Types et nombres

1. Types en Python

Les valeurs sont dites *typées* en Python, ce qui signifie qu'elles sont différenciées, classées et stockées selon le type d'objet qu'elles représentent. Une valeur peut être de type entier, flottant, chaîne de caractères, etc... Des types similaires existent dans la plupart des langages de programmation. Leurs représentations en mémoire varient beaucoup d'un langage à l'autre mais ce sont souvent les mêmes types d'objets que l'on cherche à représenter. Le typage, en Python, est dynamique et déclaré seulement implicitement par l'utilisateur, contrairement à ce qui se passe en Java et en C, par exemple.

S'il existe déjà de nombreux types de base en Python et s'il est possible de créer soi-même des types qui répondent à un problème donné (voir le cours de sur la programmation orientée objet), il est nécessaire de bien connaître quelques types de base que l'on manipule tout le temps :

(1) Les entiers relatifs. Ils sont codés par le type int qui est arbitrairement long en Python 3.x.

(2) Les nombres décimaux (dyadiques, en fait). Ils sont codés selon la norme IEEE754, qui est rappelée plus loin. On parlera de « réel flottant » à précision simple (codé sur 32-bits) et à précision double (codé sur 64-bits). En Python 3.x, les flottants sont codés en double précision, donc sur 8 octets. Python ne l'indique pas explicitement quand on demande le type d'une variable contenant un flottant, mais numpy le fait. Ainsi, la commande `np.zeros((2,2)).dtype` renvoie `dtype('float64')`.

(3) Les booléens. Ceux-ci ne se composent que de deux états : vrai ou faux. Python utilise `True` et `False` et est comme toujours sensible à la casse. Ces types sont utilisés lors des tests logiques. De manière plus avancée, on peut aussi utiliser d'autres types comme booléens lors de tests ; par exemple, une liste sera considérée comme `True` si elle est non vide.

(4) Les nombres complexes. Ils sont codés sous la forme $a+bj$ où a et b sont deux réels (flottants) et j vérifie $j^2=-1$. Par exemple, `z = 2+1.j` affectera le complexe $2+i$ (en notation mathématique traditionnelle) à la variable `z`.

(5) Les caractères. Ils sont *a priori* codés sur 1 octet (voir cours de 1ère année).

2. Unités de mesures de la mémoire

Le bit est l'unité fondamentale de mesure de l'espace mémoire : un bit peut valoir 1 ou 0. Toutes les données traitées par un ordinateur peuvent être exprimées sous forme d'une suite de bits.

Les bits sont regroupés en octets : un octet comporte 8 bits. On peut coder $2^8 = 256$ informations différentes sur un octet.

Depuis 1998, les puissances de 10 sont également utilisées en informatique. On a ainsi :
- 1 ko (kilo octet) comporte 10^3 octets,
- 1 Mo (méga octet) comporte 10^6 octets,
- 1 Go (giga octet) comporte 10^9 octets,
- 1 To (tera octet) comporte 10^{12} octets,
- 1 Po (peta octet) comporte 10^{15} octets.

3. Entiers : théorie et pratique

3.1. **Généralités sur les bases.** L'écriture usuelle des nombres fait appel à la base 10, choix dicté davantage par l'anatomie humaine que par des considérations mathématiques (certaines civilisations firent d'autres choix ; par exemple, les Babyloniens utilisaient la base 60, plus pratique). Il est utile de

savoir écrire des nombres dans une autre base ; l'écriture en base b (b étant un entier naturel au moins égal à 2) d'un entier n fait appel aux chiffres de la base b, qui sont les entiers compris entre 0 et $b-1$.

On démontre facilement le théorème suivant :

$$\forall n \in \mathbb{N}^*,\ \exists!\ p \in \mathbb{N},\ \exists!\ (a_i)_{0 \leqslant i \leqslant p} \in [\![0, b-1]\!]^p \times [\![1, b-1]\!] \colon n = \sum_{i=0}^{p} a_i b^i.$$

L'entier 0 est codé sous la forme $0 = 0 \times 1$... Dans le cas où $b = 2$, on parle d'écriture binaire d'un entier :

$$\forall n \in \mathbb{N}^*,\ \exists!\ p \in \mathbb{N},\ \exists!\ (a_i)_{0 \leqslant i \leqslant p} \in [\![0, 1]\!]^p \times [\![1, b-1]\!] \colon n = \sum_{i=0}^{p} a_i b^i \text{ avec } a_p = 1.$$

Il s'agit du codage utilisé en principe en informatique, celui-ci étant naturellement un codage en regroupements de bits d'un entier.

3.2. **Les entiers relatifs en binaire.** Le paragraphe précédent n'indique pas comment coder les entiers relatifs. Deux conventions, incompatibles, sont possibles ; aussi doit-on préciser celle utilisée.

3.2.1. *Codage comme entier signé.*

Définition 1. *(Codage des entiers relatifs en convention signée sur n bits) Soient $n \geqslant 2$ un entier et p un entier relatif compris entre $-2^{n-1}+1$ et $2^{n-1}-1$. On appelle codage de p comme entier signé sur n bits une écriture sous forme binaire où le bit de plus fort poids (à gauche dans l'écriture) représente le signe de p avec la convention suivante :*

- *si p est positif, celui-ci vaut 0 ;*
- *si p est négatif, celui-ci vaut 1 ;*

et où les autres bits sont une écriture en base binaire de p (sur $n-1$ bits).

Cette convention, qui a l'avantage d'être très simple, présente plusieurs inconvénients :

- le nombre zéro peut être codé de deux manières différentes ;
- cette convention n'est pas compatible avec l'addition bit à bit.

3.2.2. *Codage par complément à deux.*

Définition 2. *(Codage en complément à deux (CPL2) sur n bits)*

Soient $n \geqslant 2$ un entier et p un entier relatif compris entre -2^{n-1} et $2^{n-1}-1$. Il existe une unique écriture de p sous la forme :

$$p = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i \text{ avec } a_i \text{ égal à 0 ou à 1.}$$

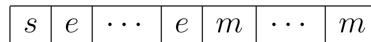
Cette convention est compatible avec l'addition bit à bit.

En Python 3.x, les entiers ont une taille arbitrairement grande. Ce n'est pas le cas dans les versions précédentes de Python, dans d'autres langages, ou même pour certaines librairies de Python, à l'instar de `numpy`, qui utilise différentes tailles d'entiers.

4. Flottants

Le codage des flottants obéit à des normes. La norme IEEE754, écrite en 1985 et révisée en 2008, définit des standards de codage. De manière simplifiée (on ne tient pas compte des cas particuliers dits « dénormalisés ») :

Définition 3. *(Signe, exposant, mantisse)*
On code un flottant x de manière générale sous la forme $x = (-1)^s(1+m)2^{e-d}$.
Dans cette écriture, s est appelé le signe, m la mantisse et e l'exposant.
- *$s \in \{0, 1\}$ est toujours codé sur un bit, qui est le bit de plus fort poids ;*
- *$e - d \in \mathbb{Z}$ est codé sur les bits suivants, comme entier naturel « décalé » d'un certain nombre, variable suivant le type de codage et sur un nombre de bits également variable ;*
- *$m \in [0, 1[$ est codé sur les bits de plus faible poids.*



s	e	$\cdots$	e	m	$\cdots$	m

Définition 4. *(Codage en simple précision aussi appelé binary32)*
L'entier e est codé sur 8 bits avec $d = 127$; m est codé sur 23 bits.

Définition 5. *(Codage en double précision aussi appelé binary64)*
L'entier e est codé sur 11 bits avec $d = 1023$; m est codé sur 52 bits.

La norme IEEE754-2008 définit également un codage en quadruple précision, sur 128 bits :

Définition 6. *(Codage en quadruple précision aussi appelé binary128)*
L'entier e est codé sur 15 bits avec $d = 16383$; m est codé sur 112 bits.

5. Exercices

Exercice 1. On suppose que les entiers sont codés sur n bits. Coder en binaire les nombres suivants : 0, 3, -1, -3, $2^{n-1} - 2$, $-2^n + 3$.
Effectuer manuellement la soustraction $113 - 91$ à partir d'un codage binaire sur 8 bits.

Exercice 2. Quel est le plus grand nombre codable en double précision ? Quel est l'écart absolu entre le plus grand nombre codable et le deuxième plus grand nombre codable en double précision ?

Exercice 3. Si l'on rentre `0.1 + 0.2 == 0.3`, Python retourne `False`. Expliquer pourquoi et écrire une fonction testant l'égalité de deux nombres décimaux.
Ce genre de plaisanteries peut avoir des conséquences fâcheuses. Voir par exemple
http ://ta.twi.tudelft.nl/nw/users/vuik/wi211/disasters.html

Exercice 4. Expliquer le comportement suivant :

```
>>> n = 1e16
>>> 1/n - 1/(n+1)
0.0
```

```
>>> n = 1e16
>>> 1/(n*(n+1))
9.999999999999999e-33
```

Exercice 5. Un signal $u(t)$ est numérisé par un convertisseur analogique-numérique 10 bits.
L'acquisition du signal consiste à obtenir N = 100 valeurs stockées en format flottant simple précision (32 bits). Pour les besoins du traitement numérique, on considère qu'il faut le double de la taille mémoire nécessaire à stocker le signal numérisé.
La gamme de micro-contrôleurs utilisés pour réaliser l'acquisition et les différents traitements numériques possèdent une mémoire interne à choisir parmi les valeurs suivantes : 512, 1024, 2048, 4096, 8192 octets.

Déterminer la quantité de mémoire nécessaire au traitement numérique du signal et en déduire la taille de mémoire interne nécessaire du micro-contrôleur.

Exercice 6. Une image non compressée est stockée dans une matrice de pixels. Un pixel codé en RGB est un triplet d'entiers codés sur 8 bits, correspondant à des proportions de rouge, vert et bleu. De combien de Mo doit-on disposer pour stocker une image en format 1600*1200, standard en HD ?

Exercice 7. (Bits de parité ; extrait banque PT 2015)

Un signal transmis peut comporter des erreurs dans les données transmises ; il est indispensable de détecter ces erreurs, et, dans la mesure du possible, de les corriger sans nécessiter une nouvelle transmission.

Une technique très simple pour s'assurer qu'une donnée binaire sera correctement lue par son récepteur est de lui adjoindre un bit de parité, égal, par définition, à :

- 0 si la donnée comporte un nombre pair de 1 (et donc si ses bits sont de somme paire)
- 1 si la donnée comporte un nombre impair de 1 (et donc si ses bits sont de somme impaire)

Après réception de la donnée, le récepteur recalcule le bit de parité, et le compare à celui que l'émetteur lui a adressé. Si la donnée n'a pas été altérée lors de la transmission, alors les deux bits de parité sont forcément identiques.

(1) Donner les bits de parité associés aux représentations binaires des entiers 5, 16 et 37.

(2) Écrire une fonction `parite(bits)` qui prend en argument une liste `bits`, constituée d'entiers valant 0 ou 1, et retournant l'entier 0 ou 1 correspondant à son bit de parité.

Les techniques de vérification les plus simples consistent à découper la donnée en blocs et à joindre un bit de parité à chaque bloc. Par exemple, certains protocoles transmettent sept bits de données pour un bit de parité.

(3) Donner un exemple d'erreur n'étant pas détectable par cette technique. Si une erreur a été détectée, est-il possible de la corriger sans retransmettre la donnée ?