Séance 4 : Codages et représentations

L1 – Université Côte d'Azur

Exercice 1 — Représentations binaires

1. Traduire 116 en binaire. On utilisera l'algorithme vu dans le premier cours.
2. En déduire la répresentation de -116 sur 8 bits.
3. On considère le nombre `110 0001` sur 8 bits. Quel est son signe ? Combien vaut-il ? On appliquera là encore l'algorithme vu en cours.
4. Même question avec `1001 1111`

Exercice 2 — Représentations en bases quelconques

1. Qu'est-ce le code Ascii ? Unicode ? et l'encodage UTF-8 ?
2. Quel est le numéro de la lettre `"r"` en Ascii.
3. Comment afficher en une boucle `for` le code des chiffres de `'0'` à `'9'` ?
4. Écrivez une fonction `liste_chiffres()` qui renvoie la chaîne contenant tous les "chiffres" `'0…9A…Z'`.
5. Pour la suite de l'exercice, on pose `chiffres = liste_chiffres()`. Que vaut `chiffres[0]` ? `chiffres[9]` ? `chiffres[10]` ? Ces chiffres-là permettent de représenter les nombres jusqu'à quelle base ? Par la suite toutes les bases considérées seront inférieures à cette valeur.
6. Écrivez une fonction `écrire_nombre(n, b)` qui renvoie la représentation d'un nombre `n` dans une base `b`.

```
>>> écrire_nombre(17, 2)  # 1×2^4 + 1
'10001'
>>> écrire_nombre(17, 10) # 1×10 + 7
'17'
>>> écrire_nombre(17, 16) # 1×16 + 1
'11'
```

7. Écrivez une fonction `valeur(ch)` qui renvoie l'indice du caractère `ch` dans la chaîne `chiffres`. Si `ch` n'est pas présent dans la chaîne, on renverra `None`.

```
>>> valeur('1')
1
>>> valeur('7')
7
>>> valeur('A')
10
>>> valeur('G')
16
```

8. Déduisez-en une fonction `lire_nombre(chaîne, b)` qui à partir d'une représentation en base `b` renvoit la valeur numérique associée.

```
>>> lire_nombre("10001", 2)
17
>>> lire_nombre("17", 10)
17
>>> lire_nombre("11", 16)
17
```

Exercice 3 — Couleur RGB

En `html` les couleurs peuvent être définies en héxadécimal, par exemple `'#FF12E4'` où `FF`, `12` et `E4` correspondent aux trois composantes de rouge, de vert et de bleu. Écrire une fonction `couleur_rgb(r,g,b)` qui prend trois entiers en paramètres compris entre 0 et 255 et qui **renvoie** la chaîne de caractère correspondante de la forme `'#RRVVBB'` ou chaque composante est codée sur un nombre à deux chiffres hexadécimaux.

Exercice 4 — Détecter l'Ascii

1. Écrivez la fonction `décoder(chaîne)` qui renvoie le tableau des numéros Ascii de chaque caractère de `chaîne`.

```
>>> décoder("bonjour camarade !")
[98, 111, 110, 106, 111, 117, 114, 32, 99, 97, 109, 97, 114, 97, 100, 101, 32, 33]
```

```
def décoder(chaîne):
    L = [0] * len(chaîne)
    for i in range(len(chaîne)):
        L[i] = ord(chaîne[i])
    return L
```

2. Déduisez-en une fonction `est_ascii(ch)` qui indique si la chaîne ne contient que des caratères Ascii.

```
>>> est_ascii("bonjour camarade !")
True
>>> est_ascii("ΞINO ΛAXANO") # Quelle est la dernière lettre de la chaîne ?
False
```

```
def est_ascii(chaîne):
    L = décoder(chaîne)
    for i in range(len(chaîne)):
        if L[i] > 127:
            return False
    return True
```

Pour info, la lettre est un omicron majuscule grec !

3. Écrivez une fonction `recoder(L)` qui renvoie la chaîne Ascii correspondant à la liste `L`. Tous les caractères non-Ascii seront remplacés par `"?"`.

```
>>> français = décoder("bonjour camarade génial !")
>>> grec = décoder("ΞINO ΛAXANO")
>>> recoder(français)
'bonjour camarade g?nial !'
>>> recoder(grec)
'???? ??????'
```

```
def recoder(liste):
    c = ""
    for i in range(len(liste)):
        if liste[i] < 128:
            c = c + chr(liste[i])
        else:
            c = c + "?"
    return c
```

Exercice 5 — Encoder le volume sonore

1. Dans le code Ascii, comment passer en un seul calcul d'une lettre minuscule à une lettre majuscule ?
2. Écrivez la fonction `HURLER(chaîne)` qui renvoie la chaîne donnée en argument où chaque minuscule est remplacée par une majuscule.

```
>>> HURLER("Parle plus fort j'entends rien !")
'PARLE PLUS FORT J'ENTENDS RIEN !'
```

```
def HURLER(chaîne):
    r = ""
    for i in range(len(chaîne)):
        d = ord(chaîne[i])
        if ord('a') <= d and d <= ord('z'):
            d = d - ord('a') + ord('A')
        r = r + chr(d)
    return r
```

3. De même, écrivez la fonction `murmurer(chaîne)` qui renvoie la chaîne tout en minuscule.

```
>>> murmurer("CHUT… Le Prof nous regarde…")
'chut… le prof nous regarde…'
```

```
def murmurer(chaîne):
    r = ""
    for i in range(len(chaîne)):
        d = ord(chaîne[i])
        if ord('A') <= d and d <= ord('Z'):
            d = d - ord('A') + ord('a')
        r = r + chr(d)
    return r
```

4. Pourquoi vos fonctions sont fausses ? En Python, comment faudrait-il faire pour gérer proprement les majuscules et minuscules.

   Notre fonction ne gère que l'ASCII et non les caractères accentués. En vrai, il faut utiliser les méthodes `"été".upper()` et `"ÉTÉ".lower()`

5. Écrivez une fonction `title(chaîne)` qui met la première lettre de chaque mot de la phrase en capitale (comme c'est l'usage dans les titres en anglais).

```
>>> title("the beauty and the beast")
'The Beauty And The Beast'
>>> title("the beauty and spider-man (first spin-off)")
'The Beauty And Spider-Man (First Spin-Off)'
```

```
def title(ch):
    ancien = False # Le caractère précédent est-il une lettre ?
    res = ""
    for i in range(len(ch)):
        if not ancien: # Si je suis en début de mot
            res = res + ch[i].upper()
        else:
            res = res + ch[i]
        ancien = est_lettre(ch[i])
    return res
```

Exercice 6 — Interpréter une chaîne de caractères

On se donne une chaîne de caractères représentant une liste de nombres. Écrivez une fonction `chaîne_vers_liste(ch)` qui renvoie la liste associée.

```
>>> chaîne_vers_liste("[12,23,  24, 2  , 1]")
[12, 23, 24, 2, 1]
```

```python
def est_chiffre(c):
    return ord("0") <= ord(c) and ord(c) <= ord("9")

def nettoyer(ch):
    """On ne conserve que les chiffres et les virgules"""
    rés = ""
    for i in range(len(ch)):
        if est_chiffre(ch[i]) or ch[i] == ",":
            rés += ch[i]
    return rés

def nombre_de_virgules(ch):
    n = 0
    for i in range(len(ch)):
        if ch[i] == ",":
            n = n+1
    return n


def chaîne_vers_liste(ch):
    ch = nettoyer(ch)
    # On gère à part le cas de la liste vide
    if ch== "":
        return []

    # Dans le tableau il y a une case de plus que de virgules
    tab = [0] * (1+nombre_de_virgules(ch))
    # k est l'indice courant du tableau
    k = 0
    # courant est la chaîne contenant les caractères du prochain nombre
    courant = ""

    for i in range(len(ch)):
        if ch[i] == ",":
            tab[k] = lire_nombre(courant, 10) # cf. exercice 2
            courant = ""
            k = k+1
        else:
            courant = courant + ch[i]
    tab[k] = lire_nombre(courant, 10) # cf. exercice 2
    return tab

```