1 \cdot b$
   4. Tant que $r^1 \neq 0$:
   5. $\quad q \leftarrow r^0 \operatorname{quo} r^1$
   6. $\quad (r^0, r^1) \leftarrow (r^1, \ldots)$
   7. $\quad (u^0, u^1) \leftarrow (u^1, \ldots)$
   8. $\quad (v^0, v^1) \leftarrow (v^1, \ldots)$
   9. Renvoyer (r^0, u^0, v^0)
2. Écrire une fonction `EuclideEtenduIt(a,b)` qui correspond à cet algorithme, et la tester sur des (grands !) éléments consécutifs de la suite de Fibonacci.

Exercice 5. *Factorisation*

Comme on l'a vu, tout entier n peut se décomposer de manière unique en produit de facteurs premiers. On va écrire un algorithme pour calculer cette décomposition. Pour cela, on commence diviser n par 2 tant qu'il est pair, et on obtient ainsi la puissance de 2 dans la décomposition de n. On continue ensuite avec tous les entiers par ordre croissant, jusqu'à avoir $n = 1$.

1. Pourquoi peut-on ne considérer que les entiers impairs, après 2, au lieu de les prendre tous ?
2. Montrer que si n n'est divisible par aucun entier $k \leq \sqrt{n}$, on peut s'arrêter car n est premier.
3. Écrire une fonction `Factorisation(n)` qui prend entrée n et calcule une décomposition de n en produit de facteurs premiers avec la statégie décrite. *Vous pouvez tester vos résultats avec la commande* `factor` *dasn le shell.*