INSA Hauts-de-France **3A**
ICy **2022/2023**

Architecture avancée

TP 1 : représentation et arithmétique en virgule flottante

Exercice 1 : Représentation des flottants

Commencez par récupérer le fichier `float_coding.py` et importez-le. Ce fichier contient des fonctions permettant de convertir un flottant en un entier dont la représentation binaire est identique à celle de la représentation 64 bits `IEEE-754` du flottant (la norme utilisée par Python).

Q 1. Utilisez la fonction `float_coding.floatbin` pour les flottants 4, 8 et 12. Le résultat étant un grand entier, il est peu compréhensible. Affichez la représentation binaire de cet entier, pour chacun des résultats de `float_coding.floatbin`. Quelle est la longueur de la représentation binaire ? Pourquoi n'est-elle pas systématiquement de 64 bits ? Expliquez pourquoi certains bits changent dans les trois représentations binaires.

Attention
L'affichage en binaire sert ici juste à ce que nous, humains, comprenions quelle est la représentation de ces flottants. Lorsque vous programmez des fonctions il est inutile, et pas souhaitable comme vu précédemment, de convertir un entier en une chaîne binaire. À la place il faut utiliser des opérations logiques pour effectuer les actions voulues sur la représentation binaire.

Q 2. Ajoutez 1 aux trois entiers retournés par la fonction `float_coding.floatbin` à la question précédente. En utilisant la fonction `float_coding.binfloat` retrouvez à quel flottant correspondent les représentations binaires de ces entiers.

Q 3. Expliquer la différence que vous observez entre les flottants originaux (4, 8 et 12) et ceux obtenus en ayant ajouté 1 à l'entier correspondant à leur représentation binaire.

Q 4. Faites une fonction `float_sign` qui prenne en paramètre un entier dont la représentation binaire correspond à celle d'un flottant (tel que renvoyé par la fonction `float_coding.floatbin`) et qui retourne le signe du réel (1 ou -1). Votre fonction utilisera des opérations logiques (directement ou indirectement en utilisant une fonction réalisée précédemment).

Q 5. Faites une fonction `float_exponent` qui prenne en paramètre un entier dont la représentation binaire correspond à celle d'un flottant (tel que renvoyé par la fonction `float_coding.floatbin`) et qui retourne l'exposant du réel (compris entre -1022 et 1023). Votre fonction utilisera des opérations logiques mais pas d'instruction itérative ou conditionnelle, ni d'autres fonctions.

Q 6. Faites une fonction `float_mantissa` qui prenne en paramètre un entier dont la représentation binaire correspond à celle d'un flottant (tel que renvoyé par la fonction `float_coding.floatbin`) et qui retourne la mantisse du réel (supérieure ou égale à 1 et inférieure à 2). Votre fonction utilisera des opérations logiques mais pas de fonctions.

Q 7. Ecrivez une fonction `float_notation` qui prenne en paramètre un flottant et qui renvoie un triplet (signe, exposant, mantisse) correspondant au flottant passé en paramètre.
Votre fonction utilisera notamment la fonction `float_coding.floatbin`.

```
>>> float_notation (3.5)
(1, 1, 1.750000)
```

Nous allons maintenant voir l'impact d'un changement de bit dans un réel.
Q 8. Pour cela nous allons commencer par faire une fonction `change_a_bit` qui prend en paramètre un entier et qui renvoie une copie de cet entier dans laquelle, à la position donnée en paramètre, le bit a été remplacé par le bit inverse. Cette fonction doit uniquement utiliser des opérateurs logiques.

Q 9. Ecrivez une fonction `change_a_bit_in_float` qui prend en paramètre un réel ainsi que la position à modifier dans la représentation binaire et qui renvoie la valeur du réel modifié.

Vous utiliserez donc les fonctions `float_coding.floatbin` pour obtenir la représentation binaire du flottant sous forme entière. Puis, après avoir fait la modification, vous utiliserez la fonction `float_coding.binfloat` afin de faire l'opération inverse.

Q 10. Prenons le réel 2. Quelle modification implique une modification du bit de poids fort ? Du bit de poids faible ? Du bit en position 55 (la position 0 étant le bit de poids faible, comme pour `change_a_bit_in_float`) ? Pourquoi ?

Exercice 2 : Arithmétique

Python utilise la représentation `IEEE-754` double précision, i.e., sur 64 bits.

Q 1. Evaluez l'expression booléenne `.1 + .2 == .3`.

Q 2. Utilisez `float_notation` pour les flottants 0.1, 0.2 et 0.3. Vérifiez que les valeurs données sont correctes (c'est-à-dire, par exemple, qu'on a bien `.1 =` $s \times 2e \times m$, où s, e et m sont les trois valeurs renvoyées par `float_notation` pour 0.1).

Q 3. Python arrondit l'affichage des flottants (attention il arrondit l'affichage mais ne touche pas à la représentation elle-même). En utilisant format, forcez l'affichage de 20 chiffres après la virgule pour la mantisse sur le résultat de `float_notation` :

```
sem = float_notation (.1)
print('({0[0]:d}, {0[1]:d}, {0[2]:.20f})'.format(sem))
```

Faites ceci pour .1, .2, .3 ainsi que pour le résultat de .1 + .2. Expliquez pourquoi on a `.1 + .2 != .3`

L'erreur de représentation fait référence au fait que certaines fractions décimales (la plupart, en fait) ne peuvent pas être représentées exactement comme des fractions binaires. Ainsi des langages comme Python, Perl, C, C++, Java, Fortran, et bien d'autres n'affichent souvent pas le nombre décimal exact que vous attendez.

Par exemple $\frac{1}{10}$ n'est pas exactement représentable comme une fraction binaire. La machine s'efforce de convertir 0,1 à la fraction la plus proche possible de la forme $\frac{J}{2^N}$, où J est un entier contenant exactement 53 bits.

Q 4. Trouvez la meilleure valeur de N.

Q 5. Utilisez la fonction `divmod` pour trouver J et ainsi déterminer la fraction la pus proche de 0.1.

Q 6. Qu'obtenez-vous en multipliant cette fraction par 10^{55} ?

Q 7. Expliquez le résultat de l'instruction Python `format(0.1, '.17f')`.

Q 8. Utilisez les fonctions `as_integer_ratio` et `Decimal.from_float` pour vérifier.