Représentation de données : Booléens, entiers, flottants et caractères

19 juin 2019

P. Rousselin

1 Booléens

Dans ce qui suit, on note T et F deux choses distinctes, T étant interprétée comme la valeur « vraie » et F comme la valeur « fausse » (on pourrait utiliser aussi 1 et 0, ...).

Exercice 1 : Fonctions booléennes

1. Énumérer, décrire et interpréter toutes les opérations booléennes *unaires*, c'est-à-dire les fonctions qui prennent un argument booléen et retournent un booléen.
2. Faire de même avec les opérations binaires, c'est-à-dire les fonctions qui prennent deux arguments booléens et retournent un booléen.
3. Montrer les lois de De Morgan : pour tous booléens x et y,
$$\neg(x \vee y) = (\neg x) \wedge (\neg y) \quad \text{et} \quad \neg(x \wedge y) = (\neg x) \vee (\neg y).$$
4. Montrer que la fonction NAND (non-et, c'est-à-dire $(x, y) \mapsto \neg(x \wedge y)$) permet de reconstruire toutes les opérations booléennes unaires et binaires.
5. Si x et y sont deux booléens, comment retrouver x à partir de la connaissance de $x \oplus y$ et de y ? (l'opérateur $\oplus$ est le XOR). En déduire une méthode de chiffrage d'une suite de bits.

Exercice 2 : En python

1. Définir les fonctions python `xor` et `nand`, de deux variables booléennes, à l'aide des opérateurs booléens de python.
2. Écrire une fonction python `table_verite` qui prend en argument une fonction booléenne de deux variables et retourne sa table de vérité sous la forme d'une liste de deux listes de deux booléens. Par exemple la table de vérité de `et` est de la forme `[[0, 0], [0, 1]]`.
3. Écrire une fonction `afficher_table_verite` qui affiche une table de vérité sous la forme donnée précédemment.
4. Écrire une fonction python `memes_operateurs` qui prend en arguments deux opérateurs booléens binaires et retourne `True` s'ils sont égaux, `False` sinon.
5. Montrer les lois de De Morgan (on pourra utiliser les lambda-expressions).

Exercice 3 : Évaluation des opérateurs booléens en python

Évaluer les expressions suivantes dans `python3` et commenter.

1. `x = 0; x != 0 and 1/x > 3`
2. `x = 0; 1/x > 3 and x != 0`
3. `x = 0; x == 0 or 1 / x > 42`
4. `x = 0; 1 / x > 42 or x == 0`

2 Entiers

On commence par voir l'écriture en base $b \geq 2$ des entiers naturels (où b est un entier naturel).

Tout entier naturel n peut s'écrire de façon unique sous la forme

$$n = a_m b^m + a_{m-1} b^{m-1} + ... + a_1 b + a_0,$$

où, pour tout i entre 0 et m, a_i est un entier compris entre 0 et $b-1$ et a_m est non nul (sauf si n est nul, auquel cas on impose $m = 0$ et $n = a_0 = 0$). Les entiers a_m, a_{m-1}, ... a_0 sont les *chiffres* de la représentation de n en base b et $m+1$ est son nombre de chiffres.

Exercice 4 :

1. Écrire la fonction python `nb_chiffres(base, n)` correspondant à votre fonction et la tester.
2. Écrire une fonction python `chiffre(n)` correspondant à l'aide suivante (on n'aura besoin que de $n \leq 16$ donc vous pouvez vous contenter de considérer ce cas) :

```
chiffre(n)
    convertit l'entier n (entre 0 et 36) en un chiffre (↩
       caractère) entre 0 et Z

    chiffre: int -> str
    Exemples:
        >>> chiffre(12)
        'C'
        >>> chiffre(6)
        '6'
```

3. Écrire une fonction `dec_en_base(base, n)` qui prend deux entiers, dont le premier est la base (pas trop grande) et retourne une liste de (chaînes de) caractères qui sont les chiffres de la décomposition de n dans la base b.
4. Écrire une fonction `chiffre_vers_entier` qui prend un chiffre comme donné par la fonction chiffre et retourne l'entier correspondant.
5. Écrire une fonction `dec_vers_entier(base, decomp)` qui prend une base et une décomposition comme celle donnée par la fonction `dec_en_base(base, n)` et retourne l'entier correspondant.
6. Écrire une fonction `chaine_dec_en_base(base, n)` qui prend une base et un entier et retourne la décomposition sous forme de *chaîne de caractère* dans l'ordre usuel. Par exemple,

```
>>> chaine_dec_en_base(10, 167)
'167'
```

Exercice 5 : Représentation binaire

1. Écrire la fonction python `dec_binaire(n)` qui décompose en binaire un entier naturel n. La décomposition doit être sous la forme d'une *liste de booléens* (et l'élément numéro 0 de la liste est le chiffre a_0 de la décomposition).
2. Écrire la fonction python `chaine_dec_binaire(n)` qui retourne cette fois une chaîne de caractères composée de `0` et de `1` dans l'ordre d'écriture usuel. Par exemple,

```
>>> chaine_dec_binaire(12)
'1100'
```

3. Pour i entre 0 et 15, afficher `chaine_dec_binaire(i)`.
4. En déduire l'écriture en base 16 des entiers écrits en base 2 : `10110100` et `011110110101` (vous pouvez vérifier avec les fonctions précédemments écrites).

Les entiers python sont des objets assez à part en informatique, avec leur taille arbitraire (limitée, tout de même par la mémoire vive de la machine). Les entiers manipulés par les processeurs ont, à l'heure actuelle, souvent une taille limitée à 32 bits (*binary digit*, chiffre binaire).

Pour simplifier nos affichages, on se contentera de tableaux de 8 bits, mais le travail est le même sur des entiers plus grands.

Exercice 6 : Addition binaire

1. Écrire les fonctions `suc(bin)` (respectivement `pred(bin)`) qui prennent comme argument un tableau (une `list`) de 8 bits (`True` ou `False`) et retournent, sous la même forme son successeur (respectivement son prédécesseur).
   Ne pas se préoccuper pour l'instant des cas limites (nombre trop grand pour le successeur, ou trop petit pour le prédécesseur).
2. D'après vos fonctions, quel est le successeur de 255 et le prédécesseur de 0 ?
3. Écrire, avec des fonctions récursives, les fonctions `sommeR(bin1, bin2)` et `diffR(bin1,↩ bin2)` qui font la somme et la différence des entiers en binaire `bin1` et `bin2`
4. Écrire la fonction `multR(bin1, bin2)` qui multiplie les entiers en binaire `bin1` et `bin2` en utilisant les fonctions précédentes.
5. Faire de même pour la division.
6. Comment diviser/multiplier par 2 un entier en binaire ?
7. En utilisant uniquement les fonctions logiques XOR (ou exclusif) et AND (et logique), écrire une fonction `somme` qui additionne deux entiers en binaire sans utiliser de récursivité.

Exercice 7 : Entiers en complément à 2

1. Écrire (en base 10) l'entier représenté par l'octet `10011010` en complément à 2 sur 8 bit. Faire de même en le considérant cette fois comme un entier non signé.
2. Écrire sous forme binaire ou hexadécimale les entiers écrit en décimal -3, -128, 127 et 0 en utilisant la représentation en complément à deux sur 8 bits.

3. Écrire une fonction python qui prend comme argument un entier (de taille convenable à préciser) et retourne sa représentation en complément à 2 sur 8 bits.
4. Écrire une fonction python réciproque de la précédente, c'est-à-dire prenant un argument de 8 bits sous la forme d'un tableau de 8 booléens et retournant la valeur correspondante à l'interprétation de ces 8 bits comme un entier en complément à 2 sur 8 bits.
5. Vérifier sur des exemples que la fonction `somme` définie dans l'exercice précédent donne encore des résultats corrects dans le cas de la représentaiton en complément à 2.

Exercice 8 : Méthodes `from_bytes` et `to_bytes` de python

Lire la documentation des méthodes `int.from_bytes` et `int.to_bytes` et vérifier les résultats obtenus dans les deux questions de l'exercice précédent.

3 Nombres à virgule flottante

Les nombres à virgule flottantes (représentés sur 64 bits) sont de type `float` en Python.
On rappelle que dans un cas « normal », la valeur d'un flottant en double précision est

$$(-1)^s \times 2^{e-1023} \times (1, b_1 b_2 \cdots b_{52}),$$

où s est le bit de signe, e est la valeur associée aux 11 bits d'exposants (vus comme la représentation d'un entier positif) et b_1, b_2, ..., b_{52} sont les 52 bits de la mantisse.

Pour cette section, on importera le module `math`, afin de pouvoir utiliser les fonctions `math.isinf`, `math.isnan`, `math.sin`, `math.sqrt` et la constante `math.pi`. On importera aussi le module `sys` pour la constante `sys.float_info.epsilon` qui désigne le « epsilon machine ».

Exercice 9 :

1. Comment représenter les valeurs 0, 1 et 2^{-12} en flottant double précision ?
2. Le « epsilon machine » est, dans sa définition la plus courante, la différence entre la plus petite valeur représentable supérieure à 1 et 1. Quelle est sa valeur (exacte) ? Comparer avec la constante `sys.float_info.epsilon`.
3. Évaluer dans `python3` les expressions suivantes en nombres flottants où `eps` désigne le epsilon machine :

$$\sin(\pi) \quad \sqrt{2}^2 - 2 \quad 1 + \mathsf{eps} - 1 \quad 1 + \frac{\mathsf{eps}}{2} - 1 \quad 1 - 1 + \frac{\mathsf{eps}}{2} \quad 2^{1023} \quad 2^{1024}$$

$$-10^{-500} \quad \frac{1}{0} \quad 2^{-1023} \times 2^{-51} \quad 2^{-1023} \times 2^{-52} \quad 0,1 + 0,2$$

4. Expliquer les résultats obtenus.

4 Codages des caractères

Télécharger les fichiers `pluviose.txt`, `pluviose2.txt`, `pluviose3.txt`, `pluviose4.txt`

Exercice 10 :

1. Afficher chacun des fichiers, par différentes méthodes (dans un navigateur, un terminal avec la commande `cat`, etc) et voir les points qui posent problème.
2. Pour chacun des fichiers, voir la sortie de la commande (dans un terminal) `od -c -t x1 fichier`.
3. Voir, dans un terminal, pour chaque fichier, la sortie de la commande `file fichier`.
4. Voir le manuel de la commande `iconv` avec `man iconv`, puis convertir le fichier `pluviose4.txt` en « latin1 »et `pluviose3.txt` en « utf-8 ».

Exercice 11 :

1. Trouver, à la main, la représentation (en binaire) du caractère É en `utf-8`.
2. Vérifier avec l'instruction `list("É".encode("utf8"))`
3. Trouver un caractère d'usage (trop) fréquent qui n'est représenté dans la norme latin1 mais l'est en utf-8. Quelle est sa taille en utf-8 ?

Exercice 12 :

Écrire en python un programme permettant de convertir un texte codé en latin1 en utf-8. (On pourra bien sûr se contenter de certaines étapes de ce programme).