Concours blanc - Informatique - 2h

Gestion de Tests dans une entreprise

Une entreprise d'e-commerce vend des meubles tous identifiés par une référence et par un QR code. 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 1 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 (1 er 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 du code du département de naissance, suivis de 3 chiffres du code commune officiel de l'Insee) ;
- Numéro d'ordre permettant de distinguer les personnes nées au même lieu à la même période (les 3 chiffres suivants).

Les deux derniers chiffres, compris entre 01 et 97, permettent de déterminer la clé, appelée aussi "clé de contrôle", qui permettra 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. Ne pas oublier qu'il est toujours possible de transformer un nombre entier en une chaîne de caractères composées de chiffres (fonction **str**) et réciproquement (fonction **int**).

Q1. Écrire la fonction num_secu qui, à partir de la chaîne de caractères d'un numéro de sécurité sociale, donne le numéro à 13 chiffres (NIR) et la clé de contrôle.

Cette fonction a un paramètre de type string et renvoie un tuple de deux éléments de type **int**.

Exemple :

```
>>> num_secu('2 91 01 75 018 002 68')
(2910175018002, 68)
```

Q2. Écrire la fonction clef qui détermine la valeur de la clé d'un numéro de sécurité sociale. Cette fonction a un paramètre de type **int** (les 13 premiers chiffres d'un numéro de sécurité sociale) et renvoie un élément de type **int**.

Exemple :

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

Q3.
Écrire la fonction test_num_secu qui détermine si un numéro de sécurité sociale est correct. Cette fonction a un paramètre de type string et retourne un élément de type **bool**.

Exemples :

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

Partie 2 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. 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 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. 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 (exemple si on obtient 16, on additionne 1 et 6 pour avoir 7). 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. On commence par le chiffre 2, celui se trouvant le plus à droite. 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 égale à 20. Ce résultat est un multiple de 10, le nombre 4762 est donc correct au sens de l'algorithme de Luhn.

Q4. Écrire une fonction num_en_liste qui transforme un nombre entier en une liste de chiffres. Cette fonction a un paramètre de type **int** et retourne un élément de type **list**.

Exemple :

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

Q5. Écrire une fonction tuple_pairs_impairs qui détermine un tuple représentant la liste des chiffres d'indice pair et la liste des chiffres d'indice impair d'un numéro de carte de crédit. Le chiffre le plus à droite de ce numéro est considéré comme le premier chiffre d'indice impair. Cette fonction a un paramètre de type **int** et retourne un tuple composé de deux éléments de type **list**.

Exemple :

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

Q6. Écrire une fonction cree_dico qui, à partir d'un numéro de carte de crédit, crée un dictionnaire avec deux clés nommées 'pair' et 'impair'. La clé 'pair' est constituée de la liste des nombres d'indice pairs du numéro de la carte de crédit et la clé 'impair' de la liste des nombres d'indice impairs.

Exemple :

```
>>> cree_dico(4532015112830465)
{'pair': [6, 0, 8, 1, 5, 0, 3, 4], 'impair': [5, 4, 3, 2, 1, 1, 2, 5]}
```

Q7. Écrire une fonction traitement_nb_pairs qui multiplie par 2 tous les chiffres de la liste associée à la clé 'pair'. Si un chiffre est supérieur à 9, il faut réaliser la somme des deux chiffres qui le composent. Cette fonction a un paramètre de type dictionnaire et retourne un dictionnaire.

La partie correspondant à la clé 'impair' n'est pas modifiée par le traitement de cette fonction.

Exemple :

```
>>> un_dico=cree_dico(4532015112830465)
>>> traitement_nb_pairs(un_dico)
{'pair': [3, 0, 7, 2, 1, 0, 6, 8], 'impair': [5, 4, 3, 2, 1, 1, 2, 5]}
```

Q8. Écrire une fonction test_num_carte_credit qui utilise l'algorithme de Luhn pour savoir si un numéro de carte de crédit est correct. Vous devez utiliser la fonction traitement_nb_pair pour sa réalisation. Cette fonction a un paramètre de type **int** et retourne une valeur de type **bool**.

Exemple :

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

Partie 3 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 codés dans le format RGB. 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 qui utilise une matrice de 21×21 pixels pour sa représentation.

Figure 1. Exemple de QR code version 1

Un 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.

Dans la suite de cette partie, nous n'utiliserons que les QR codes de version 1 et on s'interessera aux blocs de positionnements (figure 2).

Figure 2. Bloc de positionnement

Q9. Écrire une fonction init qui réalise l'initialisation d'une liste de dimension n où chaque élément est également une liste de dimension n. Cette liste de listes représente ainsi une matrice de taille $n \times n$. Cette fonction a un paramètre de type **int** et retourne une liste de listes qui représente un QR code initialisé avec des valeurs 0.

Exemple :

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

On prend comme bloc de positionnement celui représenté dans la figure 2.

Q10. Écrire une fonction cree_bloc qui crée un bloc de positionnement de taille $n \times n$. Cette fonction, qui a pour paramètre n, retourne une liste de listes de dimension $n \times n$ contenant :

— des 1 sur les lignes 1 et $n-2$ entre les colonnes 1 et $n-2$ ainsi que sur les colonnes 1 et $n-2$ entre les lignes 1 et colonne $n-2$,

— des 0 partout ailleurs.

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]]
```

Q11. Écrire une fonction test_bloc qui teste si un bloc de positionnement (rappel : il y en a trois de taille 7×7) est bien représenté pixel par pixel dans un QR code. Cette fonction a 3 paramètres : les coordonnées x et y donnant la position du début d'un bloc de positionnement d'un QR code (toujours les coordonnées du pixel le plus haut et à gauche) et la liste de listes de dimension 21×21 associée au même QR code. Cette fonction retourne un booléen.

Remarque : on cherche à tester si un bloc de positionnement d'un QR code n'a pas subi une modification. Les coordonnées du pixel le plus haut et à gauche pour le premier bloc sont égales à (0,0), pour le second bloc à (0,14) et pour le troisième bloc à (14,0).

Exemples :

```
>>> test_bloc(0,0, mat1)
True
>>> test_bloc(1,3, mat1)
False
```

Q12. 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 1). Écrire une fonction test_QRcode qui permet de tester si un QR code est bien positionné. Cette fonction a pour paramètre une matrice de dimension 21×21 et retourne un booléen.

Exemple :

```
>>> test_QRcode(mat1)
True
```

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 3. Cela dépend bien évidemment de l'orientation du QR code lors de sa lecture.

On se propose de faire tourner un QR Code par rotation successive de 90° afin qu'il puisse se trouver dans la bonne position comme celui de la figure 1.

Figure 3. Positions possibles d'un QR code

La rotation de 90° dans le sens d'horaire d'un élément de coordonnées (x, y) donne un élément de coordonnées $(y, n-1-x)$.

Q13. Écrire une fonction rotationHoraire qui réalise la rotation de 90° d'un QR code. Cette fonction a un seul paramètre, une liste de listes, et elle renvoie la liste de listes obtenues après rotation.

Par exemple, dans la figure 3 cette fonction réalisera la première rotation de 90° du QR code.

Q14. Connaissant les 4 positions possibles lors de la lecture d'un QR code par un appareil dédié, écrire la fonction QRcode_posi qui positionne correctement un QR code. Cette fonction a un seul paramètre, une liste de listes représentant un QR code, et renvoie une liste de listes représentant le QR code correctement positionné.

Indication : utiliser les fonctions rotationHoraire et test_QRcode.

Partie 4 Requêtes SQL

L'entreprise possède une base de données nommée Gestion_Entreprise constituée de trois tables : clients, produits et ventes. Les contenus de ces tables se trouvent à la page suivante.
La table "clients" est constituée de 5 champs :

- **id** : de type INTEGER (clé primaire auto-incrémentée) ;
- num_secu : de type INTEGER (entier de 15 chiffres) ;
- nom : de type TEXT ;
- prenom : de type TEXT ;
- num_CB : de type INTEGER.

La table "produits" est constituée de 5 champs :

- **id** : de type INTEGER (clé primaire auto-incrémentée) ;
- ref_produit : de type INTEGER ;
- nom_produit : de type TEXT ;
- qrcode : de type TEXT ;
- prix : de type DECIMAL.

La table "ventes" est constituée de 3 champs :

- date : de type TEXT ;
- ref_produit : de type INTEGER (clé étrangère, pointe vers la clé primaire **id** de la table produits) ;
- num_client : de type INTEGER (clé étrangère, pointe vers la clé primaire id de la table clients).

Les dates dans cette table sont définies par une chaîne de 10 caractères suivant le format année-mois-jour. Exemple de dates : '2019−06−01', '2022−12−30'.

Q15. Écrire, en SQL, la requête (1) qui permet d'obtenir le numéro de carte de crédit de toutes les personnes référencées dans la base de données de l'entreprise dont le numéro de sécurité sociale commence par '1 75'.

Q16. Écrire, en SQL, la requête (2) permettant d'obtenir le nom et le prénom de toutes les personnes ayant effectué un achat avec un résultat sans doublon.

Q17. Écrire, en SQL, la requête (3) permettant d'obtenir la liste de tous les produits avec leur prix, classée du produit le plus cher au produit le moins cher.

Q18. Ecrire, en SQL, la requête (4) permettant d'obtenir le montant des ventes le 20 décembre 2024.

Q19. Ecrire, en SQL, la requête (5) permettant d'obtenir le montant des ventes chaque jour.

Q20. Écrire, en SQL, la requête (6) qui permet d'obtenir les produits associés à chaque numéro de carte de crédit du client et qui ont été vendus entre le 1er juin 2024 et le 30 juillet 2024.

On rappelle que SQL compare les variables de type TEXT grâce à l'ordre lexicographique : par exemple "13-06-1989 ¡ 13-07-1999" est vrai.

Base de données

Le contenu des tables clients, produits et ventes de la base de données Gestion_Entreprise est donné ci-après.

Table "clients"

id	num_secu	nom	prenom	num_CB
1	286128817863441	Eldyn	Sophie	6767342589219928
2	298082934500890	Gomez	Sophie	2324563490665454
3	298082934500896	Ruiza	Maria	9889454573204522
4	109086723487917	Kovitz	Flor	6789543778653678
5	175105642102321	Settin	Boris	4745342178563217
6	189027511732543	Val	Erwan	7856432167453492
7	191017511318196	Mottreff	Georges	8787564521392354
...	...	...	...	

Table "produits"

id	ref_produit	nom_produit	qrcode	prix
1	27	Buffet chêne	27.jpg	320
2	102	Chaise rustique	102.jpg	65
3	453	Table ronde	453.jpg	75
4	756	Table ovale	756.jpg	120
5	921	Coffret Bali	921.jpg	170
...	...	...	...	...

Table "ventes"

date	ref_produit	num_client
2024-05-15	2	2
2024-06-17	3	1
2024-06-21	3	7
2024-07-19	4	5
2024-08-19	5	4
2024-09-05	4	6
...	...	...