UNIVERSITÉ DE RENNES 1

Théorie des nombres

TP n°1 : Prise en main, tests de primalité

Critere de Fermat et nombres de Carmichael

1. Ecrire une fonction $temoin(a, n)$ qui renvoie vrai ou faux selon que a est un témoin de Fermat pour la non primalité de n.
   a. En écrire une première version qui calcule simplement a^{n-1} et le réduit modulo n.
   b. En écrire une deuxième version (disons $temoin2(a, n)$) qui introduit l'anneau $\mathbb{Z}/n\mathbb{Z}$, et calcule a^{n-1} dans cet anneau.
   c. Vérifier le résultat pour de petites valeurs de a et de n.
2. Quel est le plus petit nombre premier $n > 10^{20}$? Lire la documentation pour trouver le nom de la fonction qui etant donné un entier n renvoie le nombre premier suivant... (dans la doc de référence, chercher dans " Standard Commutative Rings -> Elements of the ring $\mathbb{Z}$ ")
3. 11 est-il témoin de Fermat pour le nombre n ci-dessus ? Tester vos 2 fonctions et expliquez ce qu'il se passe.
4. Ecrire une fonction qui etant donné un entier n, dit s'il est de Carmichael ou non (on pourra utiliser un test de Sage pour déterminer si n est premier).
5. Vérifier que le plus petit nombre de Carmichael est 561, et lister tous les nombres de Carmichael $< 10^4$. Combien y en a-t-il ?
6. Quel est le plus petit nombre impair n, non premier, tel que tel que 2 n'est pas témoin de Fermat pour la non primalité de n ? Quel est le suivant ?

Critère de Miller Rabin

1. Ecrire une fonction $temoinMR(a, n)$ qui (sous l'hypothese que n est impair), renvoie vrai ou faux selon que a est un temoin de Miller-Rabin pour la non primalité de n.
2. Combien 99 a-t-il de temoins de Miller-Rabin ?
3. Trouver le plus petit entier n impair, non premier, tel que 2 ne soit pas un temoin de Miller-Rabin pour sa non-primalité.
4. Ecrire un test probabiliste de primalité, qui se trompe avec une probabilité $< 10^{-10}$.
5. Parmi les nombres non premiers entre 1 et 1000, trouver celui qui a la plus petite proportion de temoins de Miller-Rabin.

Exemples TP1 THNO

Pour commencer

Il est recommande de lire le 1er chapitre du sagebook: http://sagebook.gforge.inria.fr.

Une fois tapee la commande, pour effectuer un calcul, faire shift-entree:

```
1+1
```

```
2
```

Pour elever a une puissance, c'est l'operateur **

```
2**5
```

```
32
```

Approximation numerique de la fraction 1/3

```
a=1/3
N(a)
```

```
0.333333333333333
```

Le ppcm se dit lcm en anglais, et pgcd se dit gcd...

```
print lcm(12,15), gcd(12,15)
```

```
60 3
```

Pour tester l'egalite de 2 entiers (ou de 2 objets): == et !=

```
print 1==2, 2!=3
```

```
False True
```

Si on a un objet, ici, disons l'entier 120, on peut lui appliquer certaines methodes.
Les methodes qu'on peut lui appliquer dependent de la nature de l'objet. Ici, la methode "factor" s'applique a un entier.
Cette methode n'est pas disponible pour d'autres types d'objets, comme une liste, ou un nombre complexe...
Ca s'appelle une syntaxe "orientee objet"...

```
120.factor()
```

```
2^3 * 3 * 5
```

facon orientee objet d'approximer notre fraction a=1/3 avec 50 chiffres significatifs: on applique la methode "n()" a l'objet a=1/3

```
a.n(digits=50)
```

```
0.33333333333333333333333333333333333333333333333333
```

S'il y a plusieurs lignes de calcul, seul le resultat de la derniere est affiche

```
1+1
n=factorial(10)
n
```

```
3628800
```

n est il premier ?

```
n.is_prime()
```

```
False
```

```
#factorisation en facteurs premiers
n.factor()
```

```
2^8 * 3^4 * 5^2 * 7
```

```
#exposant de 2 dans la factorisation de n
n.valuation(2)
```

```
8
```

```
print
n.valuation(3),n.valuation(5),n.valuation(7),n.valuation(11)
```

```
4 2 1 0
```

L'anneau Z, Z/nZ

L'anneau des entiers **Z** est deja defini: il s'appelle ZZ. Il y a aussi QQ (les rationnels), RR, CC pour les reels et complexes

```
ZZ
```

```
Integer Ring
```

```
#un element de ZZ au hasard dans l'intervalle [1,100[
ZZ.random_element(1,100)
```

```
11
```

On peut creer l'anneau Z/nZ. Ici, on le stocke dans la variable R.

```
print n
R=Integers(n)
print R
```

```
3628800
Ring of integers modulo 3628800
```

Pour savoir si R est un corps, on applique la methode "is_field" a l'anneau R. En anglais, corps se

dit "field"...

```
print R.is_field()
```

```
False
```

```
# On definit a=-5 dans Z/nZ
a=R(-5)
a
```

```
3628795
```

```
a^2
```

```
25
```

```
#a est il inversible (dans Z/nZ) ? [les unités d'un anneau
sont un autre nom pour ses element inversibles]
a.is_unit()
```

```
False
```

```
b=R(11)
b.is_unit()
```

```
True
```

Puisque b est inversible, calculons son inverse

```
c=b**-1
print c,c**-1,b*c
```

```
329891 11 1
```

```
#l'ordre de b dans le groupe multiplicatif
b.multiplicative_order()
```

```
8640
```

```
b**8640
```

```
1
```

Oups ! a n'est pas inversible !

```
a.multiplicative_order()
```

```
Traceback (click to the left of this block for traceback)
...
ArithmeticError: multiplicative order of 3628795 not defined
it is not a unit modulo 3628800
```

```
#choisir un element de R au hasard
R.random_element()
```

```
1070465
```

```
a=R.random_element()
a.is_unit()
```

```
True
```

Pourquoi obtient-on le plus souvent "false" ci-dessus ? Reponse:

```
print "nombre d'inversibles de R: ",R.unit_group_order()
print "nombre total d'elements de R: ",R.order()
print "rapport:", (R.unit_group_order()/R.order()).n()
```

```
nombre d'inversibles de R:  829440
nombre total d'elements de R:  3628800
rapport: 0.228571428571429
```

```
R.multiplicative_group_is_cyclic()
```

```
False
```

Boucles, fonctions, tests etc

Dans beaucoup de langages, les blocs d'instructions sont separes par des accolades, ou des begin-end. En sage comme en python, c'est l'indentation qui permet de savoir ou s'arrete un groupe d'instructions.

Voici une boucle où i decrit les entiers de l'intervalle [0,10[. Remarquez les ":" apres le "for", et l'indentation: le print "fin" n'est pas dans la boucle

```
for i in range(10):
    print i
print "fin"
```

```
0
1
2
3
4
5
6
7
8
9
fin
```

Ici, i parcourt les entiers de [1,20[de 2 en 2. Dans j, on calcule donc la somme des entiers impairs

```
j=0
for i in range(1,20,2):
    j=j+i
    print i,j
```

```
1 1
3 4
5 9
7 16
9 25
11 36
13 49
15 64
```

```
17 81
19 100
```

```
#attention le i dans la boucle est un entier python: 'int',
pas un entier sage:
print i
i.is_prime()
```

```
19
Traceback (click to the left of this block for traceback)
...
AttributeError: 'int' object has no attribute 'is_prime'
```

```
#conversion de i en entier sage:
i2=Integer(i)
print i2,i2.is_prime()
```

```
19 True
```

liste des nombres premiers entre 10 et 19. Remarquez les ":" apres le "for" le "if" et le "else", et l'indentation chaque fois.
Noter la conversion de i en entier Sage.

```
for i in range(10,20):
    if Integer(i).is_prime():
        print i, "est premier"
    else:
        print i, "n'est pas premier"
```

```
10 n'est pas premier
11 est premier
12 n'est pas premier
13 est premier
14 n'est pas premier
15 n'est pas premier
16 n'est pas premier
17 est premier
18 n'est pas premier
19 est premier
```

Definition d'une fonction. Remarquez encore les ":" et l'indentation.

z est une variable locale.

```
def f(x,y):
    z=x+2*y
    return z
```

```
f(1,2)
```

```
5
```

```
z
```

```
Traceback (click to the left of this block for traceback)
```

```
...
NameError: name 'z' is not defined
```