Informatique 2ème année, CPBX, TD4

Paul Dorbec

Ce TD vous permet de fouiller un peu dans le codage des nombres décimaux en Python. Des questions sont posées auxquelles vous répondrez dans un test dans Moodle.

1 Calculs en virgule flottante

En Python, les entiers sont codés de façon exacte. Cependant, les nombres réels ne peuvent pas être codés de façon exacte. Ils sont donc codés sous la forme d'un nombre à virgule muni d'un exposant, du type $m * 2^e$. Dans cette description, m est un nombre compris entre 1 et 2, sous forme binaire. Le champs de bits servant à coder cette valeur est appelée la *mantisse*. Les bits restants servent à coder l'exposant. Ainsi, le nombre 3 se code en floattant sous la forme de $1.1 * 2^1$ c'est à dire en décimal $(1 + \frac{1}{2}) * 2$.

1.1 Constat initial

Avec cette méthode de codage des nombres, on remarque vite que Python est obligé d'utiliser des approximations, notamment sur le nombre de décimales utilisées.

Question 1.1 *Affectez x à 0.1 et y à 0.2. Affichez x, y et $x + y$. Qu'en déduisez vous ?*

1.2 À la recherche de la mantisse

On va tout d'abord chercher la taille de la mantisse. Par l'exploration des petits nombres. Pour cela, on va produire des nombres qui s'écrirait en binaire $1.000\ldots001$. Lorsque la distance entre les deux 1 dépasse la mantisse, ce nombre sera en fait égal à 1.

Question 1.2 *Écrire une fonction* `cherche_precision()` *pour calculer à partir de quel n python considère que 1 et $1 + 2^{-n}$ sont le même nombre. On*

affichera le n obtenu. En déduire la taille de la mantisse *en nombre de bits en plus du 1 initial. (À renseigner dans le test.)*

Question 1.3 *Pour confirmer ce comportement, voici une autre expérience. Affectez à x le nombre qui s'écrit en remplissant exactement la mantisse de 1. Cela représente une écriture avec un 1 de plus que la taille de la mantisse trouvée précédemment. (On ignore pour l'instant l'exposant, qui n'a pas d'intérêt). Affichez x. Affichez* $x - 1$ *normalement, les deux nombres diffèrent de 1. Affichez* $x + 1$*. Dans ce cas, le nombre bascule en une écriture* $1.00\ldots000 * 2^{m+1}$*, donc le résultat semble cohérent. Si vous ajoutez un 1 de plus par contre, on sort de la partie représentable et on doit lire pour* $x + 2$ *la même valeur que pour* $x + 1$*. Notez que le calcul* $x + 3$ *donne la bonne valeur ! Qu'en pensez vous ?*

1.3 À la recherche de l'exposant

Je vous rappelle que Python code `1.` comme un nombre en virgule flottante et `1` comme un entier. Il n'impose pas de limite de taille sur les nombres entiers, donc ne vous aventurez pas à chercher le plus grand entier codable. On peut en revanche chercher le plus grand nombre en virgule flottante.

Question 1.4 *Nous allons cherchez le plus petit n tel que* $1 \times 2^n \neq 1. \times 2^n$*. Pour cela, on affecte 1 à x et* $1.$ *à y. Puis, on multiplie x et y par 2 jusqu'à ce que les deux nombres diffèrent. Quelle est la valeur finale de y ? de x ? Combien de multiplication par 2 avez vous faites ? (À renseigner dans le test.)*

Question 1.5 *Faites le même test pour le cas où x ety sont négatifs. Obtenez vous les mêmes résultats ? (À renseigner dans le test.)*

Question 1.6 *Et pour des exposants négatifs ? On prend* $y = 1.$ *et on divise y par deux tant que la valeur obtenue n'est pas zéro. À combien de division atteint-on zéro ? (À renseigner dans le test.)*

Question 1.7 *Comment pourriez-vous expliquer cette valeur ? Où est-ce que je vous ai escroqué ?* ;-)