FEUILLE D'EXERCICES 3
Les anneaux $\mathbb{Z}$ et $\mathbb{Z}/n\mathbb{Z}$

Exercice 1 – Expérimenter les commandes suivantes.

```
ZZ (c'est l'anneau ℤ)
A=IntegerModRing(12); A
A.list()
A.list_of_elements_of_multiplicative_group()
euler_phi(6)
a=A(5)
a^(10^10)
a.multiplicative_order()
```

Définir les anneaux `Zx` $= \mathbb{Z}[x]$, `Zxy` $= \mathbb{Z}[x,y]$ et `Z6t` $= (\mathbb{Z}/6\mathbb{Z})[t]$.

Exercice 2 – [Test de Fermat]
On rappelle le théorème de Fermat : si n est un nombre premier, alors pour tout a premier à n,

$$a^{n-1} \equiv 1 \mod n.$$

1) Soit $n = 2^{2^8} + 1$. En utilisant le théorème précédent, montrer que n n'est pas premier. Le vérifier en utilisant les fonctions `is_prime` et `factor`.

2) Même exercice avec $n = \frac{10^{41}+1}{11}$.

3) En vous inspirant des exemples précédents, écrivez une fonction probabiliste qui teste plusieurs a possibles (le nombre de a à tester pourrait être une entrée de votre fonction) et renvoie `True` ou `False`. Votre algorithme peut se tromper, mais on veut que s'il renvoie `False`, c'est qu'on a trouvé un a qui contredit le théorème de Fermat. Dans ce cas là on a une preuve que n n'est pas premier. Par contre, si l'algorithme renvoie `True`, on veut avoir de bonnes chances d'espérer que n soit premier, mais ça peut être faux (et l'algorithme ne nous renvoie pas de "preuve" que ce soit vrai). [1]

Exercice 3 – [Test de primalité]

1) Soit $n = 2^{2^4} + 1$. Quel est l'ordre multiplicatif de 3 modulo n ? En déduire que n est premier.
(Indice : vous pourrez commencer par prouver que n est premier si et seulement s'il existe un élément d'ordre (multiplicatif) $n-1$ modulo n.)

1. Il existe des nombres composés n tels que pour tout a premier avec n, $a^{n-1} = 1 \bmod n$ (i.e., la réciproque du petit théorème de Fermat n'est pas vraie pour ces nombres). Ces nombres sont appelés nombres de Carmichael. Votre algorithme probabiliste va forcément se tromper pour ces nombres, mais en dehors de ces cas rares, on espère qu'il marche plutôt bien.

2) En vous inspirant de la question précédente, écrire un algorithme probabiliste qui prend en entrée un entier n de la forme $n = 2^k \cdot 3^\ell + 1$ et renvoie `True` si n est premier (ainsi qu'un témoin que n est premier, comme 3 dans la question précédente). Votre algorithme peut se tromper : on veut que si l'algorithme renvoie `True`, alors on a une preuve que n est premier ; par contre, si l'algorithme renvoie `False`, il peut se tromper et n peut-être quand même premier (même si on veut que cela n'arrive pas avec bonne probabilité).
(C'est l'inverse de l'algorithme de l'exercice précédent : on autorise les erreurs quand n est premier mais pas quand n est composé)

3) Si n est premier, quelle est la probabilité que votre algorithme se trompe ? Et si n est composé ?

4) En combinant cet algorithme avec l'algorithme de l'exercice précédent, prouvez si les entiers suivants sont premiers ou composés :

(1) $n = 2^{12} \cdot 3^{11} + 1$

(2) $n = 2^{32} \cdot 3^{96} + 1$

(3) $n = 2^{23} \cdot 3^{52} + 1$

(4) $n = 2^{73} \cdot 3^{17} + 1$

(5) $n = 2^{89} \cdot 3^{113} + 1$

Exercice 4 – [Nombres de Mersenne]
Pour tout nombre entier p, on note $M_p = 2^p - 1$. De tels nombres M_p sont appelés nombres de Mersenne. Si M_p est premier, on dit que M_p est un nombre de Mersenne premier.

1) Montrer que si M_p est premier, alors p est premier.
On admet le théorème suivant.

Théorème (Test de Lucas)**.** *Soit p un nombre premier impair. Soit L la suite définie de la manière suivante. $L_1 = 4$, et pour tout $n \geqslant 1$, $L_{n+1} = L_n^2 - 2$. Alors M_p est premier si et seulement si M_p divise L_{p-1}.*

2) Écrire une procédure Lucas qui étant donné un nombre premier p détermine si M_p est premier. Pour comparer l'efficacité de votre procédure avec celle de vos voisins, essayer par exemple `time Lucas(19937)`

3) En utilisant la procédure ci-dessus, établir la liste des 20 plus petits nombres de Mersenne premiers, ou plutôt la liste des 20 plus petits nombres premiers p tels que M_p est un nombre de Mersenne premier.

Exercice 5 – [Méthode ρ de Pollard]
Cette méthode vise à trouver un facteur non trivial d'un entier n donné. Soit $f(x) = x^2 + 1$. On choisit un entier x_0, et on pose $y_0 = x_0$. Pour $n \geqslant 0$, on pose $x_{n+1} = f(x_n) \bmod n$ et $y_{n+1} = f^2(y_n) \bmod n$. À chaque étape, on calcule $\text{pgcd}(x_i - y_i, n)$, et on arrête dès que ce pgcd est différent de 1. Si en plus il est différent de n, c'est un facteur non trivial de n.

1) Écrire une procédure qui utilise cette méthode pour trouver un facteur non trivial d'un entier donné n.

2) Appliquez cette procédure à $10^{20} + 67$ et comparez entre vous les temps d'exécution obtenus.

Exercice 6 – [Textbook-RSA]
On décrit dans cet exercice un schéma de chiffrement à clé publique appelé "textbook RSA" (c'est une variante simplifiée de RSA qui n'est pas sécurisée, mais contient les idées principales).
Génération de clés. X choisit deux nombres premiers $p_1 = \frac{10^{31}+1}{11}$ et $p_2 = \frac{10^{53}+1}{11}$ et calcule $n = p_1 \cdot p_2$. X choisit également un entier $e = 17$, premier avec $p_1 - 1$ et $p_2 - 1$. La clé secrète de X est la paire (p_1, p_2), tandis que (n, e) est sa clé publique, qu'il envoie à Y. (Ce choix de n est trop petit, et puis p_1 et p_2 sont trop particuliers. On peut factoriser n trop facilement ... Tant pis. C'est juste une expérience. Dans la pratique, il faudrait choisir n qui soit difficile à factoriser.)
Chiffrement. Y veut envoyer un message s à X, avec s un entier modulo n. Pour cela, Y calcule $t = s^e \bmod n$ et envoie t à X.

1) Proposez et implémentez un algorithme de déchiffrement pour que X puisse retrouver s à partir du chiffré t et de sa clé secrète (p_1, p_2).

2) Testez votre algorithme sur $t = 1111$. Combien valait s ?
(On doit trouver $s = 650966664 \ldots 3537769$.)

3) Normalement, votre algorithme de déchiffrement n'utilise pas vraiment (p_1, p_2) mais juste la valeur de $\varphi(n) = (p_1 - 1) \cdot (p_2 - 1)$. Montrer que si n est connu, alors c'est équivalent de connaître (p_1, p_2) ou de connaître $\varphi(n)$ (i.e., expliquez comment retrouver efficacement (p_1, p_2) à partir de n et de $\varphi(n)$).

4) Si $n = 100003000700021$ et $\varphi(n) = 100002000600012$, utilisez la méthode de la question précédente pour retrouver p_1 et p_2. Comparez votre résultat avec `factor(n)`.

5) Reprenons les valeurs de p_1 et p_2 du début de l'exercice et supposons que Y veut envoyer un message $s < 10^4$. Expliquez comment un attaquant qui ne connaît que n et $e = 17$ peut retrouver le message s. Implémentez votre attaque.

Exercice 7 – [Exponentiation rapide]

Implémentez (et testez) l'algorithme d'exponentiation rapide, qui prend en entrée x et k et renvoie x^k. (Remarque : c'est assez facile si on fait un algorithme récursif)