Métropole - septembre 2025 - sujet 2

Exercice 1 (Programmation Python et cryptographie - 6 points)
Le chiffrement Playfair, popularisé par Lord Playfair et utilisé par l'armée britannique durant les guerres du XXème siècle, est basé sur le chiffrement de paires de lettres (appelées *digrammes*).

Partie A : la clef de chiffrement
Ce chiffrement utilise un tableau de 5×5 lettres contenant un mot clef. On remplit le tableau avec les lettres du mot clef (sans doublons), puis on le complète avec les lettres restantes de l'alphabet (sans la lettre W) dans leur ordre alphabétique. Une lettre n'apparait qu'une seule fois dans le tableau.

Par exemple, si on choisit comme clef le mot `PLAYFAIR`, le carré de chiffrement obtenu est le suivant :

P	L	A	Y	F
I	R	B	C	D
E	G	H	J	K
M	N	O	Q	S
T	U	V	X	Z

FIGURE 1 – Carré de chiffrement obtenu avec le mot clef `PLAYFAIR`

On commence par les lettres de la clef (cases blanches) sans les doublons (ici le `A`) puis on complète le tableau (cases grisées) avec les lettres restantes de l'alphabet, dans l'ordre alphabétique.

1. Donner le carré de chiffrement si la clef est `EPREUVEDENSI`.

On donne ci-dessous le code Python de la fonction `creer_liste_clef` qui prend en paramètre la clef de chiffrement et renvoie une liste contenant 25 lettres ordonnées de la façon suivante : d'abord les lettres de la clef choisie (sans doublon) puis les lettres de l'alphabet restantes (classées par ordre alphabétique).

```
1  def creer_liste_clef(clef):
2      """
3      hypothèse : la clef ne contient pas la lettre W
4      """
5      deja_utilises = []
6      # alphabet sans la lettre W:
7      alphabet = 'ABCDEFGHIJKLMNOPQRSTUVXYZ'
8      for i in range(len(clef)):
9          if not (clef[i] in deja_utilises):
10             deja_utilises.append(clef[i])
11     for lettre in alphabet:
12         if not lettre in deja_utilises:
13             deja_utilises.append(lettre)
14     return deja_utilises
```

Exemple :

```
>>> creer_liste_clef('PLAYFAIR')
['P', 'L', 'A', 'Y', 'F', 'I', 'R', 'B', 'C', 'D', 'E', 'G', 'H', 'J', 'K', 'M', 'N',
'O', 'Q', 'S', 'T', 'U', 'V', 'X', 'Z']
```

2. Donner l'assertion à insérer en début de la fonction `creer_liste_clef` afin de s'assurer que l'hypothèse sur la clef soit respectée.

On donne ci-dessous le code incomplet de la fonction `creer_carre` qui prend en paramètre la liste créée par la fonction `creer_liste_cle` et renvoie le carré de chiffrement.

```
1 def creer_carre(liste_clef):
2     carre = [[0 for i in range(5)] for j in range(5)]
3     for i in range(25):
4         carre[...][...] = liste_clef[i]
5     return carre
```

Exemple :

```
>>> creer_carre(creer_liste_clef('PLAYFAIR'))
[['P', 'L', 'A', 'Y', 'F'], ['I', 'R', 'B', 'C', 'D'], ['E', 'G', 'H', 'J', 'K'],
['M', 'N', 'O', 'Q', 'S'], ['T', 'U', 'V', 'X', 'Z']]
```

3. Recopier et compléter la ligne 4 du code de cette fonction `creer_carre`, en utilisant les opérateurs `%` (reste de la division entière) et `//` (division entière).

Partie B : chiffrer un message

Le chiffrement d'un message se fait en deux étapes :

- étape 1 : on découpe le message en digrammes (paires de lettres) ;
- étape 2 : on chiffre chacun des digrammes avec le carré de chiffrement.

Pour découper le message en digrammes, on prend les lettres deux par deux en tenant compte de deux cas particuliers :

- si les deux lettres sont identiques, on ajoute un `X` après la première lettre et on poursuit le découpage deux à deux à partir de la deuxième lettre ;
- s'il ne reste qu'une seule lettre, on forme une dernière paire en lui ajoutant la lettre un `X`.

Par exemple, le découpage de `BACCALAUREAT` donnera `BA`, `CX`, `CA`, `LA`, `UR`, `EA`, `TX`.

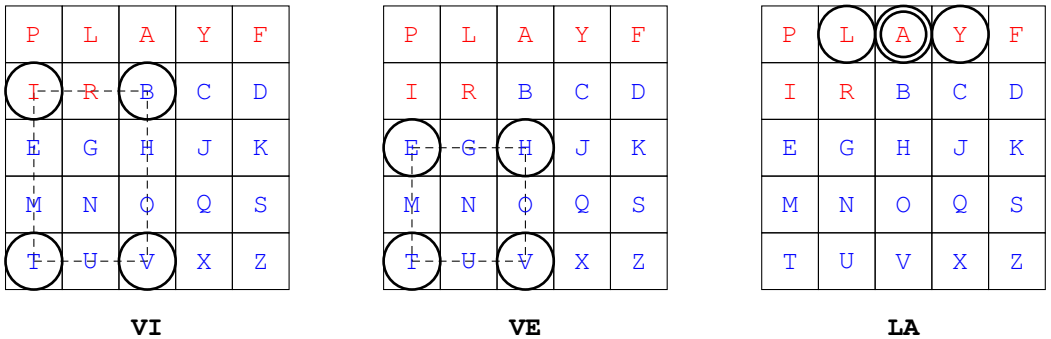
Le chiffrement d'un message se fait ensuite en chiffrant chaque digramme (paire de lettres), de la manière suivante :

- si les lettres du digramme se trouvent sur la même ligne du carré de chiffrement, il faut les remplacer par celles se trouvant immédiatement à leur droite (en bouclant sur la gauche si le bord est atteint) ;
- si les lettres apparaissent sur la même colonne du carré de chiffrement, les remplacer par celles qui sont juste en dessous (en bouclant par le haut si le bas de la table est atteint) ;
- sinon, remplacer les lettres par celles se trouvant sur la même ligne du carré de chiffrement, mais dans le coin opposé du rectangle défini par la paire originale.

Par exemple, si le message est `VIVELANSI`, les digrammes sont : `VI`, `VE`, `LA`, `NS`, `IX` et leurs codages avec la clef `PLAYFAIR` sont :

digramme	VI	VE	LA	NS	IX
chiffré	TB	TH	AY	OM	CT

comme le montre la figure 2 ci-dessous :



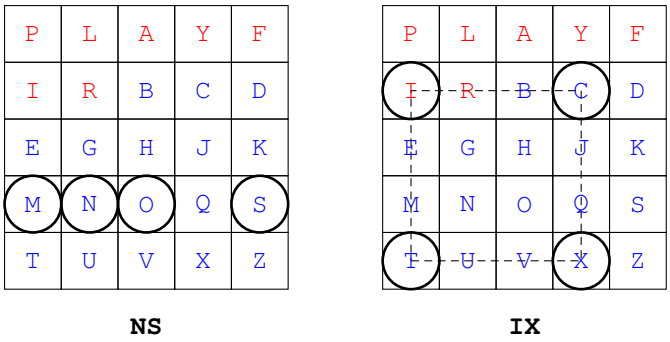


FIGURE 2 – Chiffrement de quelques digrammes

On donne ci-dessous le code incomplet de la fonction `couper_en_digrammes` qui prend en paramètre une chaîne de caractères et renvoie la liste des digrammes la constituant :

```
1  def couper_en_digrammes(message):
2      digrammes = []
3      i = 0
4      while i < len(message) - 1:
5          if message[i] == message[i+1]:
6              digrammes.append(message[i] + 'X')
7              i = i + 1
8          else:
9              ...
10             i = i + 2
11     if i == len(message) - 1: #il reste une lettre isolée
12         digrammes.append(message[i] + 'X')
13     return digrammes
```

4. Donner le code de la ligne 9 manquante de cette fonction `couper_en_digrammes`.
5. Donner le résultat de l'appel `couper_en_digrammes('BONJOUR')`.
6. Donner le chiffrement du message `BONJOUR` avec le carré de chiffrement `PLAYFAIR` donné en Figure 1.
7. Donner le code Python de la fonction `ligne_colonne` qui prend en paramètres une lettre et le carré de chiffrement créé par la fonction `creer_carre`, et qui renvoie les coordonnées de la lettre dans le carré de chiffrement.
   Exemple (avec le carré de chiffrement de la Figure 1) :

```
>>> ligne_colonne('A', carre)
(0, 2)
>>> ligne_colonne('N', carre)
(3, 1)
```

8. Donner le code Python de la fonction `sur_la_meme_ligne` qui prend en paramètres un digramme et le carré de chiffrement créé par la fonction `creer_carre`, et qui renvoie `True` si les deux lettres du digramme sont sur la même ligne, ou `False` sinon.
   Exemple (avec le carré de chiffrement de la Figure 1) :

```
>>> sur_la_meme_ligne('RT', carre)
False
>>> sur_la_meme_ligne('PL', carre)
True
```

On dispose pour la suite de la fonction `sur_la_meme_colonne`, similaire à `sur_la_meme_ligne` mais en colonne.

Voici le code incomplet de la fonction `chiffrer_digramme` qui prend en paramètres le carré de chiffrement et un digramme, et qui renvoie le digramme chiffré correspondant :

```
1  def chiffrer_digramme(digramme, carre):
2      lettre1 = digramme[0]
3      lettre2 = digramme[1]
4      i1, j1 = ligne_colonne(lettre1, carre)
5      i2, j2 = ligne_colonne(lettre2, carre)
6      if sur_la_meme_ligne(digramme, carre):
7          digramme_chiffre = carre[i1][(j1 + 1)%5] + carre[i2][(j2 + 1)%5]
8      elif sur_la_meme_colonne(digramme, carre):
9          digramme_chiffre = ...
10     else:
11         digramme_chiffre = ...
12     return digramme_chiffre
```

9. Donner le code complet des lignes 9 et 11 de cette fonction `chiffrer_digramme`.

10. Ecrire le code python de la fonction `chiffrer_playfair` qui prend en paramètres deux chaînes de caractères `message` et `clef` correspondant au message à chiffrer et au mot-clef choisi, et qui renvoie le message chiffré, en utilisant les fonctions déjà écrites précédemment.
    Exemple :

```
>>> chiffrer_playfair('VIVELANSI', 'PLAYFAIR')
'TBTHAYOMCT'
```

Exercice 2 (Systèmes d'exploitation et POO - 6 points)
Partie A

1. Écrire, dans un système Linux, la commande masquée en noir qui a permis d'obtenir le résultat de la Figure 1.

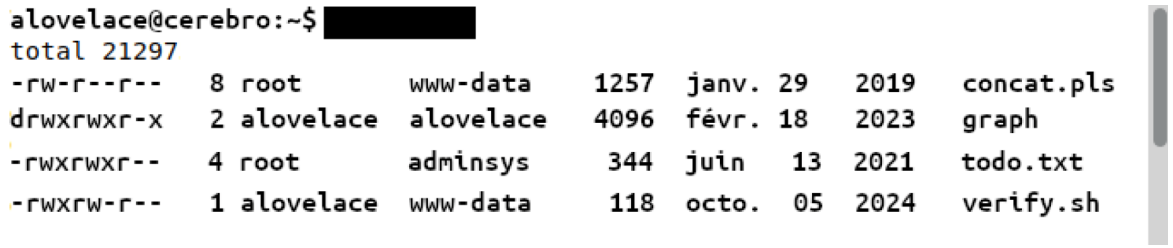


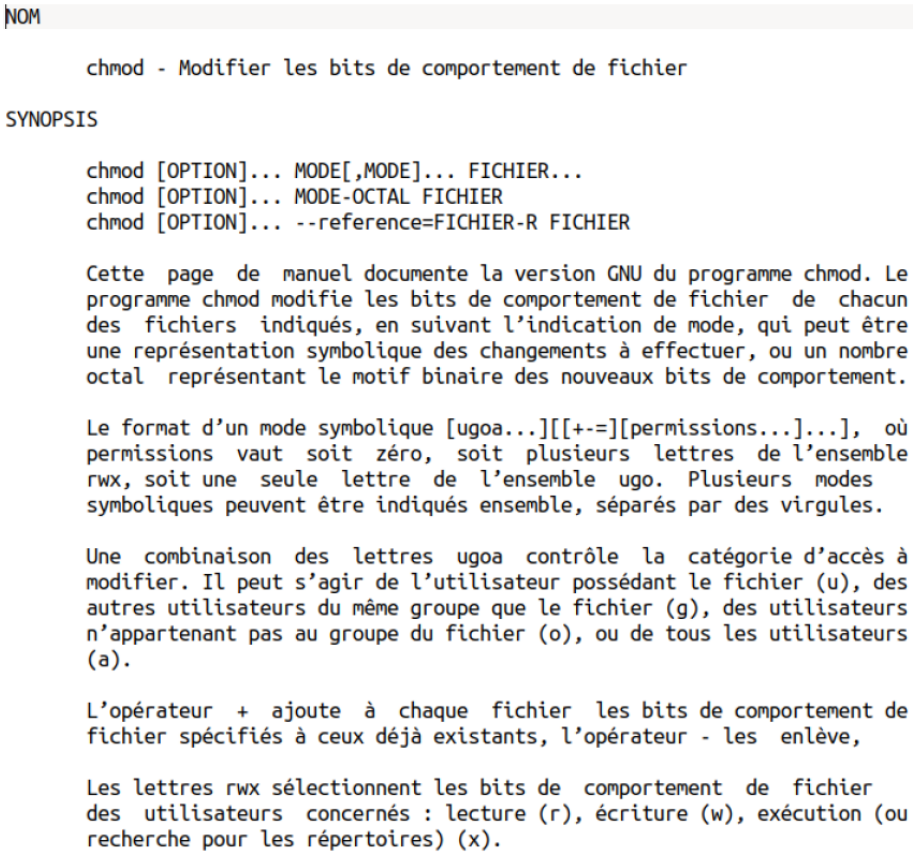
FIGURE 1 – Capture d'écran d'un terminal Linux

Dans la Figure 1, les permissions des fichiers apparaissent au début de chaque ligne et sont composées de 10 caractères. En lisant de la gauche vers la droite :

- le premier caractère indique la nature du fichier : – pour un fichier classique, `d` pour un répertoire et `l` pour un lien ;
- ensuite, on trouve trois groupes de 3 caractères chacun, indiquant si le fichier (ou répertoire) est autorisé en lecture `r`, écriture `w` ou exécution `x`. Si la permission n'est pas accordée, le caractère est remplacé par –. Les 3 groupes correspondent, dans cet ordre, aux droits du propriétaire (user), du groupe (group) puis du reste des utilisateurs (other).

2. Décrire les permissions que les trois types d'utilisateurs (user, group ou other) ont sur le fichier `verify.sh` d'après la Figure 1.
3. D'après la Figure 1, écrire la commande que l'utilisateur, propriétaire du fichier `concat.pls`, doit saisir pour supprimer ce fichier.

On s'intéresse à la façon de modifier les permissions d'un fichier avec la commande `chmod`. Pour cela on saisit dans un terminal la commande `man chmod`. La Figure 2 ci-après reproduit l'affichage obtenu.



NOM

chmod - Modifier les bits de comportement de fichier

SYNOPSIS

chmod [OPTION]... MODE[,MODE]... FICHIER...
chmod [OPTION]... MODE-OCTAL FICHIER
chmod [OPTION]... --reference=FICHIER-R FICHIER

Cette page de manuel documente la version GNU du programme chmod. Le programme chmod modifie les bits de comportement de fichier de chacun des fichiers indiqués, en suivant l'indication de mode, qui peut être une représentation symbolique des changements à effectuer, ou un nombre octal représentant le motif binaire des nouveaux bits de comportement.

Le format d'un mode symbolique [ugoa...][[+-=][permissions...]...], où permissions vaut soit zéro, soit plusieurs lettres de l'ensemble rwx, soit une seule lettre de l'ensemble ugo. Plusieurs modes symboliques peuvent être indiqués ensemble, séparés par des virgules.

Une combinaison des lettres ugoa contrôle la catégorie d'accès à modifier. Il peut s'agir de l'utilisateur possédant le fichier (u), des autres utilisateurs du même groupe que le fichier (g), des utilisateurs n'appartenant pas au groupe du fichier (o), ou de tous les utilisateurs (a).

L'opérateur + ajoute à chaque fichier les bits de comportement de fichier spécifiés à ceux déjà existants, l'opérateur - les enlève,

Les lettres rwx sélectionnent les bits de comportement de fichier des utilisateurs concernés : lecture (r), écriture (w), exécution (ou recherche pour les répertoires) (x).

FIGURE 2 – Extrait de la page de manuel de `chmod`

4. Avec les éléments de la Figure 1 et à la lecture de la documentation Figure 2, expliquer quel sera l'effet de l'instruction `chmod g-wx todo.txt`

Partie B

Dans la suite de la documentation de la commande `chmod`, on peut lire l'extrait suivant :

```
l'indication de mode, qui peut être une représentation
symbolique des changements à effectuer, ou un nombre octal
représentant le motif binaire des nouveaux bits de
comportement.
```

Cela signifie que l'on peut indiquer les permissions soit sous la forme d'une chaîne de 9 caractères composée de `-`, `r`, `w` et `x`, appelée *représentation symbolique*, soit sous la forme d'une valeur numérique `abc` composée de trois chiffres `a`, `b` et `c` compris entre 0 et 7 appelée *représentation octale*.

Le but de cette partie est d'écrire des fonctions Python permettant de passer d'une représentation symbolique à une représentation octale des permissions.

On considère une chaîne de trois caractères dont :

- le premier caractère est `'-'` ou `'r'` ;
- le second caractère est `'-'` ou `'w'` ;
- et le dernier est `'-'` ou `'x'`.

On souhaite déterminer le motif binaire de 3 bits correspondant à cette chaîne. Pour cela, on applique les règles suivantes :

- si le premier caractère est `'r'`, le premier bit (celui le plus à gauche), vaut 1, sinon il vaut 0 ;
- si le second caractère est `'w'`, le second bit vaut 1, sinon il vaut 0 ;
- Si le dernier caractère est `'x'`, le dernier bit vaut 1, sinon il vaut 0.

5. Déterminer l'écriture en base 10 du motif binaire de 3 bits correspondant à la chaîne `'rw-`

La représentation octale associée à une représentation symbolique est obtenue en appliquant ce qui précède trois fois :

- les trois caractères correspondant à user donnent le chiffre de gauche ;
- les trois suivants (correspondant à group) donnent le chiffre du milieu ;
- les trois derniers (correspondant à other) donnent le chiffre de droite.

Par exemple, la représentation octale associée à la représentation symbolique `rwxr-xr--` est `754`.

6. Écrire le code d'une fonction Python `bin_to_oct` qui prend en paramètre une chaîne de 3 caractères composée uniquement des caractères `'0'` et `'1'` et renvoie la représentation octale correspondant à ce motif binaire (le bit de poids fort étant encore à gauche).
   Exemples :

```
>>> bin_to_oct('101')
5
>>> bin_to_oct('011')
3
```

7. On considère la fonction Python `mystere` qui prend en paramètre une chaîne de 9 caractères et qui renvoie une chaîne de caractères.

```
1 def mystere(chaine):
2     resultat = ''
3     for c in chaine:
4         if c == '-':
5             resultat = resultat + '0'
6         else:
7             resultat = resultat + '1'
8     return resultat
```

   Déterminer ce que renvoie `mystere('rw-r-xr--')`

8. On considère dans cette question qu'on dispose d'une fonction Python `symb_to_bin` qui prend en paramètre une chaîne de 9 caractères correspondant à l'écriture symbolique d'une permission et qui renvoie une chaîne de caractères correspondant au motif binaire de cette permission.
   Recopier et compléter le code de la fonction Python `symb_to_oct` donné ci-dessous, qui prend en paramètre une représentation symbolique sous la forme d'une chaîne de caractères, et qui renvoie la représentation octale correspondante sous la forme d'une chaîne de caractères. Il est possible de répondre en une ou plusieurs lignes de code.

```
def symb_to_oct(chaine):
    repr_bin = symb_to_bin(chaine)
    # les 3 premiers bits correspondent au propriétaire
    user = repr_bin[0] + repr_bin[1] + repr_bin[2]
    # les 3 suivants au groupe
    group = repr_bin[3] + repr_bin[4] + repr_bin[5]
    # et les 3 derniers aux autres
    other = repr_bin[6] + repr_bin[7] + repr_bin[8]
    ...
```

Exemples :

```
>>> symb_to_oct('rwxrw-r--')
'764'
>>> symb_to_oct('rw-r--r--')
'644'
```

Partie C

Dans cette partie, on souhaite écrire un gestionnaire de fichiers Linux. Pour cela, on modélise les fichiers en utilisant la programmation orientée objet. On dispose de la classe `Fichier` ci-dessous.

```
class Fichier:
    def __init__(self, nom, poids, proprio, groupe, permission):
        self.nom = nom
        self.poids = poids # taille en octets
        self.proprietaire = proprio # nom du propriétaire
        self.groupe = groupe # nom du groupe
        self.permission = permission # représentation symbolique
```

9. Instancier l'objet `mon_fichier` de la classe `Fichier` représentant un fichier nommé `concat.pls`, dont le propriétaire est `root`, le groupe est `www-data`, le poids `1257` octets et dont les permissions sont données par la représentation symbolique `'rw-r--r--'`

10. Écrire une méthode `chown` de la classe `Fichier` qui prend en paramètre une chaîne de caractères qui correspond au nom du nouveau propriétaire et qui modifie le propriétaire du fichier.

11. Écrire une fonction `get_executable` qui prend en paramètres une liste d'objets de la classe `Fichier` ainsi qu'un nom de propriétaire et qui renvoie la liste des fichiers qui appartiennent à ce propriétaire et qui sont exécutables par ce propriétaire.

Exercice 3 (Langage SQL, graphes et programmation générale - 8 points)
Un supermarché utilise une base de données qui contient des informations sur les produits, les fournisseurs, les commandes passées et leurs détails. Le modèle relationnel de cette base est donné par le schéma ci-dessous :

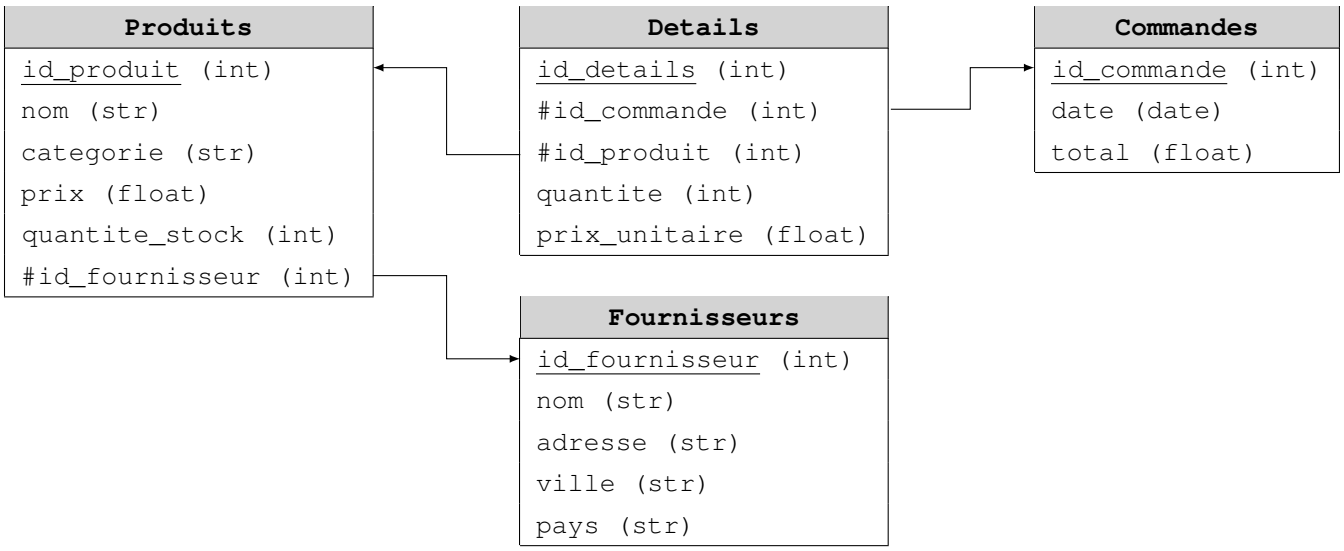


FIGURE 1 – Schéma relationnel de cette base

Dans ce schéma, les clés primaires sont soulignées et les clés étrangères sont précédées du symbole #. Le type de chaque attribut est indiqué entre parenthèses.

On considère l'extrait de la base de données ci-dessous :

- Table `Commandes` :

id_commande	date	total
1	03/06/2025	176.00
2	08/12/2024	1150.00
3	21/04/2025	155.00

- Table `Produits` :

id_produit	nom	categorie	prix	quantite_stock	id_fournisseur
1	Yaourts blanc x 4	Alimentaire	2.80	50	2
2	Lait	Alimentaire	1.20	200	2
3	Pain	Alimentaire	1.50	100	4
4	Harry Potter 1	Livre	15.00	20	3
5	Jeu d'échecs	Jeux	40.00	30	3
6	T-shirt taille M	Vêtement	10.00	80	1
7	Jeans taille M	Vêtement	25.00	60	5

- Table `Fournisseurs` :

id_fournisseur	nom	adresse	ville	pays
1	Moda e stile	Via della Moda, 45	Milano	Italie
2	Laiteries Unies	22 Avenue des Vaches	Lisieux	France
3	Livres en Folie	56 Boulevard des Livres	Toulouse	France
4	Boulangerie du Coin	34 Rue du Pain	Nantes	France
5	Estilo Español	Calle de la Moda, 123	Madrid	Espagne

⋆ Table `Details` :

id_details	id_commande	id_produit	quantite	prix_unitaire
1	1	1	20	2.80
2	1	2	100	1.20
3	2	6	40	10.00
4	2	7	30	25.00
5	3	5	2	40.00
6	3	4	5	15.00

L'énoncé de cet exercice utilise tout ou une partie des mots clefs du langage SQL suivants : `SELECT`, `DISTINCT`, `FROM`, `WHERE`, `JOIN ... ON`, `UPDATE ... SET`, `DELETE`, `INSERT INTO ... VALUES`.

Avant de mettre en vente un nouveau produit, il faut le créer dans la base.

1. Écrire une requête SQL permettant d'ajouter le produit *croissant* référencé dans la base sous le numéro 10. Il est vendu au prix unitaire de 0,90€. Le magasin se fournit, pour ce produit, auprès de *Boulangerie du Coin*.

Le fournisseur *Livres en Folie* a changé d'entrepôt. Il se trouve maintenant au 78 Rue des Jeux à Elbeuf (France).

2. Écrire la requête SQL permettant de mettre à jour la base de données.
3. Décrire le résultat obtenu avec la requête SQL ci-dessous :

```sql
SELECT nom FROM Produits WHERE categorie = 'Alimentaire';
```

4. Écrire une requête SQL permettant d'afficher les détails des commandes passées ayant un total de commande supérieur ou égal à 1 000€.
5. Écrire une requête SQL permettant d'afficher le nom des fournisseurs basés en Espagne ou en Italie.
6. Écrire une requête SQL qui permet d'afficher le nom de tous les fournisseurs qui ont vendu des produits alimentaires.
7. Écrire une requête SQL permettant d'afficher le numéro et la date des commandes ainsi que le nom des fournisseurs où des produits de catégories *Vêtement* ont été commandés.

Un fournisseur, dont l'entrepôt est situé à Toulouse, approvisionne différents supermarchés à travers la France, notamment dans les villes de Bordeaux, Calais, Lyon, Marseille, Nantes, Paris et Strasbourg.

On utilise un graphe dont les sommets sont les initiales des villes où se situent les supermarchés et l'entrepôt du fournisseur. Les arêtes sont pondérées avec les distances en kilomètres entre les villes.

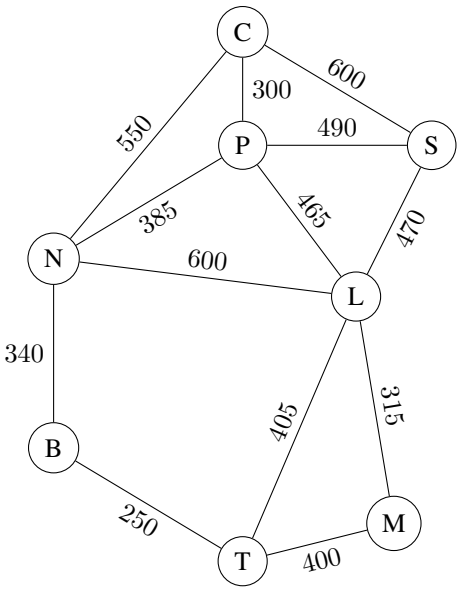


FIGURE 2 – Graphe représentant le réseau routier

8. Déterminer le plus court chemin (en termes de distance) entre Toulouse et Calais.
9. Écrire la liste des sommets dans l'ordre d'un parcours en profondeur à partir de Calais (les sommets sont toujours pris dans l'ordre alphabétique s'il y a un choix à faire).
10. Écrire la liste des sommets dans l'ordre d'un parcours en largeur à partir de Calais (les sommets sont toujours pris dans l'ordre alphabétique s'il y a un choix à faire).

Pour implémenter ce graphe, on utilise le dictionnaire en Python ci-dessous :

```
graphe = {'Paris': {'Strasbourg': 490, 'Lyon': 465, 'Nantes': 385, 'Calais': 300},
          'Strasbourg': {'Paris': 490, 'Lyon': 470, 'Calais': 600},
          'Lyon': {'Paris': 465, 'Strasbourg': 470, 'Toulouse': 405, 'Nantes': 600},
          'Nantes': {'Paris': 385, 'Lyon': 600, 'Bordeaux': 340, 'Calais': 550},
          'Calais': {'Paris': 300, 'Strasbourg': 600, 'Nantes': 550},
          'Toulouse': {'Lyon': 405, 'Bordeaux': 250, 'Marseille': 400},
          'Marseille': {'Toulouse': 400},
          'Bordeaux': {'Nantes': 340, 'Toulouse': 250}}
```

Dans cette implémentation, il manque la route entre Lyon et Marseille.

11. Écrire les instructions permettant d'ajouter dans le dictionnaire `graphe` la route entre les villes de Lyon et Marseille sachant que la distance les séparant est de 315 km.
12. Écrire une fonction `distance` qui prend en paramètres le dictionnaire `graphe` et deux villes (de type str) et qui renvoie la distance entre ces deux villes si elles sont adjacentes, ou `None` sinon.

Pour déterminer le chemin entre deux villes quelconques (s'il en existe un) dans le dictionnaire `graphe`, on utilise la fonction `trouver_chemin(graphe, ville_depart, ville_destination)` qui renvoie la liste des villes parcourues. On suppose que cette fonction est codée en Python.

13. Écrire une fonction `distance_totale` qui prend en paramètre le dictionnaire `graphe` et deux villes (de type str) et qui renvoie la distance entre ces deux villes s'il existe un chemin entre elles sinon elle retourne `None`.

On décide désormais de prendre en compte le temps nécessaire pour parcourir les distances entre les villes. Ainsi les arêtes du graphe sont pondérées à l'aide d'une liste contenant la distance (en km arrondie à l'unité) et la durée (en heure arrondie à deux décimales) nécessaire pour effectuer le trajet entre les deux villes. Par exemple :

```
>>> graphe['Paris']
{'Strasbourg': [490, 6.37], 'Lyon': [465, 5.58], 'Nantes': [385, 3.47],
'Calais': [300, 3.3]}
```

14. A partir de l'exemple précédent, déterminer la valeur de `graphe['Paris']['Nantes'][1]`

La fonction `ratio_duree_distance` ci-dessous prend en paramètre le dictionnaire `graphe`. Elle permet de calculer le ratio durée/distance pour toutes les arêtes du graphe et de l'ajouter à la pondération de chaque arête :

```
1 def ratio_duree_distance(graphe):
2     for ville, connexions in ...:
3         for destination, valeurs in ...:
4             distance, duree = ...
5             ratio = ...
6             graphe[ville][destination].append(...)
7     return graphe
```

Grâce à cette fonction, on obtient la mise à jour du dictionnaire `graphe` :

```
graphe = {'Paris': {'Strasbourg': [490, 6.37, 0.013],
                    'Lyon': [465, 5.58, 0.012],
                    'Nantes': [385, 3.47, 0.009],
                    'Calais': [300, 3.3, 0.011]},
          'Strasbourg': {'Paris': [490, 6.37, 0.013],
                         'Lyon': [470, 4.23, 0.009],
                         'Calais': [600, 9.0, 0.015]},
          'Lyon': {'Paris': [465, 5.58, 0.012],
                   'Strasbourg': [470, 4.23, 0.009],
                   'Toulouse': [405, 4.86, 0.012],
                   'Marseille': [315, 2.84, 0.009],
                   'Nantes': [600, 7.2, 0.012]},
          'Nantes': {'Paris': [385, 3.47, 0.009],
                     'Lyon': [600, 7.2, 0.012],
                     'Bordeaux': [340, 4.08, 0.012],
                     'Calais': [550, 8.25, 0.015]},
          'Calais': {'Paris': [300, 3.3, 0.011],
                     'Strasbourg': [600, 9.0, 0.015],
                     'Nantes': [550, 8.25, 0.015]},
          'Toulouse': {'Lyon': [405, 4.86, 0.012],
                       'Bordeaux': [250, 2.5, 0.010],
                       'Marseille': [400, 6.0, 0.015]},
          'Marseille': {'Lyon': [315, 2.84, 0.009],
                        'Toulouse': [400, 6.0, 0.015]},
          'Bordeaux': {'Nantes': [340, 4.08, 0.012],
                       'Toulouse': [250, 2.5, 0.010]}}
```

15. Recopier et compléter la fonction `ratio_duree_distance`.

Un élève souhaite utiliser ChatGPT pour trouver un algorithme qui détermine le chemin à privilégier entre deux villes. Il fournit son script Python contenant le dictionnaire `graphe` (et son descriptif) et les fonctions précédentes. Il écrit le prompt suivant :

```
Écris un algorithme, en langage naturel, pour trouver un chemin entre deux villes en
minimisant le ratio durée/distance où à chaque étape, on choisira l'arête avec le ratio
le plus faible.
```

16. Déterminer le nom que l'on donne à un algorithme qui construit une solution étape par étape, comme celui demandé par l'élève.
17. Déterminer le chemin trouvé grâce à cet algorithme entre Toulouse et Calais.