TP1 sous Python: Les algorithmes classiques

En Python les entiers sont stockés sur la machine sous la forme de nombres en base 2. La valeur d'un entier n'est pas limitée par un nombre de bits fixé à l'avance (ce n'est pas le cas pour les réels) et peut s'étendre jusqu'à la limite de la mémoire disponible. Les opérations usuelles se font donc exactement dans les conditions définies dans le cours.

En plus de l'addition (+), de la soustraction (-), et de la multiplication (*) Python dispose des opérations sur les nombres entiers suivantes:

1. l'affichage de la représentation binaire d'un nombre entier en base 10: bin(x) . Le résultat est une chaîne de caractère précédé du préfixe 0b:

```
In [5]: int(0b110)
Out[5]: 6

In [6]: ch=bin(22)

In [7]: ch
Out[7]: '0b10110'

In [8]: len(ch)
Out[8]: 7

In [9]: ch[4]
Out[9]: '1'

In [10]: ch[1]
Out[10]: 'b'
```

   On voit dans cette suite de commandes que la chaîne de caractère ch peut être directement utilisée comme un tableau pour accéder aux chiffres de la représentations binaire du nombre 22. La longueur de l'entier x en base 2 est donnée par la commande int.bit_length(x). La longueur d'une chaîne de caractères ch est donnée par len(ch).

2. L'opération inverse est int(x): l'exécution de la commande int(0b110) donne le nombre entier 6.
3. Le quotient euclidien de la division euclidienne de a par b: a//b (le résultat est un nombre entier).
4. Le reste euclidien de la division euclidienne de a par b: a%b

Exercice 1:

1. Il existe une fonction permettant de calculer le pgcd de deux entiers mais nous vous demandons dans cette question de la reprogrammer en utilisant l'algorithme récursif **Euclide(a,b)** du cours.

2. Démontrez que $3^{1000} \wedge 2^{1000} = 1$ et vérifiez le résultat avec votre algorithme (3^{1000} s'écrit $3 * *1000$ en python).
3. Démontrez que $\left(3^{1000} \times 2^{10}\right) \wedge 2^{1000} = 1024$ et vérifiez le résultat avec votre algorithme.
4. Démontrez que $\left(2^{1000} + 1\right) \wedge \left(2^{1000} - 1\right) = 1$ et vérifiez le résultat avec votre algorithme.

Exercice 2:

1. Programmez l'algorithme **Euclide-Etendu(a,b)** du cours.

2. En utilisant votre algorithme déduisez-en l'inverse de 127 dans $(\mathbb{Z}/239\mathbb{Z})^*$.
3. Démontrez par le calcul que l'inverse de $2^{10} - 1$ dans $\left(\mathbb{Z}/\left(2^{10} + 1\right)\mathbb{Z}\right)^*$est 2^9 et vérifiez ce résultat à l'aide de votre algorithme.
4. Quelle est l'inverse de 2^{500} dans $\left(\mathbb{Z}/\left(2^{1000} - 1\right)\mathbb{Z}\right)^*$? Expliquez comment vérifier ce résultat à l'aide de votre algorithme.

Exercice 3:

1. Programmez la fonction **Exponentiation-modulaire(a,b,n)** du cours.

2. En utilisant votre algorithme déduisez-en $739^{8247}\,(1023)$ (on demande le plus petit entier positif x tel que $x \equiv 739^{8247}\,(1023)$)

3. Soient $a = 2^{1000} - 17$, $b = 2^{1000} + 1$ et $n = 2^{2000} + 1$:

   (a) pour chacun des entiers a, b et n précédents donnez la longueur de l'écriture en base 2.

   (b) On considère le plus petit entier positif x tel que $x \equiv a^b(n)$. Quelle est la longueur maximale de x en base 2?

   (c) On considère l'algorithme Python suivant:

```
142 def PuissModulaire(a,b,n):
143     i=0
144     x=1
145     while i<b:
146         x=(a*x)%n
147         i=i+1
148     return x
```

   Pourquoi cet algorithme est-il inadapté au calcul de $x \equiv a^b(n)$ pour les valeurs du début de la question?

   (d) Lancez l'algorithme **Exponentiation-modulaire(a,b,n)** pour les valeurs du début de la question. Conclusion?

4. En utilisant votre algorithme donnez la valeur de $\omega(23)$ dans $(\mathbb{Z}/67\mathbb{Z})^*$puis celle de $\omega(29)$.