

# TP4

## Arithmétique

5 décembre 2017

### 1. Un nombre est-il premier ?

- (a) Chercher comment on obtient en python le quotient et le reste de la division euclidienne d'un entier par un autre entier.
- (b) Ecrire la définition d'un nombre premier.
- (c) Parmi les entiers suivants, dire lesquels sont premiers : 1001, 2017, 3001, 49999, 89999.
- (d) Ecrire une fonction python `is_prime` qui prend en argument un entier  $n$  et qui renvoie `true` si  $n$  est premier, `false` sinon.
- (e) Les nombres de Fermat  $F_n$  sont définis par  $F_n = 2^{2^n} + 1$  ; les nombres  $F_0, F_1, \dots, F_5$  sont ils premiers ?

### 2. Crible d'Erasthote, distribution des nombres premiers.

- (a) A l'aide du crible d'Erasthote, calculer la liste de tous les nombres premiers inférieurs à 200.
- (b) Ecrire une fonction python `primes` qui prend en argument un entier  $n$  et qui renvoie la liste de tous les premiers inférieurs à  $n$ .
- (c) Calculer la liste de tous les premiers inférieurs à 1000 ; écrire cette liste dans un fichier `primes.txt`, dix nombres par ligne.
- (d) On note  $\pi(n)$  le nombre d'entiers premiers inférieurs à  $n$ . Représenter graphiquement  $\pi(n)$  en fonction de  $n$  pour  $n$  variant de 2 à 1000. Sur le même graphique, rajouter la fonction  $\frac{n}{\log n}$ . Qu'observe-t'on ? voir le théorème des nombres premiers.
- (e) Créer en python la table suivante, la remplir et l'écrire dans un fichier texte.

$n$	$\pi(n)$	$\frac{n}{\log n}$
$10^1$		
$10^2$		
$10^3$		
$10^4$		
$10^5$		
$10^6$		

### 3. Factorisation d'un entier en premiers.

- (a) Ecrire le théorème fondamental de l'arithmétique.
- (b) Calculer la décomposition en facteurs premiers de 924.
- (c) Ecrire une fonction python `factors` qui prend en arguments un entiers  $n$  et qui renvoie la liste, dans l'ordre croissant, des facteurs premiers de  $n$ , chaque facteur étant répété autant de fois que nécessaire. Ainsi, pour  $n = 60$ , on obtiendra `[2, 2, 3, 5]`.

### 4. pgcd de deux entiers, identité de Bézout, algorithme d'Euclide.

- (a) Ecrire la définiton du pgcd de deux entiers.
- (b) En utilisant la décomposition en facteurs premiers, calculer le pgcd des deux nombres  $a = 4864, b = 3458$ .

- (c) Ecrire l'énoncé de l'identité de Bézout
- (d) A l'aide de l'algorithme d'Euclide étendu, calculer  $d$ , le pgcd des deux nombres  $a = 4864, b = 3458$ , ainsi que les coefficients de Bézout  $x, y$  tels que  $xa + yb = d$ .
- (e) Ecrire une fonction python `euclide` qui prend en arguments deux entiers  $a, b$  et qui renvoie  $x, y, d$ , où  $d$  est le pgcd de  $a, b$  et  $x, y$  les coefficients de Bézout.

## 5. Algorithme RSA.

Ecrire les codes python, mais ne pas rédiger de compte rendu sur cette partie.

- (a) Voir le fonctionnement du chiffrement RSA.
- (b) A l'aide de la fonction python `random.randint()` et de votre fonction `is_prime`, choisir deux entiers premiers distincts  $p, q$  supérieurs à  $2^{35}$  et inférieurs à  $2^{36}$ .
- (c) On pose  $n = pq$ ; vérifier que  $n$  est supérieur à  $2^{70}$ . On sait bien sûr que  $n$  n'est pas premier, mais que se passe-t'il lorsqu'on exécute `is_prime(n)`? Expliquer le phénomène observé.
- (d) Voir la définition de la fonction indicatrice d'Euler. Que vaut  $\phi(n)$  pour l'entier  $n$  défini ci-dessus?
- (e) Choisir aléatoirement un entier naturel  $e$  premier avec  $\phi(n)$ , et strictement inférieur à  $\phi(n)$ ;  $e$  est appelé exposant de **chiffrement**.
- (f) Calculer l'entier naturel  $u$  inverse de  $e$  modulo  $\phi(n)$ , et strictement inférieur à  $\phi(n)$ ;  $u$  est appelé clé de **déchiffrement**; indication : si  $d$  est le pgcd de  $e$  et  $\phi(n)$  et si  $u, v$  sont les coefficients de Bézout tels que  $d = ue + v\phi(n)$ , alors  $u$  est inverse de  $e$  modulo  $\phi(n)$ ; de plus dans l'identité de Bézout on peut toujours choisir  $u$  compris entre 1 et  $n$ .
- (g) Choisir  $M$  un entier naturel strictement inférieur à  $n$ ;  $M$  représente le message que l'on veut chiffrer. Calculer  $C = M^e$  modulo  $n$  - indication : il faut soit implémenter la méthode d'exponentiation rapide en effectuant les produits modulo  $n$ , soit utiliser la fonction python built-in `pow()`;  $C$  est le message chiffré, utilisant la clé publique de chiffrement  $e, n$ .
- (h) Vérifier que l'on retrouve bien  $M$  en effectuant l'opération  $C^u$  modulo  $n$ ; le couple  $u, n$  est la clé privée de déchiffrement.
- (i) En rassemblant les éléments précédents, écrire une fonction python `rsa_key()`, qui ne prend pas d'argument, et qui renvoie les entiers  $e, u, n$ , où  $e, n$  servira de clé publique de chiffrement et  $u, n$  de clé privée de déchiffrement.
- (j) Combien y a t'il de chaines de 10 caractères ASCII distinctes possibles, sachant qu'un caractère ASCII est codé sur 7 bits? En utilisant la fonction built-in `ord()`, écrire une fonction python `tenchars2int` qui prend en argument une chaine de caractères `tenchars` de longueur 10 au plus et qui renvoie un entier  $M$ , inférieur strictement à  $2^{70}$ , codant la chaine de caractères `tenchars`. De même, en utilisant la fonction built-in `chr()`, écrire une fonction python `int2tenchars` qui prend en argument un entier  $M$  inférieur strictement à  $2^{70}$  et qui renvoie une chaine de 10 caractères ASCII. Vérifier que les fonctions `tenchars2int` et `int2tenchars` sont bien inverses l'une de l'autre.
- (k) En rassemblant les éléments précédents, écrire une fonction python `chiffre()` qui prend en arguments une chaine de caractères de longueur arbitraire et qui renvoie cette chaine chiffrée au moyen d'une clé publique de chiffrement générée par la fonction `rsa_key()` - indication : découper le texte à chiffrer en segments de longueur 10. De même, écrire une fonction python `dechiffre()` qui prend en arguments un texte chiffré, ainsi que la clé privée de déchiffrement associée à la clé publique, et qui renvoie le texte déchiffré. Tester les fonctions `chiffre()` et `dechiffre()` sur des exemples.