

Les entiers et leurs représentations

Chap. 14,02

Préliminaire. Lors de cette séance toutes les fonctions à programmer sont données avec leur spécification. Cette dernière comporte des tests et leurs résultats. Je vous rappelle que pour exécuter automatiquement ces tests les dernière lignes du fichier contenant votre code doivent se conclure par :

```
if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

1 Les entiers naturels

1.1 La représentation binaire

1) Définir la fonction `octet` dont la spécification est :

```
def octet(dec: str) -> str:
    """
    Vérifie si la chaîne contenant la représentation d'un
    nombre entier n < 256 en base 2 est agencée sous forme
    d'un octet.
    Ajoute des 0 si nécessaire.

    >>> octet('1101')
    '00001101'
    >>> octet('11010000')
    '11010000'
    """
```

► Solution :

```
def octet(dec: str) -> str:
    """
    Vérifie si la chaîne contenant la représentation d'un nombre entier n < 256
    en base 2 est agencée sous forme d'un octet.
    Ajoute des 0 si nécessaire.

    >>> octet('1101')
    '00001101'
    >>> octet('11010000')
    '11010000'
    """
    lg_octet = 8
    manque = lg_octet - len(dec)

    if manque > 0:
```

```

    for i in range(manque):
        dec = str(0) + dec

    return dec

```

2) Définir la fonction `decompose` dont la spécification est :

```

def decompose(n: int) -> str:
    """
    Représentation du nombre n dans la base 2. La grandeur
    retournée est une chaîne de caractères représentant le
    résultat de la décomposition, les valeurs de poids le
    plus élevé en premier sous forme d'un octet.

    HYPOTHÈSE : n < 256
    REMARQUE : decompose utilise octet

    >>> decompose(13)
    '00001101'
    >>> decompose(0)
    '00000000'
    """

```

3) Définir la fonction `recompose` dont la spécification est :

```

def recompose(dec: str) -> int:
    """
    Détermine la valeur du nombre représenté par dec dans la base
    2. La chaîne doit être organisée de telle sorte que les
    valeurs de poids le plus élevé apparaissent en premier.

    HYPOTHÈSE : la chaîne est binaire et est codée sur un octet.

    >>> recompose('11010100')
    212
    >>> recompose('00000000')
    0
    """

```

1.2 La représentation hexadécimale

4) Définir la fonction `decompose_16` dont la spécification (et la première ligne de code) est :

```

def decompose_16(n: int) -> List[int]:
    """
    Représentation du nombre n dans la base 16. La grandeur retournée est
    une chaîne représentant le résultat de la décomposition, les valeurs de poids
    le plus élevé en premier.

    REMARQUE : La fonction appelle les fonctions decompose et octet.
    """

```

```
>>> decompose_16(13)
'0xD'
>>> decompose_16(0)
'0x0'
>>> decompose_16(347)
'0x15B'
"""
ref = {10: 'A', 11: 'B', 12: 'C', 13: 'D', 14: 'E', 15: 'F'}
```

5) Définir la fonction `recompose_16` dont la spécification est :

```
def recompose_16(dec: str) -> int:
    """
    Détermine la valeur du nombre représenté par dec dans la base 16.
    La chaîne doit être organisée de telle sorte que les valeurs de poids le
    plus élevé apparaissent en premier.

    >>> recompose_16('0x15')
    21
    >>> recompose_16('0x15B')
    347
    >>> recompose_16('0xF')
    15
    """
    ref = {'A': 10, 'B': 11, 'C': 12, 'D': 13, 'E': 14, 'F': 15}
```

6) Définir la fonction `de_bin_vers_hex` dont la spécification est :

```
def de_bin_vers_hex(dec: str) -> str:
    """
    À partir de dec, la représentation binaire d'un nombre entier, détermination
    de sa représentation hexadécimale en regroupant les bits par 4.

    HYPOTHÈSE : la chaîne binaire est un octet.
    REMARQUE : de_bin_vers_hex doit utiliser recompose et decompose_16.

    >>> de_bin_vers_hex('00001101')
    '0D'
    >>> de_bin_vers_hex('11111111')
    'FF'
    """
```

7) Définir la fonction `de_hex_vers_bin` dont la spécification est :

```
def de_hex_vers_bin(dec: str) -> str:
    """
    À partir de dec, la représentation hexadécimale d'un nombre entier, détermination
    de sa représentation binaire.
```

HYPOTHÈSE : la chaîne binaire est un octet.

REMARQUE : de_bin_vers_hex doit utiliser recompose et decompose_16.

```
>>> de_hex_vers_bin('0D')
'00001101'
>>> de_hex_vers_bin('FF')
'11111111'
"""
```

2 Les entiers relatifs

8) Définir la fonction `decompose_relatif` dont la spécification est :

```
def decompose_relatif(n: int) -> str:
    """
    Retourne une chaîne de caractères qui donne la représentation d'un entier relatif
    en base 2 en utilisant le complément à 2.

    HYPOTHÈSE : n peut être codé sur un octet.
    REMARQUE : decompose_relatif utilise decompose.

    >>> decompose_relatif(127)
    '01111111'
    >>> decompose_relatif(0)
    '00000000'
    >>> decompose_relatif(-1)
    '11111111'
    >>> decompose_relatif(-128)
    '10000000'
    """
```

9) Définir la fonction `recompose_relatif` dont la spécification est :

```
def recompose_relatif(dec: str) -> int:
    """
    Détermine la valeur de l'entier relatif représenté par dec dans la base 2
    lorsqu'on utilise le complément à 2.
    La chaîne doit être organisée de telle sorte que les valeurs de poids le
    plus élevé apparaissent en premier.

    HYPOTHÈSE : la chaîne est binaire et est codée sur un octet.
    REMARQUE : recompose_relatif utilise recompose.

    >>> recompose_relatif('10000000')
    -128
    >>> recompose_relatif('01111111')
    127
    >>> recompose_relatif('00000000')
    0
    """
```

3 Quelques opérations

10) Définir la fonction `addition` dont la spécification est :

```
def addition(c_1: str, c_2: str) -> str:
    """
    Retourne le résultat de l'addition des deux nombres dont les représentations
    binaires sont c_1 et c_2.

    HYPOTHÈSE : c_1 et c_2 sont des octets
    REMARQUE : Une chaîne vide est retournée si la somme des deux nombres ne peut
    pas être représentée avec un octet.

    >>> addition('00000000', '00000001')
    '00000001'
    >>> addition('00000001', '00000001')
    '00000010'
    >>> addition('00000001', '00000011')
    '00000100'
    >>> addition('11111111', '00000001')
    ''
    """
```

11) Cette définition de l'addition est-elle valable avec des entiers relatifs ?

12) Définir la fonction `decalage_vers_gauche` dont la spécification est :

```
def decalage_vers_gauche(dec: str, n: int) -> str:
    """
    Décale tous les bits de n rangs vers la gauche dans dec, représentation d'un
    entier. Des 0 sont ajoutés aux places créées sur la partie droite.

    HYPOTHÈSE : dec est un octet (ce qui n'est pas forcément le cas de la réponse).

    >>> decalage_vers_gauche('00000001', 1)
    '00000010'
    >>> decalage_vers_gauche('00000011', 1)
    '00000110'
    >>> decalage_vers_gauche('00000011', 3)
    '0000011000'
    """
```

13) Quelle opération mathématique réalise la fonction `decalage_vers_gauche` sur le nombre passé en argument ?

14) Définir la fonction `decalage_vers_droite` dont la spécification est :

```
def decalage_vers_droite(dec: str, n: int) -> str:
    """
```

Décale tous les bits de `n` rangs vers la droite dans `dec`, représentation d'un entier. Des 0 sont ajoutés aux places créées sur la partie gauche.

HYPOTHÈSE : `dec` et la réponse sont des octets.

REMARQUE : `decalage_vers_droite` doit utiliser octet.

```
>>> decalage_vers_droite('00000011', 1)
'00000001'
>>> decalage_vers_droite('10000011', 2)
'00100000'
"""
```

- 15) Quelle opération mathématique réalise la fonction `decalage_vers_droite` sur le nombre passé en argument ?