Portail Descartes
Introduction à la science info : TP 4
Codage et chaînes de caractères

amU Faculté des sciences Aix Marseille Université

Exercice 1 Dans cet exercice, on propose d'écrire des fonctions Python permettant de convertir un entier (stocké avec le type `int` en Python) en des chaînes de caractères contenant sa représentation binaire ou décimale, et vice versa. Par exemple, l'entier 19 est associé à sa représentation binaire `'10011'` et à sa représentation décimale `'19'` (à ne pas confondre avec l'entier lui-même : il s'agit ici d'une chaîne de caractères de longueur 2). **Dans tout cet exercice, on s'interdit l'utilisation de la fonction `str`.**

1. Écrire une fonction `convert_bin_to_int` prenant en paramètre une chaîne de caractères `'0'` et `'1'` et renvoyant l'entier dont la chaîne est la représentation binaire. Par exemple, `convert_bin_to_int('100101001')` renvoie l'entier 297.
2. Écrire une fonction `convert_int_to_bin` prenant en paramètre un entier et renvoyant la chaîne de caractères contenant sa représentation binaire. Par exemple, `convert_int_to_bin(297)` renvoie la chaîne `'100101001'`.
3. Écrire une fonction `convert_int_to_dec` prenant en paramètre un entier et renvoyant la chaîne de caractères contenant sa représentation décimale. Par exemple, `convert_int_to_dec(297)` renvoie la chaîne `'297'`.
4. En utilisant les fonctions précédentes, écrire une fonction `convert_bin_to_dec` prenant en paramètre une chaîne de caractères `'0'` et `'1'` et renvoyant la chaîne de caractères contenant la représentation décimale de l'entier correspondant. Par exemple, `convert_bin_to_dec('100101001')` renvoie la chaîne `'297'`.

Exercice 2 Historiquement, un procédé de cryptographie bien connu est le codage de César que Jules César utilisait dans ses correspondances. Le principe de chiffrement est simple. Étant donné un alphabet (ici, nous utiliserons l'alphabet latin non accentué et en minuscule, comprenant 26 caractères) et un message, le message chiffré s'obtient en remplaçant chacune des lettres du message d'origine par une lettre à distance fixe toujours dans la même direction. Pour les dernières lettres, dans le cas d'une distance à droite, on reprend au début de l'alphabet. Il s'agit d'un chiffrement par décalage. La valeur de décalage s'appelle la *clé de chiffrement*. À titre d'exemple, avec une clé de 5, `'a'` devient `'f'`, `'b'` devient `'g'`, ..., `'y'` devient `'d'` et `'z'` devient `'e'`. Dans l'exercice, on supposera toujours que la clé est un entier entre -25 et 25.

1. Soit le message `'pays'`. Donner sa représentation chiffrée selon le codage de César avec la clé 3. Faites de même avec la clé -3.
2. Exécuter le code suivant et en déduire le rôle des fonctions `ord` et `chr` de Python.

```
print(ord('a'))
print(ord('b'))
print(ord('y'))
print(ord('z'))
print(chr(97))
print(chr(122))
```

3. En Python, on peut stocker le message en clair, ainsi que le message chiffré, à l'aide de chaînes de caractères. Écrire une fonction `cipher_Cesar` prenant en paramètres une chaîne de caractères contenant un message en clair et une clé de chiffrement, et renvoyant la chaîne de caractères obtenue à l'aide du chiffrement de César. *Il s'agit de créer une chaîne de caractères initialement vide qui contiendra le message chiffré, parcourir le message en clair et pour chacun des caractères, produire le caractère chiffré et l'ajouter à la chaîne de caractères contenant le chiffré. Pour produire le caractère chiffré, on utilisera les fonctions* `ord` *et* `chr`.
4. L'algorithme précédent ne prend pas en charge les espaces auxquels on ne souhaite pas apporter de décalage. Ainsi, on aimerait que le message en clair 'un deux' soit chiffré, avec la clé 3, en `'xq ghxa'`. Modifier l'algorithme précédent pour y parvenir.
5. Proposer un algorithme de déchiffrement, prenant en entrée le message chiffré et une clé et renvoyant le message décodé. *Si votre algorithme de chiffrement est bien écrit, vous devriez pouvoir utiliser un appel à cette fonction !*
6. Utiliser votre fonction pour décrypter (c'est-à-dire qu'on ne connaît pas la clé !) le message chiffré

```
'vxvb xlm ng fxlltzx w texkmx gx itl wbyynlxk xm wxmknbkx tikxl wxvabyykxfxgm'
```

Exercice 3 Le codage ASCII permet de représenter 128 caractères (dont des caractères de contrôle qui ne sont pas des lettres ou symboles usuels) avec des chaînes de 7 bits. On peut la stocker en Python avec le code

```
ascii = [chr(i) for i in range(128)]
```

1. Écrire une fonction `decode_ascii` qui prend en paramètres une chaîne de caractères `'0'` et `'1'`, et qui renvoie la chaîne de caractères qui est ainsi représentée en ASCII. *On supposera que la chaîne en paramètres est bien un code ASCII, en particulier qu'elle est de longueur multiple de* 7*. On pourra utiliser la fonction* `convert_bin_to_int` *écrite dans l'exercice 1.*
2. Utiliser votre fonction pour décoder en ASCII la séquence de bits

   ```
   '10100111100011110100111001011101110110001111001010100000100100111011101100
   11011011111110010110110111000011110100110100111100011110101110010 1'
   ```

Exercice 4

1. Écrire une fonction `convert_bin_to_hexa` prenant en paramètre une chaîne de caractères `'0'` et `'1'` et renvoyant la chaîne de caractères contenant la représentation hexadécimale de l'entier correspondant. Par exemple, `convert_bin_to_hexa('10101000')` renvoie la chaîne `'A8'`. *On s'interdit le passage par le type* `int` *et on demande donc un encodage qui passe directement de la base* 2 *à la base* 16*.*
2. Utiliser votre fonction sur le paramètre `'11001010111111100010000000100100'` puis sur le paramètre `'1000010101010111100'`.