Informatique en CPGE (2018-2019)
Corrigé TP 5 : représentation des nombres

1 Nombres entiers naturels

Exercice 1 : division euclidienne.

```
def divise(a,b):
    r=a
    q=0
    while r>=b:
        q=q+1
        r=r-b
    return q,r
```

Exercice 2 : codage d'un entier naturel.

```
def base2(n):
    """ conversion en base 2 d'un nombre naturel
    n est de type int
    le retour ch de type str"""
    ch=''
    while n>0:
        ch=str(n%2)+ch
        n=n//2
    return ch

def test(ch):
    """ teste si le nombre entré est bien en base 2 """
    for c in ch: # le nombre ne peut contenir que des 0 ou des 1
        if c!="0" and c!="1":
            return False
    return True

def base10(ch):
    """ conversion de la base 2 vers décimal,
    entrée de type str
    sortie d de type int"""
    d=0
    for i in range(len(ch)):
        d+=(2**i)*int(ch[len(ch)-1-i])
    return d
```

2 Nombres entiers relatifs

Exercice 3

```
for i in range(16):
    i-=8
    print(i,bin(i),~i+1,bin(~i+1))
```

Exercice 4 : codage d'un entier relatif.

```
def relatif(r,n):
    """ r est l'entier relatis à coder,
    n est le nombre de bits utilisés"""
    if r<-2**(n-1) or r>2**(n-1)-1:
        return False
    if r<0:
        r+=2**(n)
    # conversion en base 2
    code=''
    while r>0:
        code=str(r%2)+code
        r=r//2
    # ajout des 0 éventuels
    while len(code)<n:
        code='0'+code
    return code
```

3 Nombres réels

Exercice 5 : approximations d'un réel.

```
u=3
for i in range(100):
    u=0.5*(u+2/u)
print('u100 = ',u)

from math import sqrt
from decimal import Decimal
print('racine de 2 : ',sqrt(2))
print('racine de 2 au carré : ',Decimal(sqrt(2)*sqrt(2)))
```

Exercice 6 :
Corrigé et compléments

1. $1,0 = 1,0 \times 2^0$ est codé par $s = 0$, $e = 0 + 1023$ et $m = 0$ soit 0 011 1111 1111 0000 ... 0000 ;
   son successeur est donc : 0 011 1111 1111 0000 ... 0001 soit $1 + 2^{-52}$.
   $2^{-52} \simeq 2.220446049250313 \times 10^{-16}$ est la valeur de epsilon.
   $2,0 = 1,0 \times 2^1$ est codé par 0 100 0000 0000 0000 ... 0000 ;
   son successeur est 0 100 0000 0000 0000 ... 0001 soit $(1 + 2^{-52}) \times 2^1 = 2 + 2\times$epsilon.
   etc ...
   le successeur de 2^{52} est $(1 + 2^{-52}) \times 2^{52} = 2^{52} + 1$ (il n'y a pas de flottants entre les deux) le successeur de 2^{53} est $2^{53} + 2$ ...
   Conclusion : l'écart entre un flottant x strictement positif et son successeur est donc en général $2^{-52}(2^{Ent(\log_2(x))}) = 2^{Ent(\log_2(x))-52} \simeq x \times 2^{-52}$ (Ent = partie entière).

2. Le plus grand flottant est codé par 0 111 1111 1110 1111 …1111 ; l'exposant est $2046 - 1023 = 1023$, la mantisse tronquée $1/2 + 1/4 + 1/8 + ... + 1/2^{52} = 1 - 2^{-52}$ et le nombre vaut donc $(1 + 1 - 2^{-52}) \times 2^{1023} = (2 - 2^{-52}) \times 2^{1023} = 1.7976931348623157 \times 10^{308}$.
   Le plus petit flottant positif est codé par 0 000 0000 0001 0000 …0000 ;
   l'exposant est $1 - 1023 = -1022$, la mantisse tronquée 0 et le nombre vaut donc $1,0 \times 2^{-1022} = 2.225073858507202 \times 10^{-308}$.

   Pour tout $x \in [2^n; 2^{n+1}[$, avec $n \in \mathbb{Z}$, l'écart entre x et son successeur est : 2^{n-52}.

Exercice 7 : codage d'un flottant.

```
def flottant(x):
    # écriture du signe
    s='0'
    if x<0:
        s='1'
        x=-x
    # calcul de l'exposant et de la mantisse
    exp=0
    while x>=2:
        x/=2
        exp+=1
    while x<1:
        x*=2
        exp-=1
    # signe s, exposant exp, mantisse x
    # codage binaire du réel
    # exposant décalé
    exp+=1023
    # en binaire sur 11 bits (voir exercice 2)
    b=''
    while exp>0:
        b=str(exp%2)+b
        exp=exp//2
    long=11-len(b)
    for i in range(long):
        b='0'+b
    # mantisse tronquée
    x-=1
    m=''
    for i in range(52):
        x*=2
        if x>=1:
            m+='1'
            x-=1
        else:
            m+='0'
    ieee=s+' '+b+' '+m
    return ieee
```