# DS2 /52

# Gestion de Tests dans une entreprise

## Partie I - Tests de code de sécurité sociale

1. Écrire la fonction `num_secu` qui, à partir de la chaîne de caractères d'un numéro de sécurité sociale, donne le numéro sous la forme d'un entier. Le programme devra parcourir la chaîne de caractères représentant le numéro de sécurité sociale en supprimant les caractères d'espacement, puis la transformer en un nombre entier. Cette fonction a un paramètre de type `string` et retourne une valeur de type `int`.
   Exemple :
   ```
   >>> num_secu('2 91 01 75 018 002')
   2910175018002
   ```
   - **3 pts**

```python
def num_secu(code) :
    ''num_secu(code : str) -> int''
    code_se = ''
    for c in code :
        if c != ' ' :
            code_se = code_se + c
    return int(code_se)
```

2. 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` et retourne un élément de type `int`.
   Exemple :
   ```
   >>> clef(2910175018002)
   68
   ```
   - **1 pt**

```python
def clef(num) :
    ''clef(num : int) -> int''
    reste = num % 97
    return 97 - reste
```

3. Écrire la fonction `num_secu_complet` qui détermine le numéro complet de sécurité sociale. Cette fonction a un paramètre de type `int` et retourne un élément de type `int`.
   Exemple :
   ```
   >>> num_secu_complet(2910175018002)
   291017501800268
   ```
   - **1 pt**

```python
def num_secu_complet(num) :
    ''num_secu_complet(num : int) -> int''
    return 100 * num + clef(num)
```

*4.* 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`.

- **3 pts**

```
def test_num_secu(code) :
    ''test_num_secu(code : str) -> bool''
    code_int = num_secu(code)
    code_sans_cle = code[:-2]
    num = num_secu(code_sans_cle)
    num = num_secu_complet(num)
    return num == code_int
```

## Partie II - Test de numéro de carte de crédit

*5.* 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]
```

- **2 pts**

```
def num_en_liste(num) :
    ''num_en_liste(num : int) -> list''
    chaine = str(num)
    return [int(c) for c in chaine]
```

*6.* 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])
```

- **3 pts**

```
def tuple_pairs_impairs(num) :
    ''tuple_pairs_impairs(num : int) -> (L1 : list, L2 : list)''
    L = num_en_liste(num)
    n = len(L)
    L1 = []
    L2 = []
    for k in range(n) :
        if (n-1 - k) % 2 == 0 :
            L1.append( L[n-1-k] )
        else :
            L2.append( L[n-1-k] )
    return (L1, L2)
```

***7.*** 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]}
```

- **1 pt**

```
def cree_dico(num) :
    ''cree_dico(num : int) -> dict''
    T = tuple_pairs_impairs(num)
    return {'pair' : T[0], 'impair' : T[1]}
```

***8.*** 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.
Remarque : 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]}
```

- **3 pts**

```
def traitement_nb_pairs(dico) :
    ''traitement_nb_pairs(dico : dict) -> dico : dict''
    L = dico['pair']
    n = len(L)
    for i in range(n) :
        nb = 2 * L[i]
        q = nb // 10
        r = nb % 10
        L[i] = q + r
    dico['pair'] = L
    return dico
```

***9.*** 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
```

- **2 pts**

```
def test_num_carte_credit(num) :
    ''test_num_carte_credit(num : int) -> bool''
    dico = cree_dico(num)
    dico = traitement_nb_pairs(dico)
    S = sum(dico['pair']) + sum(dico['impair'])
    return S % 10 == 0
```

## Partie III - Tests de QR code

***10.*** 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]]
```

- **1 pt**

```python
def init(n) :
    ''init(n : int) -> list(list)''
    return [ [0 for _ in range(n)] for _ in range(n)]
```

***11.*** Écrire la fonction `charge_valeur` qui a pour but de réduire les données de l'image dans une liste de listes de dimension $21 * 21$. Attention : l'image correspondant à un QR code représente une liste de listes de dimension 420*420 dont on veut réduire tous les blocs constitués de 20*20 pixels à un seul pixel pour avoir à partir de l'image une liste de listes de dimension 21*21. Ne pas oublier que tous les pixels d'un bloc sont identiques. Cette fonction a un paramètre de type image et retourne une liste de listes de triplets (couleur des pixels).
Indication : utiliser la fonction `getpixel` du module Python `Gestion_QRCode` (voir sa définition dans l'**annexe 1**).

- **2 pts**

```python
def charge_valeur(image) :
    ''charge_valeur(image : image) -> reduc : list(list)''
    reduc = init(21)
    for i in range(21) :
        for j in range(21) :
            reduc[i][j] = image.getpixel(20 * i, 20 * j)
    return reduc
```

***12.*** Écrire une fonction `cree_bloc` qui crée un bloc de positionnement. Cette fonction, qui n'a pas de paramètre, retourne une liste de listes de dimension 7*7.

- **2 pts**

```python
def cree_bloc() :
    ''cree_bloc() -> list(list)''
    A = [0 for _ in range(7)]
    B = [0] + [1 for _ in range(5)] + [0]
    C = [0, 1, 0, 0, 0, 1, 0]
    return [A[:], B[:], C[:], C[:], C[:], B[:], A[:]]
```

***13.*** Écrire une fonction `test_bloc` qui teste si un bloc de positionnement (rappel : il y en a trois) 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.

- **3 pts**

```
def test_bloc(x, y, mat) :
    ''test_bloc(x : int, y : int, mat : list(list)) -> bool''
    bloc_pos = cree_bloc()
    for i in range(7) :
        for j in range(7) :
            if mat[x + i][y + j] != bloc_pos[i][j] :
                return False
    return True
```

***14.*** 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 ??**). 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
```

- **1 pt**

```
def test_QRcode(mat) :
    ''test_QRcode(mat : list(list)) -> bool''
    return test_bloc(0,0,mat) and test_bloc(0,14,mat) and test_bloc(14,0,mat)
```

***15.*** Écrire une procédure [(1)] `tourHoraire` qui réalise une rotation de 90°, dans le sens des aiguilles d'une montre, des 4 éléments du QR code. La fonction a trois paramètres, les coordonnées $x$ et $y$ d'un élément de la liste de listes et une liste de listes de dimension 21*21.
Exemple :

```
>>> tourHoraire(0, 1, mat1)
```

- **4 pts**

```
def tourHoraire(x, y, mat) :
    ''tourHoraire(x : int, y : int, mat : list(list)) -> None''
    n = len(mat) - 1
    pix = mat[x][y]
    mat[x][y] = mat[n-y][x]
    mat[n-y][x] = mat[n-x][n-y]
    mat[n-x][n-y] = mat[y][n-x]
    mat[y][n-x] = pix
    return None
```

(1). Une procédure est une fonction qui retourne la valeur None mais cette valeur n'est pas destinée à être utilisée ou à être capturée.

***16.*** Écrire la procédure `rotationHoraire` qui réalise la rotation de 90° d'un QR code. Cette procédure a un seul paramètre, une liste de listes de dimension 21*21.
Par exemple, dans la **figure ??** cette fonction réalisera la première rotation de 90° du QR code.

- **4 pts**

```
def rotationHoraire(mat) :
    ''rotationHoraire(mat : list(list)) -> None''
    n = len(mat)
    for i in range(n//2) :
        for j in range(i, n-i-1) :
            tourHoraire(i, j, mat)
    return None
```

***17.*** Connaissant les 4 positions possibles lors de la lecture d'un QR code par un appareil dédié, écrire la procédure `QRcode_posi` qui positionne correctement un QR code. Cette procédure a un seul paramètre, une liste de listes de dimension 21*21.
Indication : utiliser les fonctions `rotationHoraire` et `test_QRcode`.

- **2 pts**

```
def QRcode_posi(mat) :
    '' QRcode_posi(mat : list(list)) -> None''
    for i in range(3) :
        if test_QRcode(mat) :
            return None
        else :
            rotationHoraire(mat)
```

## Partie IV - Gestion réseau

***18.*** Compléter les 3 lignes manquantes du tableau $M+$ sur le **DR**.

- **2 pts : contenu des 2 dernières cases de la $1^{ère}$ colonne**
- **3 pts : contenu des 3 dernières lignes des 6 dernières colonnes**

| | **A** | **B** | **C** | **D** | **E** | **F** |
|---|---|---|---|---|---|---|
| étape initiale | 0 | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| $A(0)$ | - | 2 | $\infty$ | ① | $\infty$ | $\infty$ |
| $D\,(1_A)$ | - | ② | $\infty$ | - | 3 | $\infty$ |
| $B\,(2_A)$ | - | - | 5 | - | ③ | 7 |
| $E\,(3_D)$ | - | - | 5 | - | - | ④ |
| $F\,(4_E)$ | - | - | ⑤ | - | - | - |
| $C\,(5_B)$ | - | - | - | - | - | - |

***19.*** Donner la valeur du plus court chemin entre "A" et "F". Expliquer comment on obtient cette valeur à l'aide du tableau $M+$. Expliciter le chemin le plus court trouvé pour aller de "A" à "F".

- **1 pt : le plus court chemin entre $A$ et $F$ est de longueur $4$ car...**
- **1 pt : le plus court chemin entre $A$ et $F$ est : $A - D - E - F$ car...**

## Partie V - Requêtes SQL

***20.*** Écrire, en SQL, la requête (**1**) qui permet d'obtenir le numéro de la carte de crédit de toutes les personnes référencées dans la base de données de l'entreprise dont le numéro de sécurité sociale commence par 2. On utilisera le caractère '_' comme séparateur des milliers. Par exemple 10000000 sera réécrit comme 10_000_000.

- **1 pt**

```
SELECT num_CB FROM clients WHERE num_secu > 199_999_999_999_999
```

***21.*** 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.

- **2 pts**

```
SELECT DISTINCT clients.nom, clients.prenom FROM clients
JOIN ventes ON clients.id = ventes.num_client
```

***22.*** Écrire, en SQL, la requête (**3**) qui permet d'obtenir les produits associés à chaque numéro de carte de crédit du client et qui ont été vendus entre le 1 juin 2020 et le 30 juillet 2020. On rappelle que SQL compare les variables de type `TEXT` grâce à l'ordre lexicographique.
Par exemple '1989-06-13 $<$ 1999-07-13' est vrai.

- **4 pts**

```
SELECT produits.nom_produit, clients.num_CB FROM produits
JOIN ventes ON produits.ref_produit = ventes.ref_produit
JOIN clients ON clients.id = ventes.num_client
WHERE (ventes.date >= 2020-06-01) AND (ventes.date <= 2020-07-30)
GROUP BY clients.num_CB
```