DST 1 IPT
Corrigé

Exercice 1 - Représentation entiers et flottants.

1. **a.** On a, sachant que la lettre B code le chiffre 11 :

$$(2B4)_{16} = 2 \times 16^2 + 11 \times 16^1 + 4 \times 16^0 = 2 \times 258 + 11 \times 16 + 4 = 512 + 176 + 4 = 692.$$

b. Il s'agit d'un nombre codé sur 8 bits. On lit les puissances de 2 qui sont présentent :

$$(11010010)_2 = 128 + 64 + 16 + 2 = 210.$$

2. On peut poser la soustraction, avec la règle « $1 + 1 = 0$, et je retiens 1 ». On trouve :

$$(10010111)_2 + (01011001)_2 = (11110000)_2.$$

On pouvait aussi convertir en base 10, additionner, on trouve 240.

3. **a.** On a $0 \leqslant x < 128$ dont il suffit de coder x en binaire de manière standard sur 8 bits. Or on a

$$x = 64 + 7 = 64 + 4 + 2 + 1 = (01000111)_2$$

On a $-128 \leqslant -96 < 0$ donc on représente $y = -96$ par complément à 2 par l'écriture en binaire de $y + 256 = 160$ sur 8 bits. Or on a :

$$160 = 128 + 32 = (10100000)_2.$$

Ainsi, -96 est représenté par complément à 2 sur 8 bits par $(10001100)_2$.

b.
- Que vaut le nombre M ? Ici, on est sur $n = 8$ bits, et $2^{8-1} = 128$. En base 2, $(01101010)_2 = 106$, donc, puisque $0 \leqslant 106 < 128$, on a $M = 106$.
- Que vaut le nombre N ? Cette fois-ci, la séquence commence par un 1, donc on sait qu'on a codé un nombre négatif. En base 2, $(11000101)_2 = 197 > 128$, donc le nombre N vérifie

$$N + 2^8 = 197 \iff N = 197 - 256 = -59.$$

4. **a.** Puisque l'exposant décalé E est codé sur 4 bits, on a

$$0 \leqslant E \leqslant 2^4 - 1 = 15.$$

Si l'exposant décalé vaut 0, alors on a

$$2^{0-7} = 2^{-7} = \frac{1}{128}$$

tandis que si tous les bits de l'exposant décalé sont à 1, on a $E = 15$, et

$$2^{15-7} = 2^8 = 256.$$

b.
- Que vaut le nombre A ? Ici le bit de signe vaut 0. Les 4 bits suivants code l'exposant : $E = (1110)_2 = 14$. La mantisse est codée en puissance négative de 2 par les 4 derniers bits 1111 :

$$m = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16}.$$

Finalement,

$$A = 2^{14-7} \times (1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16}) = 2^7 \times (1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16}) = 2^7 + 2^6 + 2^5 + 2^4 + 2^3 = 128 + 64 + 32 + 16 + 8 = 248$$

- Que vaut le nombre B ? Cette fois-ci, la séquence commence par un 1, donc on sait qu'on a codé un nombre négatif. Les 4 bits suivants code l'exposant : $E = (0111)_2 = 7$. La mantisse est codée en puissance négative de 2 par les 4 derniers bits 1010 :

$$m = 1 + \frac{1}{2} + \frac{1}{8}.$$

Finalement,

$$B = -2^{7-7} \times (1 + \frac{1}{2} + \frac{1}{8}) = -(1 + 0.5 + 0.125) = -1.625.$$

Exercice 2 - Algorithmes et programmation.

1. La variable p est multipliée par i à chaque étape de la boucle, pour i dans range(1,n+1). Attention, à la fin de la boucle *for*, la variable i vaut n et pas $n+1$. La fonction renvoie donc

$$1 \times \ldots \times n = n!$$

Remarque : On peut bien sûr donner la réponse sans explication, mais on verra qu'on peut faire une vraie preuve de ce que vaut le résultat, à l'aide d'un invariant de boucle.

2. Voici le code demandé :

```
energie=(1/2)*masse*energie**2
if energie >=1000000 :
    print("grande énergie cinétique")
elif 100000<energie and energie<100000 :
    print("énergie cinétique modérée")
else :
    print("énergie cinétique faible")
```

3. **a.** On fait une boucle *for* :

```
s=0
for i in range(1,11) :
    s=s+i**2
```

b. On fait une double boucle *for* imbriquée :

```
s=0
for i in range(0,101) :
    for j in range(1,101) :
        s=s+i*j
```

4. La commande *in* permet de savoir si un élément est dans une liste :

```
est_present=x in L
```

Si on l'avait oubliée, on pouvait aussi parcourir la liste avec une boucle *for*.

5. **a.** On fabrique une variable max_L (initialisée avec la première valeur de la liste), qui va stocker la plus grande valeur de la liste et on parcourt la liste avec une boucle *for*. Si on trouve une valeur plus grande que la valeur stockée, on actualise avec cette valeur.

```
def RechMax(L) :
    max_L=L[0]
    for i in range(1,len(L)) :
        if max_L<L[i] :
            max_L=L[i]
    return max_L
```

b. Même concept, mais avec deux variables :

```
def Rech2Max(L) :
    if L[0]>L[1] :
        max2_L=[L[0],L[1]]
    else :
        max2_L=[L[1],L[0]]
    for i in range(2,len(L)) :
        if max2_L[1]<L[i] and max2_L[0]>=L[i] :
            max2_L[1]=L[i]
        elif max2_L[0]<L[i] :
            max2_L[1]=max2_L[0]
            max2_L[0]=L[i]
    return max2_L
```

6. Comme on ne sait pas quand le critère d'arrêt va avoir lieu, on utilise une boucle *while* :

```
def temps_1() :
    essais=0
    res=0
    while res==0 :
        essais+=1
        res=random.randint(0,1)
    return(essais)
```