Exercice 1 : Vrai ou Faux

1) En python, le plus grand nombre de type *int* est plus grand que le plus grand nombre de type $float$.
2) Tous les décimaux appartenant à [1,2] ont une représentation exacte en flottant.
3) Avec la représentation signe, exposant, mantisse, pour effectuer une division par deux en binaire, on retire 1 à l'exposant
4) Avec la représentation signe, exposant, mantisse, si x est une puissance de 2 alors sa mantisse ne contient que des 0.
5) L'expression $1 + 2 ** (-5) == 1$ a la valeur $True$ si le codage des flottants utilise une mantisse codée sur 4 bis
6) Si les flottants sont écrits avec une mantisse codée sur 10 bits alors $1 + 10 ** (-2) == 1$ a la valeur $True$.

1) Oui, le type int utilise des blocs de 64 bits dont le nombre est limité par la capacité de la RAM. Pour une RAM de 16Go, on peut donc atteindre $10^{10^{10}}$. Pour les flottants, les 10bits réservés à aux exposants positifs limitent plus sévèrement la valeur maximale à $2^{2^{10}} \approx 2^{1000} \approx (2^{10})^{100} \approx 10^{300}$
2) Non, on a typiquement une résolution est de l'ordre de $2^{-52} \approx 10^{0.3*52} \approx 10^{-16}$
3) Oui, si $x = (1,m)2^p$ alors $x' = \frac{x}{2} = (1,m)2^{p-1}$ donc on retranche 1 à l'exposant et la mantisse est la même. Cependant, on gagne d'un « ordre de grandeur » en précision.
   Testons cette affirmation avec 4,4 et 2,2

$$(4,4)_{10} = \left(100,\overline{0110}\right)_2 = +1{,}0001\ 1001\ 1001 \ldots 1010 \times 2^2$$

$$(2,2)_{10} = \left(10,\overline{0011}\right)_2 = +1{,}0001\ 1001\ 1001 \ldots 1010 \times 2^1$$

4) Oui, cela s'apparente au codage d'un entier
5) Oui car autour de la valeur unité, la résolution est donc de 2**(-4)
6) Faux, car sur 10 bits et autour de 1, la résolution est $2^{-10} \approx 10^{-3}$

Exercice 2 : Transcodage

1) Convertir $x = -2{,}895 \times 10^2$ en binaire. On donnera le résultat sous la forme d'une notation scientifique
2) Idem avec $x = -(10{,}125)_{10}$
3) Convertir $x = (1{,}01011)_2 \times 2^2$

Il faut d'abord écrire ce nombre en distinguant partie entière et décimale :

- Partie entière : $-(289)_{10} = -[100100001]_2$
- Partie décimale : $-0{,}5 = -[1]$
- Donc $x = -(1{,}001000011)_2 \times 2^8$

Il faut d'abord écrire ce nombre en distinguant partie entière et décimale :

- Partie entière : $-(10)_{10} = -[1010]_2$
- Partie décimale : $-0{,}125 = -[001]_2$
- Donc $x = -(1{,}010001)_2 \times 2^3$

On a $x = (1{,}01011)_2 \times 2^2 = 2^2 + 2^0 + 2^{-2} + 2^{-3} = 5{,}375$

Exercice 3 :

On souhaite montrer que l'expression $0{,}1 + 0{,}1 + 0.1 > 0{,}3$ renvoie $True$.

1) Justifier que $(0{,}1)_{10} \approx +1, \overline{1001} \times 2^{-4}$.

En machine, $(0{,}1)_{10}$ est arrondi à la valeur supérieure et la matisse précédente devient (sur 52 bits) :

$$(0{,}1)_{10} \approx (1, \quad 1001 \quad 1001 \quad \dots \quad 1001 \quad 1001 \quad 1010)_2 \times 2^{-4}$$

2) Calculer 0,1+0,1+,01
3) Montrer que $(0{,}3)_{10} = +1, \overline{0011} \times 2^{-2}$
4) Pourquoi $0{,}1 + 0{,}1 + 0.1 > 0{,}3$?

Montrons que $0.1 + 0.1 + 0.1! = 0.3$:

- Calculons $x = 0{,}1$

$(0{,}1)_{10} = \left(0{,}00011\ \overline{0011}\right)_2 = +1, \overline{1001} \times 2^{-4}$

Avec arrondi :

$(0{,}1)_{10} \approx (1, \quad 1001 \quad 1001 \quad \dots \quad 1001 \quad 1001 \quad 1010)_2 \times 2^{-4}$

- Exprimons 0,3 :

$(0{,}3)_{10} = \left(0{,}01001\ \overline{1001}\ \right)_2 = +1, \overline{0011} \times 2^{-2}$

$(\ 1, \ 0011 \quad 0011 \ \dots \quad 0011 \quad 0011 \quad 0011) \times 2^{-2}$

Pas d'arrondi ici car le digit négligé est 0 :

$(\ 1, \ 0011 \quad 0011 \ \dots \quad 0011 \quad 0011 \quad 0011) \times 2^{-2}$

- Calculons :$0.1 + 0.1 + 0.1$:

$$\begin{array}{l} 1, \quad 1001 \quad 1001 \ \dots \quad 1001 \quad 1001 \quad 1010 \\ 1, \quad 1001 \quad 1001 \ \dots \quad 1001 \quad 1001 \quad 1010 \\ 1, \quad 1001 \quad 1001 \ \dots \quad 1001 \quad 1001 \quad 1010 \\ 100, \quad 1100 \quad 1100 \ \dots \quad 1100 \quad 1100 \quad 1110 \end{array}$$

Donne en écriture scientifique :

$0{,}1 + 0{,}1 + 0{,}1 = (1, \quad 0011 \quad 0011 \ \dots \quad 0011 \quad 0011 \quad 0011\ 10) \times 2^{-2}$

Avec arrondi

$0{,}1 + 0{,}1 + 0{,}1 = (1, \quad 0011 \quad 0011 \ \dots \quad 0011 \quad 0011 \quad 0100\ ) \times 2^{-2}$

A comparer avec 0,3

$0.3 = (1, \quad 0011 \quad 0011 \ \dots \quad 0011 \quad 0011 \quad 0011)) \times 2^{-2}$

L'erreur s'explique par une meilleure précision des petits nombres et par les arrondis successifs.

Donc l'écart entre 0,1+0,1+0.1 et 0,3 est de l'ordre de :

$$2^{-52} - 2^{-53} - 2^{-54} = 2^{-54}(4 - 2 - 1) = 2^{-54} \approx 6 \times 10^{-17}$$

Exercice 4 :

On représente des nombres réels avec une écriture signe, mantisse, exposant. Dans cet exercice, on utilise six bits $b_0b_1b_2b_3b_4b_5$ pour cette écriture.

- Le bit b_0 représente est 0 pour + et 1 pour –
- b_1, b_2, b_3 code la mantisse tronquée
- b_4, b_5 code l'exposant uniquement positif (pas d'exposant décalé, b_4 est le bit de poids fort)

1) Donner les valeurs $b_0b_1b_2b_3b_4b_5$ pour $x = 6{,}0$
2) Quel est le plus petit nombre positif que l'on peut écrire ainsi ?
3) Quel est le plus petit nombre strictement positif que l'on écrire ainsi ?
4) Quel est le successeur de 2,0 ?
5) Combien de nombres appartenant à l'intervalle [1,2[peut-on écrire ?

1) $b_0 = 0$
   Pour la mantisse $(6{,}0)_{10} = (110{,}0)_2 = 1{,}1 \times 2^2$
   $donc\ b_1 = 1, b_2 = 0, b_3 = 0$
   Pour l'exposant : $b_4 = 1, b_5 = 0$
   Donc le mot binaire associé est 010010
2) Ce plus grand nombre aura pour représentation $011111 \equiv +1{,}111 \times 2^3 \approx 8 + 4 + 2 + 1 = 15$
3) Le plus petit nombre aura pour représentation : $000000 \equiv +1{,}000 \times 2^0 = 1$
4) Il faut trouver le pas pour $x = 2{,}0$
   $(2{,}0)_{10} = (10{,}0)_2 = 1 \times 2^1 \equiv 000001$
   Le dernier digit de la mantisse tronquée est sur 2^{-2} donc la valeur suivante sera $2{,}0 + 2^{-2} = 2{,}25$
5) $(1)_{10} = (1)_2 = (0\ 000\ 00)$ et $(2{,}0)_{10} = (10{,}0)_2 = 1{,}0 \times 2^1 = 0\ 000\ 01$ et donc 8 valeurs possibles :

- 0 000 00=> +1
- 0 001 00 => +1,125
- 0 010 00=> +1,25
- 0 011 00=> +1,375
- 0 100 00=>+1,5
- 0 101 00=> +1.625
- 0 110 00=>+1,75
- 0 111 00=>+1,875
- 0 000 01

Exercice 5 :

Que renvoie le code ci-dessous ?

```
def f():
    x=1.0
    while x!=x+1:
        x=x*2
    return x
```

Nous avons vu que doubler un nombre revient à augmenter d'une unité un exposant donc perdre un ordre de grandeur (en binaire en précision)

$$x = 1{,}0000..\times 2^0 \rightarrow x = 1{,}0000 \times 2^1 \rightarrow \cdots \rightarrow x = 1\ldots.\times 2^p$$

Si on peut écrire $x = 2^p = 1{,}0000\ldots 2^p$ le dernier digit de la mantisse tronqué portera sur 2^{p-52}.

Si $p = 53$ alors le pas est supérieur à 1 ainsi $x == x + 1$ et on trouve $x \approx 10^{16}$

Exercice 6 :

Ecrire le nombre 3,625 en binaire.

$$(3{,}625)_{10} = (11{,}101)_2 = 1{,}1101 \times 2^1 = 2^1 + 2^0 + 2^{-1} + 2^{-3}$$

Exercice 7 :

On utilise la norme IEEE-754 vue en cours

1) Comment est codé le nombre -4.5 ?
2) Quel est le nombre réel codé par :
   1 011111111110 1000 0000…0000 ?

1) $(-4{,}5)_{10} = (-100{,}1)_2 = -1{,}001 \times 2^2 \equiv$
   1 10000000001 0010 $\overline{0000}$
2) − pour le signe, 0111 1111 1110 correspond à 1022 donc $p = -1$ et pour la mantisse on a donc 1*2^-1+1*2^-4 et donc le nombre codé est -0,75

Exercice 8 : Erreurs d'arrondies

Le 25 février 1991, une batterie anti-missile a laissé passer un missile scud à cause d'une erreur de calcul. L'horloge interne du système de défense comptait le temps en dixièmes de secondes. La mantisse tronquée des temps mesurées était calculée en 24 bits.

1) Faire la conversion de la période d'échantillonnage T_e en binaire (notation scientifique).
2) Créer une liste d'entiers contenant tous les bits de la mantisse tronquée (en pensant à arrondir correctement).
3) Ecrire un programme permettant de calculer la valeur de T_e réellement mesurée ?
4) Sachant que l'horloge est réinitialisée tous les 100h, donner le temps de décalage maximal avec une horloge parfaitement réglée.
5) Le missile se déplaçant à 1676m/s, quelle est l'imprécision sur sa position ?

$(0,1)_{10} \approx (1,\quad 1001\quad 1001\quad \dots\quad 1001\quad 1001\quad 1010)_2 \times 2^{-4}$

```
liste=[1,0,0,1]*5+[1,0,1,0]
def conversion(liste):
    resultat=0
    for i in range(len(liste)):
        resultat=resultat+liste[i]*2**(-5-i)
    return resultat+2**-4

print(conversion(liste)*10*60*60*100-0.1*10*60*60*100)#5ms d'écart
print(0.005364418029785156*1676)#9m
```

liste2=[1,0,0,1]*(52//4-4)+[1,0,1,0]