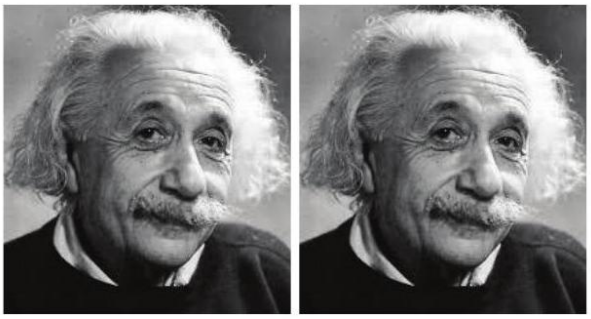
Une entreprise possède 5 sites de production. Le Directeur Général veut améliorer la sécurité. Il souhaite attribuer à chaque employé une carte personnelle et infalsifiable lui permettant l'accès à son site de production et peut-être à d'autres sites de l'entreprise. Chaque employé est affecté uniquement à un site que l'on appellera son site d'origine.

Chaque site de production ne pourra compter plus de 100 employés. La carte attribuée à l'employé comportera sa photo d'identité ainsi qu'une puce. Le code représentant le nom de la personne sera intégré dans la photo mais pas dans la puce.

Un tel code a été placé dans la photo de droite de la figure 1. Comme on peut le remarquer, il y a très peu de différences entre les deux photos, l'une sans code, l'autre avec un code intégré.



Chaque employé sera référencé par un code composé d'un entier compris entre 1 et 5 (indiquant le site de référence où il travaille) ainsi qu'une séquence de 7 caractères (pris parmi les 7 premiers caractères de l'alphabet en majuscules : 'A',...,'G'). Le site sera codé en machine sur 3 bits. Par exemple, pour un employé travaillant dans le site n°3, le code pourrait être 3DABGEFC. Ce code n'affiche pas le nom de l'employé mais un automate permet, à partir des 7 caractères, de le retrouver.

► **Q1.** Donner le nombre maximal d'employés par site ainsi que le nombre maximal de sites que pourrait gérer cette entreprise avec ce type de codage. La réponse peut être donnée par une expression numérique sans chercher à la calculer.
Indiquer si la technique de codage est suffisante pour gérer le personnel de cette entreprise.

Partie 1 – sur la distance de Levenshtein

La problématique est de savoir si les codes créés pour gérer les employés sont suffisamment disjoints les uns des autres, c'est-à-dire si la mesure de la différence entre deux codes est suffisamment importante. Cela est possible grâce à la distance de Levenshtein qui peut être utilisée pour analyser tous les codes. L'entreprise veut tester tous les codes associés aux employés et changer les codes de tous ceux dont la distance de Levenshtein est inférieure à la valeur 3.

On considère les opérations suivantes sur une chaîne de caractères `a` :

- suppression d'un caractère de `a` ;
- insertion d'un caractère dans `a` ;
- remplacement d'un caractère de `a` par un autre caractère.

On peut remarquer la «commutativité» de ces opérations (on peut les faire dans n'importe quel ordre).

► **Q2.** Ces opérations sur les chaînes de caractères sont-elles anodines en Python ?

Dans cette partie, nous allons développer les fonctions permettant de répondre à cette problématique. La distance de Levenshtein est une valeur donnant une mesure de la différence entre deux chaînes de caractères. Elle est égale au nombre minimal de caractères qu'il faut supprimer, insérer ou remplacer pour passer d'une chaîne A à une chaîne B. Par exemple, on peut passer du mot «`niche`» au mot «`chien`» par la suite d'opérations suivantes :

$$\texttt{niche} \to \texttt{ciche} \to \texttt{cche} \to \texttt{che} \to \texttt{chie} \to \texttt{chien}.$$

Cette transformation a un coût de 5 opérations élémentaires, ce qui n'est pas optimal.

On considère que chaque opération élémentaire mise en œuvre (supprimer, insérer ou remplacer) a un coût de 1 ; la distance de Levenshtein est la somme de ces coûts. Par exemple, la distance de Levenshtein entre les chaînes `Soveil` et `Soleil` est 1 car il a fallu (et il a suffit de) réaliser une seule transformation, une substitution pour remplacer `v` par `l` dans la chaîne `Soveil`.

► **Q3.** On s'intéresse à la distance de Levenshtein entre les chaînes de caractères `Auttame` et `Automne`.
Écrire, en Python, les transformations (insertion, substitution, suppression) appliquées à la chaîne `Auttame` afin d'obtenir la chaîne `Automne`. On déterminera systématiquement la valeur de la chaîne, notée `s`, avant et après chaque transformation ainsi que le nom de la transformation appliquée.
Quelle est la distance de Levenshtein entre ces deux chaînes.

De façon générale, on considère deux chaînes de caractères $\texttt{a} = \texttt{a}_0\texttt{a}_1\ldots\texttt{a}_{n-1}$ et $\texttt{b} = \texttt{b}_0\texttt{b}_1\ldots\texttt{b}_{p-1}$ et on note $d(i,j)$ la distance de Levenshtein (c'est-à-dire le nombre minimal d'opérations) entre le préfixe $\texttt{a}_0\ldots\texttt{a}_{i-1}$ (de longueur i) de `a` et le préfixe $\texttt{b}_0\ldots\texttt{b}_{j-1}$ (de longueur j) de `b`.

► **Q4.** Exprimer $d(i,0)$ et $d(0,j)$ pour tous $(i,j) \in [\![0,n]\!] \times [\![0,p]\!]$.

► **Q5.** Soit $i \geqslant 1$ et $j \geqslant 1$. Si `a[i-1]` et `b[j-1]` sont égaux, quel lien y a-t-il alors entre $d(i,j)$ et $d(i-1,j-1)$?

► **Q6.** Si $i \in [\![1,n]\!]$ et $j \in [\![1,p]\!]$ sont tels que `a[i-1]` et `b[j-1]` ne soient pas égaux, justifier que l'on a alors :

$$d(i,j) = 1 + \min\{d(i-1,j)\ ;\ d(i,j-1)\ ;\ d(i-1,j-1)\}.$$

► **Q7.** Écrire une fonction Python appelée `mini` prenant en arguments trois entiers et renvoyant leur minimum.

► **Q8.** Compléter la fonction Python suivante afin quelle renvoie la distance de Lenvenshtein entre les chaînes de caractères `a` et `b` (il y a 5 zones à compléter) :

```
def distance(a, b):
    """ entrée : a et b deux chaînes de caractères
        sortie : entier correspondant à la distance de Levenshtein entre a et b
    """
    n, p = len(a), len(b)
    d = [[0 for j in range(p+1)] for i in range(n+1)]
    for i in range(n+1):
        ......
    for j in range(p+1):
        ......
    for i in range(1, n+1):
        for j in range(1, p+1):
            if a[i-1] == b[j-1]:
                ......
            else:
                ......
    return ......
```

► **Q9.** Quelle est la complexité (dans le pire des cas) de la fonction précédente ?

On revient à la problématique initiale et l'on suppose que tous les codes ont été rassemblés dans une liste nommée `table_code`.

► **Q10.** Écrire une fonction `code_bon_lvs` qui supprime tous les codes de la liste `table_code` qui sont à une distance de Levenshtein d'un autre code inférieure ou égale à 3. Les codes à supprimer seront remplacés dans la liste `table_code` par le code nul (c'est-à-dire la chaîne de caractères `'0000000'`).

► **Q11.** Écrire une fonction `pourcentage_lvs` qui détermine le pourcentage de codes nuls dans la liste `table_code`.

Partie 2 – gestion des données

Nous allons nous intéresser à la gestion des données qui sont stockées dans la base de données nommée `Personnel` et sera constituée de deux tables intitulées `Employes` et `ListeCategories`. Des extraits des contenus de ces tables se trouvent en dernière page du sujet.

La table `Employes` est constituée de 8 champs :

- `id` : de type entier ;
- `nom` : de type chaîne de caractères ;
- `prenom` : de type chaîne de caractères ;
- `email` : de type chaîne de caractères - clé primaire (l'adresse email est définie sans son extension @cpp.com dans la table) ;
- `age` : de type entier ;
- `code` : de type chaîne de caractères (voir descriptif plus loin) ;
- `site` : de type entier - liste des sites où l'employé est autorisé à entrer en plus de son site d'origine (voir descriptif plus loin) ;
- `code_categorie` : de type entier - indice où la catégorie est référencée dans la table nommée ListeCategories.

La table `ListeCategories` est constituée de 2 champs :

- `id` : de type entier - clé primaire ;
- `categorie` : de type chaîne de caractères - liste des métiers de l'entreprise.

Des lecteurs de codes sont installés aux portes des différents sites afin de limiter les accès aux seules personnes habilitées à y entrer.

Rappelons que le code est défini sous la forme d'une chaîne de 8 caractères, le premier caractère indiquant le numéro du site auquel appartient une personne. Par défaut, chaque personne est attribuée à un et un seul site, son site d'origine. En revanche, il est possible à toute personne de cette entreprise de se déplacer dans d'autres sites si l'autorisation en a été donnée (champ site de la table `Employes`). Le reste des caractères correspond à la clé de sécurité associée à chaque employé.
La gestion du champ site est particulière. On définit un entier qui permet de connaître les sites où l'employé est autorisé à entrer. On attribue les valeurs suivantes :

- 1 : pour le site numéroté 1 ;
- 2 : pour le site numéroté 2 ;
- 4 : pour le site numéroté 3 ;
- 8 : pour le site numéroté 4 ;
- 16 : pour le site numéroté 5.

Ainsi, si un employé appartenant au site 1 est autorisé à entrer dans les sites 2 et 3 , la valeur du champ site aura comme valeur $2+4$, soit 6. Pour un autre employé appartenant au site 4 autorisé à entrer dans les sites 1,2 et 5, la valeur du champ site sera $1+2+16$, soit 19.
Attention : comme on peut le remarquer sur les deux exemples précédents, le numéro du site d'origine (le site où l'employé travaille par défaut) n'est jamais pris en compte dans le calcul donnant la valeur du champ site.

► **Q12.** Écrire une fonction `liste_site` donnant la liste des sites où un employé peut se rendre en plus de son site d'origine dès que l'on donne une valeur (un entier) représentant la valeur du champ site de la table `Employes`. Si la valeur donnée est incorrecte, la fonction `liste_site` retournera une chaîne vide.
L'ordre des sites dans le résultat n'a pas d'importance.

L'équipe de direction souhaite avoir certaines informations au sujet des employés de l'entreprise.

► **Q13.** Écrire en SQL une requête donnant le nom et le prénom des employés de plus de 50 ans.

► **Q14.** Écrire en SQL une requête donnant comme résultat l'adresse email (sans le nom de domaine) des employés pouvant accéder uniquement aux sites 3 et 4 en plus de leur site d'origine.

► **Q15.** Écrire en SQL une requête donnant le nombre de personnes qui ne sont pas originaires du site 2 mais qui peuvent y accéder.

► **Q16.** Écrire en SQL une requête donnant comme résultat le nom et la catégorie des personnes de l'entreprise ayant au moins 20 ans. Les noms seront classés par ordre alphabétique.

► **Q17.** Écrire en SQL donnant le nombre de personnes de plus de 40 ans pour chacune des différentes catégories.

► **Q18.** Écrire en SQL une requête donnant les deux catégories ayant le plus d'employés de plus de 40 ans.

► **Q19.** Écrire en SQL une requête donnant les catégories ayant au moins 3 employés de moins de 25 ans.

Annexe - extraits de la base de donnée «Personnel»

Voici ci-dessous un extrait de la table `Employes` et l'intégralité de la table `ListeCategories` de la base de données Personnel.

id	nom	prenom	email	age	code	site	code_categorie
1	Genereux	Alain	alain.genereux	47	1AAACDEF	0	1
2	Tanguy	Alain	alain.tanguy	22	1BBACABE	10	8
3	Smith	Alan	alan.smith	47	5BCABCAE	5	2
4	Lefoll	Claude	claude.lefoll	28	2EEABECC	24	1
5	Herberts	Dany	dany.herberts	58	1EEEEABC	0	1
6	Korbs	Eva	eva.korbs	33	2FEAFEAB	9	5
7	Niels	Edwin	edwin.niels	24	2EFDDACA	28	4
8	Joly	Emilie	emilie.joly	25	3FAFABEF	19	4
9	Esteban	Flore	flore.esteban	20	4BECEBAA	12	2
10	Serin	Jacques	jacques.serin	22	2GBBCEAE	21	6
11	Clerc	Jerome	jerome.clerc	29	3DDAABBC	3	1
12	Brown	Katia	katia.brown	46	5AFBECCA	3	1
13	Forbs	Laura	laura.forbs	18	2CBAEAEC	0	1
14	Phil	Marc	marc.phil	33	4BCDABCD	17	7
15	Auzas	Michel	michel.auzas	32	5FECDAEA	15	7
16	Kotta	Michelle	michelle.kotta	27	2ABEEABC	9	7
17	Lambert	Pierre	pierre.lambert	35	3DABGEFC	10	2
18	Klader	Sylvie	sylvie.klader	32	2EAEAEAB	20	1
19	Durand	Sylvie	sylvie.durand	31	1CBBCAEA	16	6
20	Olm	Tatiana	tatiana.olm	25	4EFEFACB	19	3
...							

id	categorie
1	ingénieur
2	secrétaire
3	comptable
4	technicien
5	opérateur
6	administratif
7	chef de projet
8	responsable de département

Annexe - exemples de syntaxe sur les chaînes de caractères

```
>>> c = "Bientôt les vacances"
>>> c[0], c[3], c[9]
('B', 'n', 'e')
>>> c[:6]
'Bientô'
>>> c[4:]
'tôt les vacances'
>>> c[4:11]
'tôt les'
```