Concours Commun INP (CCINP), France

Session 2019, filière TSI

Informatique

Durée : 3 heures

Ce que traite le sujet

- Parties I et II : présentation et distance de Levenshtein
- Partie III : gestion des données
- Parties IV à VI : codage des couleurs et stéganographie

Sujet reproduit sans retouche à partir de la page suivante.

Résumé des parties, chapitres de cours liés et autres années :
prepahelp.com/annales/ccinp/tsi/2019/info

Prepa Help ne revendique aucun droit sur le sujet, propriété de l'organisateur du concours.

CONCOURS COMMUN INP

ÉPREUVE SPÉCIFIQUE - FILIÈRE TSI

INFORMATIQUE

Jeudi 2 mai : 14 h - 17 h

N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

Les calculatrices sont interdites

Le sujet comporte 12 pages. Seul le **Document Réponse** de 11 pages est à rendre en fin d'épreuve.

A

Remarques générales

Important : nous informons les candidats que chaque partie de ce sujet peut être traitée séparement. Vous pouvez utiliser toutes les fonctions des questions précédentes même si vous ne les avez pas implémentées.

Les réponses aux questions sont à rédiger sur le **Document Réponse** (**DR**) et ne doivent pas dépasser les dimensions des cadres proposés.

Si la réponse attendue est spécifique à un langage de programmation, seul le **langage Python** est permis.

Les structures algorithmiques doivent être clairement identifiables par des indentations visibles ou par des barres droites entre le début et la fin de la structure comme l'exemple ci-dessous :

```
si (Condition)
    | alors
    |     | Instructions
    | sinon
    |     | Instructions
fin si
```

Dans les questions où l'on demandera d'écrire une fonction, on donnera systématiquement sa signature, c'est-à-dire le nom de la fonction avec les types des paramètres ainsi que le type du résultat retourné par cette fonction.

Par exemple, la fonction **foo** qui prend en entrée deux paramètres de type entier et retourne une liste, aura comme description :

Signature de la fonction *foo* :

```
foo(int,int) -> list
```

SÉCURISATION DE L'ENTRÉE DU PERSONNEL D'UNE ENTREPRISE

Partie I - Présentation

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.

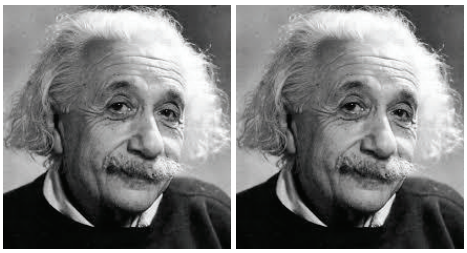


Figure 1 – Photos d'Einstein

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^o 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 II - 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. Ceci 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.
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.
On considère que chaque opération élémentaire mise en oeuvre (supprimer, insérer ou remplacer) a un coût de 1. On précise que chaque opération élémentaire ne porte que sur un seul caractère. La distance de *Levenshtein*, au sens mathématiques du terme, est la somme de ces coûts.
Par exemple, la distance de *Levenshtein* entre les chaînes `Soveil` et `Soleil` est de coût égal à 1 car il a fallu réaliser une seule transformation, une substitution pour remplacer 'v' par 'l' dans la chaîne `Soveil`.

On va s'intéresser à déterminer quelle est la distance de *Levenshtein* entre les chaînes de caractères `Auttame` et `Automne`.

Q2. Écrire, dans le langage 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.
Vous devez compléter le **Document Réponse**, à l'aide de *s=Auttame*.
Calculer le coût de la distance de *Levenshtein*.

L'algorithme qui vous est proposé ci-après décrit le calcul de la distance de *Levenshtein* entre deux chaînes de caractères. Cet algorithme retourne un entier (éventuellement nul) donnant la distance entre deux chaînes de caractères au sens de *Levenshtein*.

```
1  levenshtein(chaine1, chaine2) =
2  entree :
3      chaine1[], chaine de caracteres d'indice 1 a n avec n egal a longueurChaine1 ;
4      chaine2[], chaine de caracteres d'indice 1 a m avec m egal a longueurChaine2 ;
5  sortie :
6      entier : la valeur de la distance ;
7  declarer :
8      d[] : matrice de nombres entiers initialisée avec des 0, indice de 0 a n et de 0 a m ;
9      i, j, coutSubstitution : entier ;
10 debut :
11     pour i de 0 a n
12         d[i,0] := i
13     pour j de 0 a m
14         d[0,j] := j
15     pour i de 1 a n
16         pour j de 1 a m
17             si (chaine1[i] = chaine2[j])
18                 alors
19                     coutSubstitution := 0
20                 sinon
21                     coutSubstitution := 1
22             fin si
23             d[i,j] := minimum(
24                             d[i-1,j] + 1,                    # pour une suppression
25                             d[i,j-1] + 1,                    # pour une insertion
26                             d[i-1,j-1] + coutSubstitution)   # pour une substitution
27     retourner d[n, m]
28 fin
```

Q3. Déterminer la matrice d de l'algorithme de *Levenshtein* après l'instruction de la ligne 14 et après celle de la ligne 26, ainsi que la distance de *Levenshtein* lors de son exécution entre les chaînes 'GAZ' et 'LA'.

Q4. Déterminer la complexité de l'algorithme précédent.

Tous les codes ont été rassemblés dans une liste nommée *table_code*.

Q5. Écrire une fonction *code_bon_lvs* qui supprime tous les codes de la liste *table_code* dont la mesure de *Levenshtein* est inférieure ou égale à la valeur 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'.

Signature de la fonction *code_bon_lvs* :
```
code_bon_lvs(list) -> NoneType
```

Q6. Écrire une fonction *pourcentage_lvs* qui détermine le pourcentage de codes nuls dans la liste *table_code*. Le résultat sera une chaîne de caractères constituée d'un nombre entier (pas un nombre flottant) et du caractère %. On permet l'arrondi d'un calcul à l'entier inférieur ou à l'entier supérieur.

Signature de la fonction *pourcentage_lvs* :
```
pourcentage_lvs(list) -> str
```

Exemple :
```
>>> pourcentage_lvs(table_code)
'17%'
```

Partie III - 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*. Les contenus de ces tables se trouvent en **Annexe**, page 12.

La table *Employes* est constituée de 8 champs :

- *id* : de type INTEGER ;
- *nom* : de type TEXT ;
- *prenom* : de type TEXT ;
- *email* : de type TEXT – clé primaire (l'adresse email est définie sans son extension @cpp.com dans la table) ;
- *age* : de type INTEGER ;
- *code* : de type TEXT (voir descriptif plus loin) ;
- *site* : de type INTEGER – 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 INTEGER – indice où la catégorie est référencée dans la table nommée *ListeCategories* (clé étrangère).

La table *ListeCategories* est constituée de 2 champs :

- *id* : de type INTEGER – clé primaire ;
- *categorie* : de type TEXT – 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. Par contre, 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*.

Q7. 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.

Signature de la fonction *liste_site* :

```
liste_site(int) -> list
```

Exemple :

```
>>> liste_site(21)
[1, 3, 5]

>>> liste_site(50)
[]
```

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

Q8. Écrire en SQL la requête 1 suivante donnée en algèbre relationnel :

$$\pi_{nom,prenom,age}\left(\sigma_{age>50}\left(Employes\right)\right)$$

Q9. Écrire en SQL la requête 2 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.

Q10. Écrire en SQL la requête 3 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.

Q11. Écrire en SQL la requête 4 donnant comme résultat la liste des âges des employés avec comme information associée le nombre d'employés ayant le même âge. On demande que cette liste soit décrémentale par rapport au nombre de personnes ayant le même âge.
Par exemple, dans la table *Employes*, il y a 2 personnes qui ont 22 ans.

Partie IV - Codage des couleurs

La photo, une image couleur, est décrite informatiquement comme un tableau (une matrice) de pixels. Chaque pixel sera représenté par une couleur au format RGB[1].
Par exemple, la couleur d'un pixel pourrait être en format RGB (64,78,191). Si on écrit les trois composantes RGB en code binaire, on aura le triplet (0100000**0**, 0100111**0**, 1011111**1**). Le bit de poids faible de chacune des composantes RGB est représenté en gras. On va se servir de ce triplet de valeurs définies en gras pour coder une information dans l'image.
La modification du bit de poids faible a très peu de conséquences sur la représentation de l'image.

Q12. Donner le code RGB en décimal et en hexadécimal des éléments suivants :
- un pixel de couleur `bleu` ;
- un pixel de couleur `blanc`.

Un pixel est codé dans le format RGB en (10,10,10). Indiquez la couleur de ce pixel.

Q13. Donner pour le codage RGB le nombre de couleurs possibles. La réponse peut être donnée par une expression numérique sans chercher à la calculer.

1.
Le système RGB (`Red`, `Green`, `Blue`) ou en français (`Rouge`, `Vert`, `Bleu`), permet de coder les couleurs en informatique. Un écran informatique est composé de pixels représentant une couleur au format RGB.
La composante `R` est codée sur 8 bits de 0 à 255 en décimal et de 00 à FF en hexadécimal. Il en va de même pour les autres composantes. Le codage des couleurs va du plus foncé au plus clair.
Par exemple, la couleur d'un pixel `orange` pourrait avoir comme valeur (255,100,100). Un pixel sera de couleur `gris` si les composantes `R`, `G` et `B` sont identiques.

Partie V - Codage de l'information

Cette partie sera consacrée à l'implémentation de fonctions qui serviront notamment dans la ***partie VI****, réservée au décodage de l'information se trouvant dans la photo de l'employé(e).*

Les informations constituant le code sont définies dans une trame composée de blocs non consécutifs. Chaque bloc sera associé à un pixel. Un premier bloc est constitué d'un pixel pour représenter le numéro du site et est suivi d'une séquence de 7 blocs représentant chacun un caractère. Le premier bloc sera considéré comme le bloc de référence.
Pour un pixel codant le numéro du site de l'employé, les bits de poids faible indiquent directement le numéro au format binaire. Ainsi, pour l'exemple précédent, le codage correspondant au site n°3 sera (01000000, 01001111 , 10111111) puisque le triplet 011 correspond à la valeur 3 en code décimal. Par conséquent, on met une information dans l'image en modifiant uniquement les bits de poids faible, ce qui a peu de conséquences sur la représentation de cette image.
Pour un pixel codant une des lettres de l'identifiant de l'employé, on valide la convention suivante : le caractère 'A' sera associé à 001, le caractère 'B' à 010, jusqu'au caractère 'G' qui aura comme valeur 111.
Par exemple, un pixel codant le caractère 'B' (010) pourrait avoir pour format RGB (...0, ...1, ...0).
On ne traitera pas le caractère associé à 000 qui aura pour lettre 'W', réservée pour la gestion du personnel de service (gardiens, personnel de nettoyage, etc.) ayant l'autorisation d'entrer dans les sites de l'entreprise.
Chaque trame sera constituée de 8 blocs, c'est-à-dire 8 pixels (voir la **figure 2**).

- Un pixel de référence est défini par ses coordonnées que l'on nomme *posi* dans la suite.
- Le paramètre ou la variable *posi* est un couple de valeurs (colonne,ligne) qui est obtenu à l'aide d'une clé se trouvant dans la puce. Les valeurs des coordonnées nommées *posi* sont supposées données.
- Les caractères définis dans le code sont, dans l'image, décalés d'une valeur *delta* qui est déterminée à partir du pixel de référence en calculant la somme des valeurs de ses composantes R,G,B. Autrement dit, si le pixel de référence est à la position (*x,y*) et si la valeur *x+delta* n'est pas supérieure à la largeur (*imx*) de l'image, le premier bloc sera à la position (*x+delta,y*), sinon le bloc se trouvera sur la ligne suivante et ceci jusqu'au 7e caractère défini dans le code de l'employé.

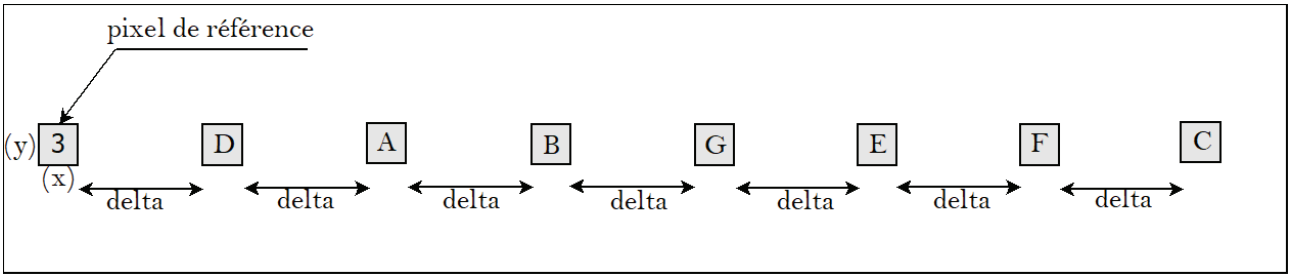


Figure 2 – Trame

Q14. Écrire une fonction *bin2()* qui, à partir d'un nombre entier, retourne un nombre en code binaire. Cette fonction convertit un nombre entier en une chaîne de chiffres binaires sans le *0b* devant, comme le ferait la fonction *bin()* de Python. Le résultat sera toujours écrit sous la forme d'une séquence d'au moins 3 digits de type *string*.

Remarque : pour l'implémentation de cette fonction, l'utilisation de la fonction *bin()* est possible. On rappelle que la fonction *bin()* convertit et retourne la chaîne de caractères en format binaire d'un entier donné. Par exemple, *bin(12)* retourne comme valeur la chaîne de caractères '0b1100'.

Signature de la fonction *bin2* :

```
bin2(int) -> str
```

Exemples :

```
>>> bin2(45)
'101101'
>>> bin2(1)
'001'
```

Q15. Écrire la fonction *num()* qui calcule à partir d'une chaîne en chiffres binaires, par exemple '010', sa valeur en décimal.
Si la chaîne est vide, la fonction retournera comme valeur -1.

Signature de la fonction *num* :

```
num(str) -> int
```

Exemples :

```
>>> num('101')
5
>>> num('')
-1
```

Partie VI - Décodage de l'information

Cette partie sera consacrée à l'implémentation de fonctions pour le décodage de l'information.

Remarque : vous pouvez utiliser les fonctions des questions précédentes même si elles n'ont pas été traitées.

On donne le code suivant :

```
1  from PIL import Image
2  import os
3  os.chdir("C:\photos")              # Positionnement sur le bon répertoire.
4  im = Image.open("a123.bmp")        # Lecture de l'image de l'employé dont la
5                                     # photo est a123.
6  imx, imy = im.size                 # La taille de l'image (largeur, hauteur).
7                                     # Résultat : (277, 250).
8  posi = (30,20)                     # Position du pixel (x,y).
9  px = im.getpixel(posi)             # Récupère la valeur au format RGB du pixel
10                                    # situé aux coordonnées (30,20).
11                                    # Résultat : (2, 57, 139).
12                                    # Les indices lignes et colonnes débutent à 0.
```

Lorsque l'employé passe sa carte sur le lecteur pour ouvrir une porte, les données de la puce et le code se trouvant dans la photo sont lus par le lecteur qui génère 2 valeurs envoyées au processus d'analyse (dont vous allez implémenter certaines fonctions). Ces 2 valeurs sont le pixel de référence (avec sa valeur *posi*) et le code associé à l'employé (un entier suivi de 7 caractères). Le pixel de référence est différent pour chacun des employés.
Les caractères du code dans l'image ne sont pas placés d'une manière consécutive mais séparés les uns des autres par une même valeur (*delta*) calculée à l'aide des valeurs `RGB` du pixel de référence (voir la **figure 2**) en réalisant la somme des 3 composantes `R`, `G` et `B`.

Q16. Écrire la fonction *lettre()* qui, à partir d'une chaîne de 3 bits, détermine le caractère. La codification des 7 caractères a été réalisée sur seulement 3 bits. On rappelle que le caractère 'A' est codé '001', le caractère 'B' est codé '010' et ainsi de suite jusqu'au caractère 'G' qui est codé '111', ceci en suivant l'ordre des 7 premières lettres de l'alphabet.

Signature de la fonction *lettre* :
```
lettre(str) -> str
```

Exemples :
```
>>> lettre('001')
'A'
>>> lettre('101')
'E'
```

Q17. Écrire la fonction *bpf* qui récupère les valeurs des bits de poids faible d'un pixel.
Cette fonction a 2 paramètres, l'image et la position du pixel. Elle retourne une chaîne de caractères composée de 3 bits.

Signature de la fonction *bpf* :
```
bpf(BmpImageFile, (int,int)) -> str
```

Exemple :
```
>>> bpf(im,posi)
'011'
```

Q18. Soit la fonction *valeur_delta()* dont le code est :

```
1 def valeur_delta(im,posi):
2       px = im.getpixel(posi)
3       inter = (px[0] + px[1] + px[2]) % (128-41) + 40
4       if inter % 2 == 1 :
5         return inter + 1
6       else :
7         return inter
```

Cette fonction a 2 paramètres : l'image et les coordonnées du point de référence (le pixel du point de référence).

Signature de la fonction *valeur_delta* :
```
valeur_delta(BmpImageFile, (int,int)) -> int
```

Indiquer ce que fait cette fonction en précisant les valeurs retournées. Le résultat sera décrit sous la forme d'un intervalle ou d'une union d'intervalles de nombres entiers.

Les deux fonctions suivantes *position_bloc()* et *num_site()* ne sont pas à implémenter.

La fonction *position_bloc()* détermine les coordonnées d'un bloc de la trame se trouvant à un indice compris entre 0 et 7, l'indice 0 représentant le pixel de référence. Le premier bloc des caractères du code se trouve à l'indice 1 et le dernier à l'indice 7. Rappelons que l'indice 0 (le pixel de référence) est le bloc constitué du numéro du site où l'employé travaille.
La fonction *position_bloc()* a 4 paramètres : la position initiale du pixel de référence, la valeur de l'intervalle, la largeur de l'image et l'indice du bloc. Elle retourne les coordonnées du pixel du bloc recherché.

Signature de la fonction *position_bloc* :
```
position_bloc((int,int),int,int,int) -> (int,int)
```

Exemple :
```
>>> position_bloc(posi,70,imx,1)
(100, 20)
```

La fonction *num_site()* retourne comme résultat le numéro du site où la personne travaille.
Cette fonction a 2 paramètres : l'image et la position du pixel de référence.

Signature de la fonction *num_site* :
```
num_site(BmpImageFile,(int,int)) -> int
```

Exemple :
```
>>> num_site(im, posi)
3
```

Q19. Écrire la fonction *lecture_lettres()* qui retourne la séquence des lettres codées dans l'image, c'est-à-dire les 7 lettres constituant une partie du code de l'employé.
Cette fonction a 2 paramètres, l'image et la position du pixel de référence. Elle retourne une chaîne de caractères.

Signature de la fonction *lecture_lettres* :
```
lecture_lettres(BmpImageFile,(int,int)) -> str
```

Exemple :
```
>>> lecture_lettres(im,posi)
'DABGEFC'
```

Q20. Indiquer ce que fait la fonction suivante :

```
1 def lecture_code(im, posi):
2     num = num_site(im, posi)
3     msg = lecture_lettres(im, posi)
4     return str(num) + msg
```

Annexe

Base de données “Personnel”

Le contenu des tables *Employes* et *ListeCategories* de la base de données *Personnel* est donnée ci-après.

Table *“Employes”*

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

Table *“ListeCategories”*

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

FIN

IMPRIMERIE NATIONALE – 19 1162 – D’après documents fournis

CONCOURS COMMUN INP

Numéro d'inscription

Numéro de table

Né(e) le

Nom :

Prénom :

Emplacement QR Code

Filière : TSI

Session : 2019

Épreuve de : Informatique

Consignes

- *Remplir soigneusement l'en-tête de chaque feuille avant de commencer à composer*
- *Rédiger avec un stylo non effaçable bleu ou noir*
- *Ne rien écrire dans les marges (gauche et droite)*
- *Numéroter chaque page (cadre en bas à droite)*
- *Placer les feuilles A3 ouvertes, dans le même sens et dans l'ordre*

TSIIN07

DOCUMENT RÉPONSE

SÉCURISATION D'UNE ENTREPRISE

Q1

Nombre maximal d'employés par site : .....

Nombre maximal de sites : .....

Réponse justifiée : .....

.....

B

NE RIEN ÉCRIRE DANS CE CADRE

Q2

1) Première séquence

Valeur de *s* avant la transformation : .....

Nom de la transformation : .....

Code Python :

Valeur de *s* après la transformation : .....

2) Deuxième séquence

Valeur de *s* avant la transformation : .....

Nom de la transformation : .....

Code Python :

Valeur de *s* après la transformation : .....

…/…

Q2 suite

3) Troisième séquence

Valeur de *s* avant la transformation : .....

Nom de la transformation : .....

Code Python :

Valeur de *s* après la transformation : .....

Coût de cette distance : .....

Q3

Après la ligne 14, matrice d :

Après la ligne 26, matrice d :

Distance de Levenshtein =

Q4

Complexité : .....

Q5

```
def code_bon_lvs(t):
```

Q6

```
def pourcentage_lvs(t):
```

CONCOURS COMMUN INP

Numéro d'inscription

Numéro de table

Né(e) le

Nom :

Prénom :

Emplacement QR Code

Filière : TSI

Session : 2019

Épreuve de : Informatique

Consignes

- *Remplir soigneusement l'en-tête de chaque feuille avant de commencer à composer*
- *Rédiger avec un stylo non effaçable bleu ou noir*
- *Ne rien écrire dans les marges (gauche et droite)*
- *Numéroter chaque page (cadre en bas à droite)*
- *Placer les feuilles A3 ouvertes, dans le même sens et dans l'ordre*

Q7

```
def liste_site(valeur):
```

NE RIEN ÉCRIRE DANS CE CADRE

Q8

Requête 1 =

Q9

Requête 2 =

Q10

Requête 3 =

Q11

Requête 4 =

Q12

Pixel bleu : .....

Pixel blanc : .....

Couleur du pixel au format RGB (10,10,10) : .....

Q13

Nombre de possibilités pour le code RGB : .....

.....

Q14

```
def bin2(n):
```

Q15

```
def num(ch):
```

CONCOURS COMMUN INP

Numéro d'inscription

Numéro de table

Né(e) le

Nom :

Prénom :

Emplacement QR Code

Filière : TSI

Session : 2019

Épreuve de : Informatique

Consignes

- *Remplir soigneusement l'en-tête de chaque feuille avant de commencer à composer*
- *Rédiger avec un stylo non effaçable bleu ou noir*
- *Ne rien écrire dans les marges (gauche et droite)*
- *Numéroter chaque page (cadre en bas à droite)*
- *Placer les feuilles A3 ouvertes, dans le même sens et dans l'ordre*

Q16

```
def lettre(ch_bin):
```

Q17

```
def bpf(im,posi):
```

NE RIEN ÉCRIRE DANS CE CADRE

Q18

Explication : .....

.....

.....

.....

.....

.....

Résultat : .....

Q19

```
def lecture_lettres(im,posi):
```

Q20

Réponse : .....

.....

FIN

Travailler les chapitres de ce sujet

Prepa Help suit le programme de TSI chapitre par chapitre : cours détaillé, méthodes, pièges, exercices et problèmes corrigés pas à pas, tirés des concours. L'essai gratuit donne accès aux chapitres offerts de ta filière, sans carte bancaire.

Chapitres liés à ce sujet

- Chaînes de caractères et algorithmes sur les textes
- Listes et algorithmes de recherche (séquentielle, dichotomique)
- Spécifier, tester, prouver : terminaison et correction
- Bases de Python : expressions, types, instructions, fonctions
- Architecture de l'ordinateur et représentation des nombres
- Images : matrices de pixels, opérations, convolution

Ouvrir l'essai gratuit

Scanne le code ou clique sur le bouton.

prepahelp.com/#essai

Une question sur le concours ou sur ta filière : **écrire sur WhatsApp**

Les autres années et épreuves : prepahelp.com/annales/ccinp/tsi