TP1 : Représentation des nombres

1 Entiers

Exercice 1.1 :

1. Tester les commandes : 173//11 et 173%11 et indiquer leur lien avec la division euclidienne.
2. Trouver la représentation du nombre $\overline{14}^{10}$ en base 2, ainsi que de $\overline{1010111}^{2}$ en base 10.
3. Trouver la représentation du nombre $\overline{183}^{10}$ en base 16, ainsi que de $\overline{7E3}^{16}$ en base 10.
4. Exécuter les instructions : >>> bin(25) >>> int('11001',2) >>> 0b1010101 Quels résultats obtient-on ? Indiquer leur type. Valider vos résultats de la question 2b.
5. Même question avec : >>> hex(27) >>> int('1b',16) >>> 0x5ad9. Quels résultats obtient-on ? Indiquer leur type. Valider vos résultats de la question 3.

Exercice 1.2 :

Calculer la représentation sur huit bits de l'entier naturel 117, puis celle de son opposé.

Vérifier ces deux représentations avec Python. Les résultats sont-ils cohérents ? [1]

Exercice 1.3 : A lgorithme de Horner

On se retrouve donc très souvent à calculer des quantités polynômiales en b.

L'algorithme de Horner (encore utilisé aujourd'hui dans vos ordinateurs) permet de réduire ce coût en faisant :

$$a_0 + ... + a_7 b^7 = a_0 + b(a_1 + b(a_2 + b(a_3 + b(a_4 + b(a_5 + b(a_6 + ba_7))))))$$

C'est presque deux fois plus efficace !

1. Calculer avec la méthode de Horner
$$4 \times 5^5 + 3 \times 5^4 + 2 \times 5^3 + 3 \times 5^2 + 2$$
2. Ecrire une fonction récursive prenant en entrée un polynôme (sous la forme de la liste de ses coefficient $[a_0, ..., a_n]$) ainsi qu'un nombre b et calculant $a_0 + ... + a_n b^n$.
3. Combien de multiplications et additions sont nécessaires pour évaluer une quantité polynômiale de degré n via la méthode classique ? Via Horner ?

2 Entiers multiprécision et flottants

Exercice 2.1 : E ntiers multi-précision

Nous allons utiliser le module time de python. Pour cela il faut tapper

from time import time.

La fonction time() permet de récupérer le temps qu'il s'est écoulé (en secondes) depuis le 1er Janvier 1970

1. Etablir un protocole permettant de mesurer le temps que met un programme à s'exécuter grâce à la fonction time.

[1] Note : Python mémorise les entiers relatifs en complément à 2, mais les affiche différemment, sans doute pour en faciliter la lecture.

2. Comment expliquer la différence de temps d'exécution de chacun des deux blocs d'instructions suivants :

```
n=0
for i in range(1800):
    n=n*2**i

n=2**180
for i in range(1800):
    n=n*2**i
```

Exercice 2.2 :

- Observer l'effet des commandes suivantes :

1. >>> 1 2. >>> float(1) 3. >>> 1. 4. >>> int(1.278)

A quel type appartiennent ces nombres ?

- Observer l'effet des commandes suivantes :

>>> 1 + 2**-53 - 1 >>> 1 - 1 + 2**-53 >>>1.0 + 2**53 == 2**53 >>> (1+ 2**53) - 2**53 Expliquer le phénomène « d'absorption ».

Exercice 2.3 :

- Observer l'effet des commandes suivantes :

>>> 10.**100**100 >>> 2.**1023 >>> 2.**1024 >>>2**1024

- Expliquer ces différents résultats. Observer l'effet des commandes suivantes :

>>> 1 + 10**-15 > 1 >>> 1 + 10**-16 > 1

Exercice 2.4 :

Observer les résultats de :

1.1+2.7, 1.1*2.7, (1./3**2)*3**2 et (1./3**6)*3**6

Expliquer le phénomène.

Exercice 2.5 :

Représenter en base k un entier naturel exprimé en base 10 (avec$k < 9$). (Revoir si nécessaire la méthode vue en cours). L'objectif est d'obtenir l'expression de l'entier n en base k, sous la forme : $a_i...a_1a_0$.

Algorithme proposé en pseudo code :

Données : n, k **Tant que** n≠0 **faire** $a\leftarrow$ reste de la division euclidienne de n par k $n\leftarrow$ quotient de la division euclidienne de n par k **Résultat :** la représentation $a_i \ldots..a_1a_0$

Programmer cet algorithme en Python.