Cours_Representation_Entiers_Correction

November 17, 2021

Ce fichier est un notebook Python.

Il comporte deux types de cellules :

- les cellules d'édition dans lesquelles vous pouvez saisir du texte éventuellement enrichi de mises en formes ou de liens hypertextes avec la syntaxe du langage HTML simplifié qui s'appelle Markdown. Voir http://daringfireball.net/projects/markdown/ pour la syntaxe de Markdown.
- les cellules de code où l'on peut saisir du code Python3 puis le faire exécuter avec la combinaison de touches `CTRL + RETURN`

Une cellule peut être éditée de deux façons différentes :

- en mode *commande* lorsqu'on clique sur sa marge gauche qui est surlignée alors en bleu, on peut alors :
  - changer le type de la cellule en appuyant sur `m` pour passer en cellule Markdown ou sur `y` pour passer en cellule de code
  - insérer une cellule juste au-dessus en appuyant sur `a`
  - insérer une cellule juste en-dessous en appuyant sur `b`
  - couper la cellule en appuyant sur `x` etc ...
- en mode *édition* lorsqu'on clique sur l'intérieur de la cellule.

L'aide complète sur les raccourcis claviers est accessible depuis le bouton `Help` dans la barre d'outils ci-dessus.

1 Exercice 1

[1]:
```
def chiffres2nombre(t):
    """
    Renvoie  l'écriture en base 10 d'un entier à partir
    du tableau de ses chiffres par poids décroissant
    de gauche à droite

    Parameter:
        t : tableau d'entiers

    Return:
```

```
        int
    """
    # à compléter
    n = 0
    expomax = len(t) - 1
    for k in range(expomax + 1):
        n = n + 10 ** (expomax - k) * t[k]
    return n

# tests unitaires
assert chiffres2nombre([7,3,4]) == 734
assert chiffres2nombre([5,0,1]) == 501
assert chiffres2nombre([0]) == 0
assert chiffres2nombre([9]) == 9
```

[2]:
```
def horner(t):
    """
    Renvoie -> à compléter !

    Parameter:
        t : tableau d'entiers

    Return:
        int
    """
    nombre = 0
    #à compléter
    for chiffre in t:
        nombre = nombre * 10
        nombre = nombre + chiffre
    return nombre

# tests unitaires à compléter !
```

2 Exercice 2

[3]:
```
def liste_chiffre(n):
    """Renvoie la liste de chiffres en base 10 d'un entier n"""
    leschiffres = []
    while n >= 10:
        chiffre = n % 10
        leschiffres.append(chiffre)
        n = n // 10
    leschiffres.append(n)
    leschiffres.reverse()
    return leschiffres
```

```
# Jeu de tests unitaires
for k in range(100):
    #[int(c) for c in str(k)]
    #est un raccourci pour obtenir  la liste des chiffres
    assert liste_chiffre(k) == [int(c) for c in str(k)]
```

3 Exercice 3

[4]:
```
def bits2nombre(t):
    """
    Renvoie l'entier en base dix représenté
    par un tableau  de bits t avec bits de poids fort
    à gauche

    Parameter :
        t : tableau d'entiers (0 ou 1)

    Returns :
        int
    """
    n = 0
    # à compléter avec l'algorithme d'Horner
    for bit in t:
        n = n * 2 + bit
    return n


# Jeu de tests unitaires
assert bits2nombre([0]) == 0
assert bits2nombre([1]) == 1
assert bits2nombre([1, 0]) == 2
assert bits2nombre([1, 1]) == 3
assert bits2nombre([1, 0, 0]) == 4
assert bits2nombre([1, 0, 1]) ==5
assert bits2nombre([1, 1, 0]) == 6
assert bits2nombre([1, 1, 1]) == 7
```

4 Exercice 4

Vidéo sur la page http://video.math.cnrs.fr/magie-en-base-deux/

[9]:
```
from IPython.display import HTML

HTML('<iframe width="1920" height="1080" src="https://www.youtube.com/embed/
 ↪HsS7kaFDrXg" title="YouTube video player" frameborder="0"␣
 ↪allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope;␣
 ↪picture-in-picture" allowfullscreen></iframe>')
```

[9]:
```
<IPython.display.HTML object at 0xcd91c8>
```

[6]:
```
def codage_binaire_glouton(n):
    binaire = []
    puissance2 = 1
    while puissance2 <= n:
        puissance2 = puissance2 * 2
    puissance2 = puissance2 // 2
    #puissance2 contient alors la plus grande puissance de 2 <= n
    while puissance2 >= 2:
        if puissance2 <= n:
            binaire.append(1)
            n = n - puissance2
        else:
            binaire.append(0)
        puissance2 = puissance2 // 2
    binaire.append(n)
    return binaire

# Jeu de tests unitaires
assert codage_binaire_glouton(0) == [0]
assert codage_binaire_glouton(1) == [1]
assert codage_binaire_glouton(2) == [1, 0]
assert codage_binaire_glouton(3) == [1, 1]
assert codage_binaire_glouton(4) == [1, 0, 0]
assert codage_binaire_glouton(5) == [1, 0, 1]
```

[7]:
```
def codage_binaire2(n):
    """
    Renvoie le tableau de bits (poids fort à gauche)
    d'un entier n (en décimal)
    Algo des divisions en cascade

    Parameter :
        n : int

    Returns :
        tableau d'entiers
    """
    binaire = []
```

```
    while n >= 2:
        binaire.append(n % 2)
        n = n // 2
    binaire.append(n)
    binaire.reverse()
    return binaire

# Jeu de tests unitaires
assert codage_binaire2(0) == [0]
assert codage_binaire2(1) == [1]
assert codage_binaire2(2) == [1, 0]
assert codage_binaire2(3) == [1, 1]
assert codage_binaire2(4) == [1, 0, 0]
assert codage_binaire2(5) == [1, 0, 1]
```

5 Exercice 5

[8]:
```
def additionBinaire8bits(t1, t2):
    """
    Renvoie le tableau de bits (sur 8 bits)
    de l'addition binaire de deux entiers
    représentés par les tableaux de bits t1 et t2

    Parameter :
        t1, t2 : tableaux d'entiers (0 ou 1)

    Returns:
        tableau d'entiers (0 ou 1)
    """
    t3 = [0]  * 8
    retenue = 0
    for k in range(7, -1 , -1):
        s = t1[k] + t2[k] + retenue
        t3[k] =  s % 2
        retenue = s // 2
    if retenue != 0:
        raise OverflowError("Déplacement de capacité")
    return t3

# Jeu de tests unitaires
assert additionBinaire8bits([0,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,0]) ==␣
 ↪[0,0,0,0,0,0,0,0]
assert additionBinaire8bits([0,0,0,0,0,0,0,1], [0,0,0,0,0,0,0,0]) ==␣
 ↪[0,0,0,0,0,0,0,1]
assert additionBinaire8bits([0,0,0,0,0,0,0,1], [0,0,0,0,0,0,0,1]) ==␣
 ↪[0,0,0,0,0,0,1,0]
```

```
assert additionBinaire8bits([0,0,0,0,0,0,1,0], [0,0,0,0,0,0,1,0]) ==␣
↪[0,0,0,0,0,1,0,0]
assert additionBinaire8bits([1,0,1,0,1,1,1,0],[0,0,0,0,1,1,1,1]) == [1, 0, 1,␣
↪1, 1, 1, 0, 1]
print("Tests sans dépassement de capacité réussis")
# on va capturer l'exception
try:
    additionBinaire8bits([1,0,0,0,0,0,0,0],[1,0,0,0,0,0,0,0])
except OverflowError as e:
    print(e)
print("Test avec dépassement de capacité réussi")
```

```
Tests sans dépassement de capacité réussis
Déplacement de capacité
Test avec dépassement de capacité réussi
```

6 Exercice 6

6.1 Exemple de dépassement de capacité pour des entiers non signés sur 8 bits

[33]:
```
import numpy as np
np.uint8(255) + np.uint8(1)
```

[33]:
```
0
```

[11]:
```
from typing import List

def complement_deux(n:int, nbits:int)->List[int]:
    """
    Renvoie la notation en compléments à 2 de l'entier signé n
    sous la forme d'un tableau de bits ordonnés de gauche à droite
    par poids décroissant

    Parameters
    ----------
    n : int    Précondition  -2**(nbits-1) <= n < 2**(nbits-1)
    nbits : int

    Returns
    -------
    tableau de nbits bits
    """
    # Précondition
    assert -2**(nbits-1) <= n < 2**(nbits-1)
    if 0 <= n < 2**(nbits-1):
        binaire = codage_binaire2(n)
        return [0] * (nbits - len(binaire)) + codage_binaire2(n)
```

```
    else:
        return codage_binaire2(n + 2 ** nbits)


# Jeu de tests unitaires
assert complement_deux(0, 8) == [0, 0, 0, 0, 0, 0, 0, 0]
assert complement_deux(5, 8) == [0, 0, 0, 0, 0, 1, 0, 1]
assert complement_deux(-2, 8) == [1, 1, 1, 1, 1, 1, 1, 0]
assert complement_deux(2**7 - 1, 8) ==  [0, 1, 1, 1, 1, 1, 1, 1]
assert complement_deux(-2**7, 8) == [1, 0, 0, 0, 0, 0, 0, 0]
assert complement_deux(2**7 - 2, 8) == [0, 1, 1, 1, 1, 1, 1, 0]
assert complement_deux(-2**7 + 1, 8) == [1, 0, 0, 0, 0, 0, 0, 1]
assert complement_deux(-1, 8) == [1, 1, 1, 1, 1, 1, 1, 1]
assert complement_deux(-2, 8) == [1, 1, 1, 1, 1, 1, 1, 0]
```