DST Informatique : Gestion de Tests dans une entreprise

Une entreprise d'e-commerce vend des meubles tous identifiés par une référence et par un QR code [1]. Tous les clients sont identifiés par leur numéro de sécurité sociale. Tous les achats s'effectuent à l'aide d'un numéro de carte de crédit. Cette entreprise met en œuvre différents tests afin d'éviter les erreurs de numéros de sécurité sociale, de numéros de carte de crédit ou de QR codes.

Partie I - Tests de code de sécurité sociale

En France, le numéro de sécurité sociale correspond au numéro d'inscription au répertoire national d'identification des personnes physiques (RNIPP).
Il est formé du numéro d'inscription (NIR) à 13 chiffres et d'une clé de contrôle à 2 chiffres.
Le NIR, créé à partir de l'état civil, est composé de la façon suivante :

- Sexe (1er chiffre) ;
- Année de naissance (les deux chiffres suivants) ;
- Mois de naissance (les deux chiffres suivants) ;
- Lieu de naissance (les cinq chiffres ou caractères suivants - 2 chiffres [2] du code du département de naissance, suivis de 3 chiffres du code commune officiel de l'Insee [3]) ;
- Numéro d'ordre permettant de distinguer les personnes nées au même lieu à la même période (les 3 chiffres suivants).

La clé de contrôle est composée de deux chiffres allant de 01 à 97 et permet de contrôler l'exactitude du numéro de sécurité sociale.

Pour obtenir cette clé, on détermine tout d'abord, le reste de la division par 97 du nombre formé par les 13 premiers chiffres. La clé correspond au résultat de ce nombre retranché de 97.

Exemple : soit le numéro de sécurité sociale à 13 chiffres : "2 91 01 75 018 002".
Le reste de la division de 2910175018002 par 97 est égal à 29.
La clé est constituée du résultat : $97 - 29 = 68$.
Le numéro de sécurité sociale complet est donc : "2 91 01 75 018 002 68".

Dans cette partie, le numéro de sécurité sociale de 13 chiffres est une chaîne de caractères composée uniquement de chiffres avec des espaces de séparation entre les différents éléments constituant ce numéro. On ne prendra pas en compte le cas de la Corse.

Des rappels sur les chaînes de caractères sont donnés en Annexe.

Question 1
Écrire la fonction `secu_en_liste` qui a pour paramètre une chaîne de caractères `secu` correspondant au numéro de sécurité sociale à 13 chiffres, et qui renvoie le numéro sous forme d'une liste d'entiers.
Le programme devra parcourir la chaîne de caractères représentant le numéro de sécurité sociale en supprimant les caractères d'espacement, puis transformer chaque caractère en entier et stocker chaque entier dans une liste.
Exemple :

```
>>> secu_en_liste("2 91 01 75 018 002")
[2, 9, 1, 0, 1, 7, 5, 0, 1, 8, 0, 0, 2]
```

1. Un QR code (Quick Response code) désigne un type de code-barres en deux dimensions, lequel se compose de modules noirs disposés dans un carré à fond blanc (voir figure 1).
2. Pour simplifier le problème, nous supposons que les deux départements corses 2A et 2B sont représentés par le code 20 comme avant 1976.
3. Institut national de la statistique et des études économiques.

Solution

```
def secu_en_liste(secu):
    lst = []
    for ch in secu:
        if ch != ' ':
            lst.append(int(ch))
    return lst
```

Question 2

Ecrire une fonction `liste_en_int` qui a pour paramètre une liste d'entiers `lst` et qui renvoie l'entier correspondant à la liste.

Exemple :

```
>>> liste_en_int([2, 9, 1, 0, 1, 7, 5, 0, 1, 8, 0, 0, 2]
2910175018002
```

Solution

```
def liste_en_int(lst):
    nb = 0
    n = len(lst)
    for k in range(len(lst)):
        nb = nb + lst[n-1-k] * 10**k
    return nb
```

Question 3

En utilisant les deux fonctions précédentes, écrire une fonction `num_secu` qui a pour paramètre une chaîne de caractères `secu` correspondant au numéro de sécurité sociale à 13 chiffres, et qui renvoie l'entier correspondant au numéro.

Exemple :

```
>>> num_secu("2 91 01 75 018 002")
2910175018002
```

Solution

```
def num_secu(secu):
    return liste_en_int(secu_en_liste(secu))
```

Question 4

Ecrire une fonction `clef` qui a pour paramètre un entier `num` corresondant au numéro de sécurité sociale à 13 chiffres et qui renvoie la clé de `num`

Exemple :

```
>>> clef(2910175018002)
68
```

Solution

```
def clef(num):
    return 97 - num % 97
```

Question 5
Ecrire une fonction `num_secu_complet` qui a pour paramètre une chaîne de caractères `secu` correspondant au numéro de sécurité sociale à 13 chiffres et qui renvoie la chaîne de caractères correspondant au numéro de sécurité sociale complet.
Exemple :

```
>>> num_secu_complet("2 91 01 75 018 002")
"2 91 01 75 018 002 68"
```

Solution

```
def num_secu_complet(secu):
    num = num_secu(secu)
    cle = clef(num)
    return secu + " " + str(cle)
```

Question 6
Ecrire une fonction `test_num_secu` qui a pour paramètre une chaîne de caractères `secu` correspondant au numéro de sécurité sociale complet et qui renvoie `True` si le numéro de sécurité sociale et valide et `False` sinon.
Exemple :

```
>>> test_num_secu("2 91 01 75 018 002 68")
True
>>> test_num_secu("2 91 01 75 018 002 93")
False
```

Solution

```
def test_num_secu(secu):
    secu_13 = secu[:-3]
    return num_secu_complet(secu_13) == secu
```

Partie II - Test de numéro de carte de crédit

Pour savoir si un numéro de carte de crédit est valide, on utilise très souvent l'algorithme de Luhn [4]. Comme pour le numéro de sécurité sociale, il y a une clé appelée **somme de contrôle** *(checksum en anglais)* qui fait partie du numéro d'une carte de crédit. Ce numéro est un entier composé de 16 chiffres. Le dernier chiffre est la clé qui permet de contrôler l'exactitude du numéro.
Le principe de l'algorithme de Luhn est le suivant. On commence toujours par le chiffre se trouvant le plus à droite. Ce chiffre sera le premier élément de la liste dites des *« indices impairs »*. Puis on complète cette liste en prenant un chiffre sur deux du numéro de carte bancaire, toujours en le lisant de la droite vers la gauche.

Pour la liste des chiffres *« d'indices pairs »*, on commence par le deuxième chiffre le plus à droite du numéro de la carte de crédit, on se déplace de la droite vers la gauche comme pour la liste précédente et on construit la liste, en prenant un chiffre sur deux. De plus, pour les nombres de cette liste des *indices pairs*, on double tous les chiffres. Si un nombre est supérieur à 9, on réalise la somme des deux chiffres qui le composent *(par exemple si on obtient 16, on additionne 1 et 6 pour avoir 7).*

4. L'algorithme de Luhn, ou code de Luhn, ou encore formule de Luhn est aussi connu comme l'algorithme "modulo 10".

Par conséquent, tous les nombres des deux listes sont composés uniquement de chiffres compris entre 0 et 9. On calcule alors la somme totale des chiffres de ces deux listes. Si cette somme est un multiple de 10, alors le numéro de la carte de crédit est valide.

Exemple : soit 4762 un nombre *(on se limite à 4 chiffres mais le raisonnement est identique pour un nombre à 16 chiffres)*. Appliquons-lui la formule de Luhn :
Le nombre 4762 se transforme en deux listes correspondant aux *indices impairs* et *pairs* soit [2, 7] et [6, 4]. Puis en deux autres listes, la liste des *indices impairs* inchangés [2, 7] et la liste des *indices pairs* doublés [12, 8]. La liste des *indices pairs* [12, 8] se réduit en la liste [3, 8] du fait de la sommation des chiffres constituant le nombre 12. La somme des éléments des deux listes obtenues [2, 7] et [3, 8] est bien égale à 20. Ce résultat est un multiple de 10, le nombre 4762 est donc correct au sens de l'algorithme de Luhn.

On aurait pu raisonner sur une seule liste et obtenir le même résultat. 4762 se transforme en la liste [8, 7, 12, 2] puis en la liste [8, 7, 3, 2] après réduction. La somme des chiffres 8+7+3+2 est égale à 20.

Question 7
Écrire une fonction `num_en_liste` qui a pour paramètre un entier `num` qui transforme un nombre entier en une liste de chiffres. *On pourra commencer par convertir l'entier en une chaîne de cracatères.*
Exemple :

```
>>> num_en_liste(4532015112830465)
[4, 5, 3, 2, 0, 1, 5, 1, 1, 2, 8, 3, 0, 4, 6, 5]
```

Solution

```
def num_en_liste(num):
    lst = []
    s = str(num)
    for elt in s:
        lst.append(int(elt))
    return lst
```

Question 8
Écrire une fonction `impairs` qui a pour paramètre un entier `num` correspondant au code de carte de crédit et qui renvoie la liste des chiffres *d'indices impairs*.
Exemple :

```
>>> impairs(4532015112830465)
 [5, 4, 3, 2, 1, 1, 2, 5]
```

Solution

```
def impairs(num):
    lst = []
    s = str(num)
    k = len(s) -1
    while k >= 0:
        lst.append(int(s[k]))
        k = k - 2
    return lst
```

Question 9

Écrire une fonction `pairs` qui a pour paramètre un entier `num` correspondant au code de carte de crédit et qui renvoie la liste des chiffres *d'indices pairs*.
Exemple :

```
>>> pairs(4532015112830465)
 [6, 0, 8, 1, 5, 0, 3, 4]
```

Solution

```
def pairs(num):
    lst = []
    s = str(num)
    k = len(s) -2
    while k >= 0:
        lst.append(int(s[k]))
        k = k - 2
    return lst
```

Question 10
Écrire une fonction `traitement_nb_pairs` qui a pour paramètre une liste `pair` d'entiers correspondant à la liste des chiffres d'indices pairs et qui, d'une part multiplie par 2 tous les chiffres de la liste, d'autre part effectue la transformation si un chiffre est supérieur à 9.
Exemple :

```
>>> traitement_nb_pairs([6, 0, 8, 1, 5, 0, 3, 4])
 [3, 0, 7, 2, 1, 0, 6, 8]
```

Solution

```
def traitement_nb_pairs(pair):
    lst = []
    for elt in pair:
        a = elt * 2
        if a > 9:
            a = a // 10 + a % 10
        lst.append(a)
    return lst
```

Question 11
Écrire une fonction `test_num_carte_credit` qui a pour paramètre un entier `num` correspondant au numéro d'une carte de crédit et qui renvoie `True` si le numéro est valide et `False` sinon.
Exemple :

```
>>> test_num_carte_credit(4532015112830465)
True
```

Solution

```
def test_num_carte_credit(num):
    lst_impairs = impairs(num)
    lst_pairs = traitement_nb_pairs(pairs(num))
    s = 0
    for elt in lst_pairs:
```

```
        s += elt
    for elt in lst_impairs:
        s += elt
    return s % 10 == 0
```

Partie III - Tests de QR code

Les QR codes ont été inventés en 1994, par Masahiro Hara, un ingénieur de l'entreprise japonaise Denso-Wave. Cette invention a permis d'assurer le référencement des pièces détachées dans les usines Toyota. Les QR codes sont constitués essentiellement de pixels noirs et blancs. Cependant, il existe des QR codes bicolores mais avec un jeu de couleurs très contrastées.
Les QR codes peuvent être partiellement raturés ou déchirés car un de leurs avantages est qu'ils peuvent accepter un certain taux d'erreurs, entre 7 % et 30 % suivant la version du QR code.
Il existe 40 versions qui peuvent stocker entre 10 et 7089 caractères numériques.

Nous nous restreignons ici à la version 1 du QR code.

FIGURE 1 – QR code version 1

Dans cette version, le QR code est représenté par une matrice de 21 × 21 pixels. Chaque pixel prend la valeur 1 lorsqu'il est blanc et 0 lorsqu'il est noir.
Le QR code est constitué de différents éléments : des motifs de positionnement (3 blocs de 7×7 pixels), des motifs de synchronisation (6 zones blanches de séparation et 11 pixels de couleur blanc et noir), des motifs de format d'information et une zone comprenant les données utilisateurs avec des motifs de correction (figure 2).

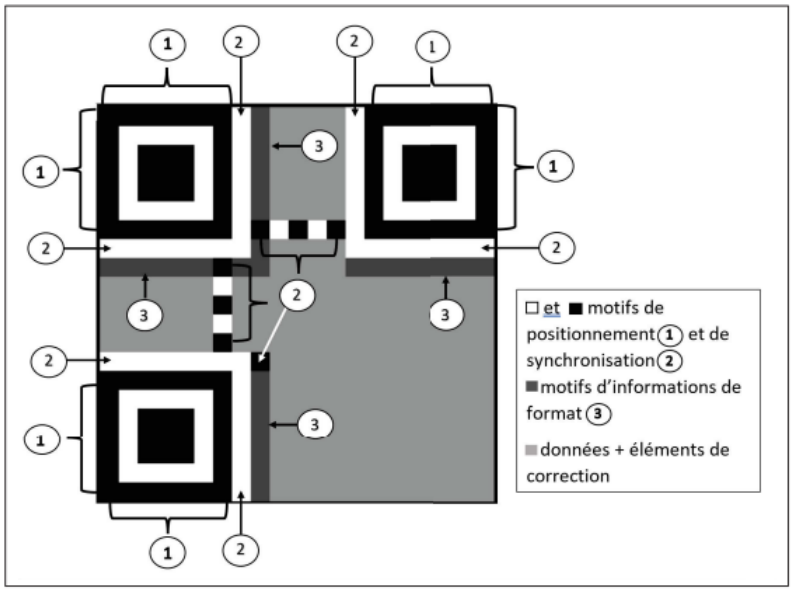

FIGURE 2 – Organisation d'un QR code

Question 12
Écrire une fonction `init` qui a pour paramètre un entier strictement positif `n` et qui renvoie une liste de longueur n où chaque élément est également une liste de longueur n constituée uniquement de 0. Ainsi cette liste de listes représente une matrice de taille n × n .
Exemple :

```
>>> init(4)
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

Solution

```
def init(n):
    lst = []
    for k in range(n):
        lst_n = []
        for i in range(n):
            lst_n.append(0)
        lst.append(lst_n)
    return lst
```

On prend comme bloc de positionnement celui représenté dans la figure 3.

FIGURE 3 – Bloc de positionnement

Question 13
Ecrire une fonction `cree_bloc` qui a pour paramètre un entier `n` (supérieur ou égal à 4) et qui renvoie une liste de listes n × n correspondant au bloc de positionnement de la figure 3.
Exemple :

```
>>> cree_bloc(7)
[[0, 0, 0, 0, 0, 0, 0],
[0, 1, 1, 1, 1, 1, 0],
[0, 1, 0, 0, 0, 1, 0],
[0, 1, 0, 0, 0, 1, 0],
[0, 1, 0, 0, 0, 1, 0],
[0, 1, 1, 1, 1, 1, 0],
[0, 0, 0, 0, 0, 0, 0]]
```

Solution

```
def cree_bloc(n):
    b = init(n)
    for k in range(1, n-1):
        b[1][k] = 1
        b[k][1] = 1
        b[n-2][k] = 1
        b[k][n-2] = 1
```

```
    return b
```

Question 14

Ecrire une fonction `test_bloc` qui teste si un bloc de positionnement est bien représenté pixel par pixel dans un QR code.

Cette fonction a pour paramètres 2 entiers `x` et `y` correspondant aux coordonnées de la position du début du bloc de positionnement d'un QR code (ce sont les coordonnées du pixel le plus haut à gauche), et `qr_code` la liste de liste de dimension 21 × 21 associée au QR code.

Cette fonction renvoie `True` si le bloc de positionnement n'a subi aucune modification dans le QR code et `False` sinon. *On testera donc chaque pixel.*

Exemple :

```
>>> test_bloc(0, 0, qr_code) ##qr_code est la liste de liste du QRcode
True
>>> test_bloc(1, 3, qr_code)
False
```

Solution

```
def test_bloc(x, y, qr_code):
    bloc = cree_bloc(7)
    for i in range(7):
        for j in range(7):
            if qr_code[y+j][x+i] != bloc[j][i]:
                return False
    return True
```

On considère qu'un QR code est bien positionné lorsque ses 3 blocs de contrôle sont effectivement présents en haut à gauche, en haut à droite et en bas à gauche (comme sur la figure 2).

Question 15

Écrire une fonction `test_QRcode` qui a pour paramètre une liste de listes `qr_code` de dimension 21 × 21 associée au QR code et qui renvoie `True` si le QR code est bien positionné et `False` sinon.

Exemple :

```
>>> test_QRcode(qr_code)
True
```

Solution

```
def test_QRcode(qr_code):
    return test_bloc(0, 0, qr_code) and test_QRcode(14, 0, qr_code) and
    ↪  test_QRcode(0, 14, qr_code)
```

Lors de la lecture d'un QR code par un appareil dédié (scanner, caméra ou autre) le processus de lecture permet de placer un QR code dans l'une des quatre positions possibles, comme illustré dans la figure 4. Cela dépend bien évidemment de l'orientation du QR code lors de sa lecture.

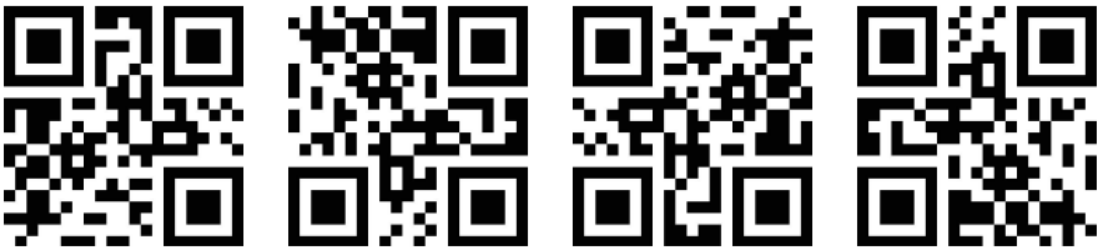

FIGURE 4 – Les 4 positions possibles lors de la lecture d'un QR code

Question 16

Écrire une fonction `rotation_horaire` qui a pour paramètre une liste de listes `qr_code` de dimension 21×21 associée au QR code et qui réalise la rotation de 90° dans le sens des aiguilles d'une montre d'un QR code. Cette fonction modifie la liste de listes `qr_code` et ne renvoie rien.

Remarque : On utilise la fonction `deepcopy` de la bibliothèque `copy` pour créer une copie de la liste de listes.

```
>>> from copy import deepcopy
>>> qr_code_copie = deepcopy(qr_code)
```

Solution

```
from copy import deepcopy
def rotation_horaire(qr_code):
    qr_code_copie = deepcopy(qr_code)
    n = len(qr_code)
    for i in range(n):
        for j in range(n):
            qr_code[i][j] = qr_code_copie[n-1-j][i]
```

Question 17

Connaissant les 4 positions possibles lors de la lecture d'un QR code par un appareil dédié, écrire la fonction `QRcode_posi` qui a pour paramètre une liste de listes `qr_code` de dimension 21×21 associée au QR code et qui positionne correctement un QR code. Cette fonction modifie la liste de listes `qr_code` et ne renvoie rien.

Solution

```
def QR_code_posi(qr_code):
    while not test_QRcode(qr_code):
        rotation_horaire(qr_code)
```

Annexe : Rappel sur les chaînes de caractères

On considère `ch` et `txt` deux chaînes de caractères et `n` un entier.

- `int(ch)` convertit la chaîne de caractères `ch` en `int`.
  Exemple : si `ch` vaut `"290"`, `int(ch)` vaut 290.

- `str(n)` convertit l'entier `n` en chaîne de caractères.
  Exemple : si `n` vaut 290, `str(n)` vaut `"290"`.

- `ch[i]` est la chaîne de caractères composée du caractère de `ch` d'indice `i`.
  Exemple : si `ch` vaut `"Bonjour"`, `ch[2]` vaut `'n'`.

- `len(ch)` renvoie le nombre de caractères de la chaîne de caractères `ch`.
  Exemple : si `ch` vaut `"Bonjour"`, `len(ch)` renvoie 7.

- `'a' in ch` renvoie `True` si le caractère `'a'` est dans `ch` et `False` sinon.
  Exemple : si `ch` vaut `"Bonjour"`, `'o' in ch` renvoie `True` et `'b' in ch` renvoie `False`.

- `ch + txt` est la chaîne de caractères résultant de la concaténation des deux chaînes de caractères `ch` et `txt`.
  Exemple : si `ch` vaut `"Bonjour"` et `txt` vaut `" !!!"`, `ch + txt` vaut `"Bonjour !!!"`.

- Parcours d'une chaîne de caractères :

```
for k in range(len(ch)):
    print(ch[k])
```

```
for elt in ch:
    print(elt)
```

  Ces deux programmes affichent chaque caractère de la chaînes de caractères `ch`.
  Exemple : si `ch` vaut `"Info"`, les deux programmes affichent : `"I"`
  `"n"`
  `"f"`
  `"o"`