BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2025

NUMÉRIQUE ET SCIENCES INFORMATIQUES

JOUR 1

Durée de l'épreuve : **3 heures 30**

L'usage de la calculatrice n'est pas autorisé.

Dès que ce sujet vous est remis, assurez-vous qu'il est complet.

Ce sujet comporte 19 pages numérotées de 1/19 à 19/19.

Le sujet est composé de trois exercices indépendants.

Le candidat traite les trois exercices.

Exercice 1 (6 points)

Cet exercice porte sur les protocoles réseaux, l'algorithmique et la POO.

Partie A

Un cœur de réseau est constitué d'un maillage dans lequel les routeurs R_1 à R_6 sont reliés par des liaisons physiques dont le débit en Mbps (Méga-bits par seconde) est indiqué sur la figure suivante.

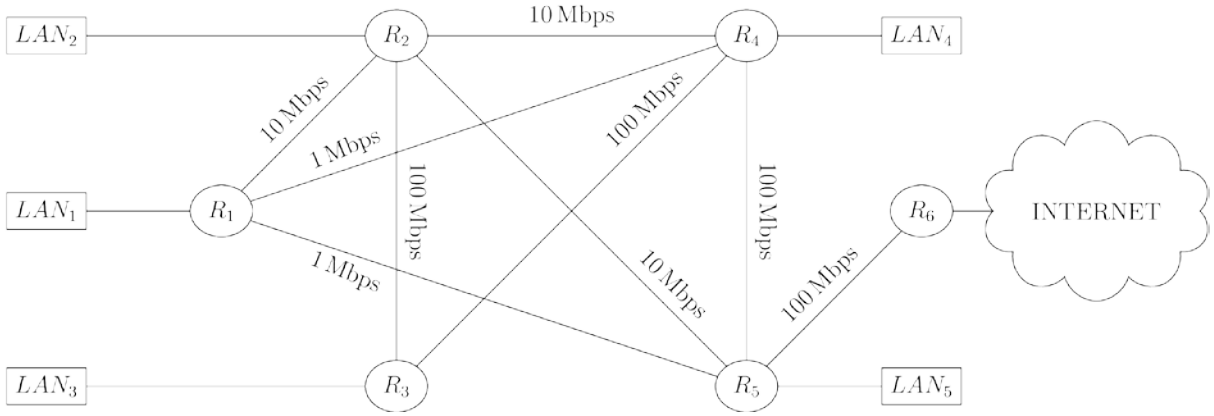

Figure 1. Réseau

Derrière chaque routeur R_1 à R_5, un switch distribue un réseau local LAN_1 à LAN_5.

Les protocoles utilisés sont le protocole RIP et le protocole OSPF, qui minimisent respectivement le nombre de routeurs traversés et la somme des coûts des liaisons physiques empruntées.

1. Recopier et compléter la table de routage de R_1 sachant que le protocole de routage RIP est utilisé.

   La destination est le routeur à atteindre, le prochain saut est le premier routeur traversé et la distance est le nombre de routeurs traversés.

Table de routage R_1		
destination	prochain saut	distance
R_2	R_2	0
R_3	R_2	1
R_4		
R_5		
R_6		

2. Un utilisateur du réseau local LAN_1 interroge, via son navigateur web, un serveur d'Internet. Détailler, suivant le protocole RIP, la route suivie dans le maillage par la requête à destination de ce serveur.

Le coût d'une liaison est donné par la relation coût = $\frac{10^2}{d}$, où d est le débit de la liaison en Mbps.

Exemple : le coût de la liaison entre R_1 et R_2 est $\frac{10^2}{10} = 10$.

Le tableau suivant donne le coût de chacune des liaisons physiques entre R_1 et les autres routeurs du graphe qui lui sont connectés.

Coût des liaisons depuis R_1			
routeur	R_2	R_4	R_5
coût	10	100	100

3. Recopier et compléter la table de routage de R_1 sachant que le protocole de routage OSPF est utilisé.

   La distance est la somme totale des coûts des liaisons physiques empruntées pour atteindre la destination.

Table de routage R_1		
destination	prochain saut	distance
R_2	R_2	10
R_3		
R_4		
R_5		
R_6		

Un utilisateur du réseau local LAN_1 interroge, via son navigateur web, un serveur d'Internet.

4. Donner, selon le protocole OSPF, la route suivie dans le maillage par la requête à destination de ce serveur.

Le protocole de routage OSPF est toujours utilisé et le routeur R_2 tombe en panne.

5. Donner la nouvelle route suivie dans le maillage par une requête partant du réseau LAN_1 à destination d'Internet et en donner la nouvelle valeur de la distance.

Partie B

Le réseau LAN_1 est supposé disposer d'un serveur DHCP (Dynamic Host Control Protocol) gérant l'attribution d'adresses IPv4. Il est rappelé, ici, qu'une adresse IPv4 est une séquence de 4 octets, généralement représentée par les 4 entiers correspondants en écriture décimale, séparés par des points (notation décimale pointée). Un réseau est caractérisé par des machines dont les adresses ont en commun les n mêmes premiers bits. La notation CIDR du réseau a la forme "adresse réseau / n". Appliqué à une adresse du réseau, le masque permet de calculer le préfixe réseau commun. Il est constitué de 4 octets consécutifs, dont (de gauche à droite) les n premiers bits sont égaux à `1` et les suivants à `0`. On obtient l'adresse du réseau en effectuant un `ET` logique, bit à bit, entre chaque octet composant l'adresse d'une machine et l'octet qui lui correspond dans le masque.

Exemple de `ET` :

```
  l'entier 192 s'écrit : 1 1 0 0 0 0 0 0
  l'entier 255 s'écrit : 1 1 1 1 1 1 1 1
-------------------------------------------------------
ET logique, bit à bit  : 1 1 0 0 0 0 0 0 ( soit l'entier 192 )
```

6. Recopier et compléter le tableau suivant correspondant à une machine d'un réseau d'adresse `192.168.1.100` et de masque `255.192.0.0`. Ce tableau détermine l'adresse du réseau (en binaire puis en décimale pointée).

Calcul d'une adresse de réseau				
machine (binaire)	11000000	10101000		
masque (binaire)	11111111	11000000	00000000	00000000
réseau (binaire)	11000000			
réseau (déc. pointée)	`192`			

On obtient le complémentaire d'un octet en échangeant respectivement chacun des 0 et des 1 qui le composent, par un 1 ou un 0.

Par exemple, le complémentaire de `1 1 0 0 1 0 1 0` est `0 0 1 1 0 1 0 1`.

Par un procédé analogue à celui de la question précédente, l'adresse de broadcast d'un réseau est obtenue en effectuant un `OU` logique bit à bit entre chaque octet composant l'adresse réseau et le complémentaire de l'octet correspondant dans l'écriture binaire du masque.

Exemple de OU :

```
          12 s'écrit : 0 0 0 0 1 1 0 0
           9 s'écrit : 0 0 0 0 1 0 0 1
-------------------------------------------------------------
OU logique, bit à bit : 0 0 0 0 1 1 0 1 ( soit l'entier 13 )
```

7. Recopier et compléter le tableau ci-après pour déterminer l'adresse de broadcast du réseau suivant (en binaire puis en décimale pointée).

Calcul d'une adresse de broadcast				
réseau (binaire)	11000000	10000000	00000000	00000000
masque (binaire)	11111111	11000000	00000000	00000000
complément du masque (binaire)				
broadcast (binaire)				
broadcast (déc. pointée)				

Une machine du réseau LAN_1 a reçu du serveur DHCP l'adresse `172.16.1.100`, avec le masque `255.255.0.0`.

Sa passerelle par défaut a pour adresse `172.16.255.254`.

8. En déduire les informations suivantes :
   - l'adresse du réseau LAN_1 (en décimale pointée) ;
   - l'adresse de broadcast (en décimale pointée) ;
   - le nombre total d'adresses pouvant être distribuées par le serveur, en ne tenant pas compte des éventuelles restrictions possiblement posées par l'administrateur du réseau.

On souhaite, désormais, simuler en `Python` le fonctionnement du réseau LAN_1.

On donne, ci-après, les documentations des méthodes `split` et `join`.

- la documentation de la méthode `split` de la classe `str` est la suivante :

```
split(self, séparateur)
  Renvoie la liste des sous-chaines de caractères
délimitées par le séparateur dans l'instance courante de
chaîne de caractères.
  Exemple :
>>> 'ab-pq-rs'.split('-')
['ab', 'pq', 'rs']
```

- la documentation de la méthode `join` de l'objet `str` :

```
join(self, iterable)
Fusionne les chaînes de caractères contenues dans iterable
en le liant par la sous-chaîne depuis laquelle cette
méthode est appelée.
Exemple:
>>>'.'.join(['ab', 'pq', 'rs'])
'ab.pq.rs'
```

On commence par créer la classe `IPv4` suivante.

```
 1 class IPv4(object):
 2     def __init__(self, adresse:str):
 3         """
 4         Constructeur de la classe de calcul sur une
 5         adresse IPv4 dont la notation décimale pointée
 6         est passée en paramètre.
 7         """
 8         self.adresse = adresse
 9 
10     def octets(self)->list[int]:
11         """
12         Découpe l'adresse décimale pointée en la liste
13         des 4 entiers correspondants.
14         """
15         return [int(i) for i in self.adresse.split(".")]
```

Pour mémoire, l'opérateur `&` entre deux entiers effectue le `ET` logique bit à bit de la représentation binaire de ces entiers et renvoie l'entier correspondant à la représentation binaire ainsi obtenue.

Exemple :

```
>>> 192 & 255
192
```

9. Recopier sur la copie et compléter les lignes 15 et 16 du code de la méthode `masquer` de la classe `IPv4`, donné ci-dessous. La méthode doit correspondre au docstring.

```
 1      def masquer(self, masque: str)->str:
 2          """
 3          Détermine le préfixe masqué de l'adresse,
 4          le masque (décimal pointé) étant passé en
 5          paramètre.
 6          >>> add  = IPv4('192.168.1.100')
 7          >>> add.masquer('255.192.0.0')
 8          '192.128.0.0'
 9          """
10          tmp = []
11          ip = self.octets()
12          crible = IPv4(masque).octets()
13          for i in range(4):
14              # Opération booléenne :
15              tmp.append(... & ...)
16          return ".".join(...)
```

10. Recopier sur la copie et compléter les lignes 19 et 20 du code de la méthode `adresse_suivante` de la classe `IPv4`, donné ci-dessous. La méthode doit correspondre au docstring.

```
 1      def adresse_suivante(self, adresse_max:str)->str:
 2          """
 3          Détermine l'adresse décimale pointée suivant
 4          immédiatement l'adresse courante, sous réserve
 5          d'existence d'une adresse disponible
 6          >>> add = IPv4('192.168.1.100')
 7          >>> add.adresse_suivante('192.168.1.254')
 8          '192.168.1.101'
 9          >>> add = IPv4('192.168.1.255')
10          >>> add.adresse_suivante('192.168.255.254')
11          '192.168.2.0'
12          """
13          assert self.adresse < adresse_max
14          liste_courante = self.octets()
15          liste_suivante = list()
16          retenue = 1
17          for index in range(4):
18              somme = liste_courante[3 - index] + retenue
19              valeur, retenue = ..., ...
20              liste_suivante = ...
21          return '.'.join(liste_suivante)
```

Exercice 2 (6 points)

Cet exercice porte sur la programmation Python et la récursivité.

Le jeu du baguenaudier est un jeu de casse-tête constitué d'une réglette comportant n cases, numérotées de 1 à n.

Chaque case peut être soit vide, soit contenir un pion.

« Jouer » une case consiste à placer un pion dans la case (remplir) si elle est vide ou enlever un pion (vider) si elle est remplie.

Initialement, toutes les cases sont vides.

Le but du jeu est de remplir toutes les cases du baguenaudier en suivant les règles suivantes :

- on ne peut jouer qu'une case à la fois ;
- chaque case ne peut contenir qu'un pion ;
- on peut toujours jouer la case 1 ;
- si le baguenaudier n'est ni vide ni rempli, on peut aussi jouer la case qui suit la première case remplie ;
- aucune autre case ne peut être jouée.

Exemple de situation avec un baguenaudier de 5 cases.

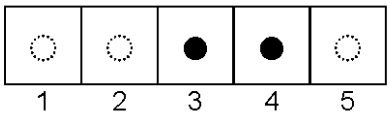

Figure 1. Situation baguenaudier 5 cases

Dans cette situation, on peut poser un pion dans la case 1 ou enlever le pion de la case 4 mais on ne peut pas jouer les cases 2, 3 et 5.

1. On considère un baguenaudier à 4 cases sur lequel les trois dernières cases sont remplies.

   Désigner les cases que l'on peut jouer et dessiner l'état du baguenaudier à la suite de chacun des coups possibles.

Pour modéliser le baguenaudier on utilise un tableau (de type `list`) de booléens. Une case vide est représentée par `False` et une case remplie par `True`.

L'exemple de situation du baguenaudier de 5 cases de la Figure 1 présentée plus haut sera ainsi représenté par le tableau suivant.

```
[False, False, True, True, False]
```

La fonction `initialiser` prend en paramètre un nombre entier n et elle renvoie une liste modélisant un baguenaudier vide de n cases.

Exemple :

```
>>> initialiser(3)
[False, False, False]
```

2. Écrire le code de la fonction `initialiser`

La fonction `victoire` renvoie un booléen indiquant si toutes les cases du baguenaudier sont remplies.

```
1 def victoire(tab):
2     for etat_case in tab:
3         if etat_case == ...:
4             return ...
5     ...
```

3. Recopier et compléter les lignes 3, 4 et 5 du code de la fonction `victoire`.

La fonction `indice_premiere_case_occupee` prend en paramètre un tableau `tab` modélisant l'état du baguenaudier et renvoie l'indice de la première case remplie du baguenaudier, ou `None` si le baguenaudier est vide.

Exemples :

```
>>> indice_premiere_case_occupee([True, True, True])
0
>>> indice_premiere_case_occupee([False, False, False, True,
True])
3
>>> indice_premiere_case_occupee([False, False, False, False])
# pas de sortie car valeur None
```

4. Écrire le code de la fonction `indice_premiere_case_occupee`.

La fonction `coup_valide` prend en paramètres `tab` qui est une liste modélisant l'état d'un baguenaudier, et un entier `case` qui est l'indice de la case à jouer. La fonction renvoie `True` si le coup respecte les règles et `False` si ce n'est pas le cas.

L'indice de la première case du jeu est 0. Pour que le coup soit valide, il faut aussi s'assurer que l'indice est un entier positif inférieur à la longueur de `tab`.

5. Écrire le code la fonction `coup_valide`.

La fonction `changer_case` prend en paramètres un tableau `tab` modélisant l'état du baguenaudier et un indice de case nommé `case`. Si le coup désigné par `case` est valide, la fonction renvoie le tableau modélisant le baguenaudier obtenu après avoir changé l'état de la case correspondante. Si le coup joué n'est pas valide, le baguenaudier passé en paramètre est renvoyé sans modification.

Exemples :

```
>>> changer_case([False, False, False], 1) # coup non valide
[False, False, False]
>>> changer_case([False, False, False], 0) # coup valide
[True, False, False]
>>> changer_case([False, True, False], 2)  # coup valide
[False, True, True]
```

6. Écrire le code de la fonction `changer_case`.

La résolution d'un baguenaudier de 3 cases, en Figure 2 et 4 cases, en Figure 3 sont données ci-dessous.

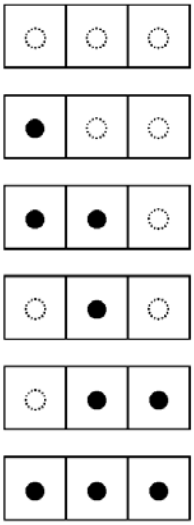

Figure 2. Résolution du baguenaudier à 3 cases

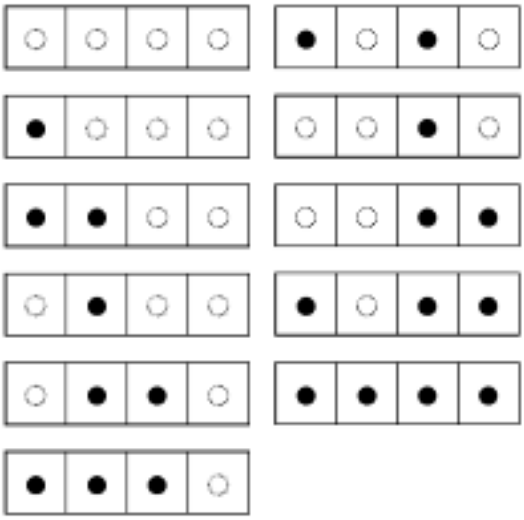

Figure 3. Résolution du baguenaudier à 4 cases

Il est possible d'obtenir la solution du jeu du baguenaudier, sous forme d'affichage, en utilisant des fonctions récursives nommées `vider` et `remplir`. Ces deux fonctions prennent en paramètre un entier `n` correspondant au nombre de cases du jeu. La fonction `vider` vide un baguenaudier de `n` cases initialement remplies. La fonction `remplir` remplit un baguenaudier de `n` cases initialement vides.

Le code de la fonction `vider`, incomplet, est donné ci-dessous. On notera que les cases sont numérotées à partir de 1 dans la suite et que l'appel `vider(0)`ne fait rien.

```
1 def vider(n):
2     if n == 1:
3         print('Vider case 1')
4     elif n > 1:
5         vider(n-2)
6         print(...)
7         remplir(n-2)
8         vider(n-1)
```

7. Recopier et compléter la ligne 6 du code de la fonction `vider` pour qu'elle affiche le texte `'Vider case '` suivi de la valeur de l'entier `n`.

8. En supposant que l'appel `remplir(1)` affiche `'Remplir case 1'` et que l'appel `remplir(0)` ne fait rien, donner les affichages, dans la console, produits par l'exécution de `vider(3)`.

9. En vous inspirant de la fonction `vider`, de la résolution donnée à la question précédente pour `n = 3` et des Figures 2 et 3, écrire le code de la fonction récursive `remplir`.

10. On envisage d'utiliser la fonction `vider` pour un baguenaudier de 2000 cases. Expliquer si ce code est adapté à un baguenaudier de si grande taille. Justifier votre réponse.

Exercice 3 (8 points)

Cet exercice porte sur la programmation en Python, les bases de données relationnelles, le langage SQL, les systèmes d'exploitation et la sécurisation des communications.

Pour simplifier la gestion de ses mots de passe, Alice décide de créer un gestionnaire de mots de passe regroupant les informations de connexion de chaque site qu'elle utilise.

Partie A

Dans cette partie, on s'intéresse à la création des mots de passe d'Alice. On s'aidera de la table ASCII de la Figure 1 donnant le code ASCII en décimal et en hexadécimal des différents caractères utilisés dans ce sujet (lettres minuscules, lettres majuscules et les caractères spéciaux encadrés).

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Figure 1. Table ASCII. Les caractères spéciaux utilisés dans ce sujet sont encadrés

Source : d'après https://fr.m.wikipedia.org/wiki/Fichier : ASCII-Table.svg

Alice souhaite créer une fonction Python `gen_mdp` lui permettant de créer aléatoirement un mot de passe. Cette fonction :

- prend en paramètres :
  - `longueur` (de type `int`) : le nombre de caractères du mot de passe ;
  - `cont_min` (de type `bool`) : un booléen indiquant si le mot de passe peut contenir des minuscules (`True`) ou non (`False`) ;
  - `cont_maj` (de type `bool`) : un booléen indiquant si le mot de passe peut contenir des majuscules (`True`) ou non (`False`) ;
  - `cont_spe` (de type `bool`) : un booléen indiquant si le mot de passe peut contenir des caractères spéciaux (`True`) ou non (`False`) ;
- et renvoie un mot de passe (de type `str`) respectant les conditions définies par les paramètres.

On donne ci-dessous le code de la fonction `gen_mdp` qui sera à compléter au fur et à mesure des questions.

```
 1 from random import randint
 2
 3 def gen_mdp(longueur, cont_min, cont_maj, cont_spe):
 4     # Pour qu'un mot de passe soit non vide, il doit
 5     # pouvoir contenir des minuscules ou des majuscules
 6     # ou des caractères spéciaux.
 7     assert (cont_min or cont_maj or cont_spe)
 8     minuscules = [chr(i) for i in ...]
 9     majuscules = [...]
10     caracteres_speciaux = ... + ...
11     jeu_caracteres = []
12     if cont_min:
13         ...
14     ...
15         ...
16     ...
17         ...
18     mot_de_passe = ''
19     n = len(jeu_caracteres)
20     for i in range(longueur):
21         mot_de_passe = ...
22     return mot_de_passe
```

On rappelle que l'opérateur + permet en Python d'additionner deux éléments de type `int`, mais aussi de concaténer deux éléments de type `list`, ou encore deux éléments de type `str`.

On rappelle également que `chr(i)` renvoie la chaîne représentant un caractère dont le code ASCII est le nombre entier `i`.

Par exemple, `chr(97)` renvoie la chaîne de caractères `'a'`, tandis que `chr(33)` renvoie la chaîne de caractères `'!'`.

1. Alice s'inscrit sur un nouveau site lui demandant de créer un mot de passe de 8 caractères minimum, composé uniquement de majuscules et de minuscules. Écrire un appel à la fonction `gen_mdp` qui permette de répondre aux exigences de ce site.

On admet que les caractères spéciaux sont uniquement ceux encadrés à la Figure 1.

2. Recopier et compléter les lignes 8 à 10 du code de la fonction `gen_mdp` permettant d'initialiser les trois variables `minuscules`, `majuscules` et `caracteres_speciaux`. On utilisera obligatoirement la syntaxe de tableaux donnés en compréhension pour cette initialisation.
   Après cette initialisation, on doit avoir :

   - `minuscules` initialisée à `['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']`
   - `majuscules` initialisée à `['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']`
   - `caracteres_speciaux` initialisée à `['!', '"', '#', '$', '%', '&', "'", '(', ')', '*', '+', ',', '-', '.', '/', ':', ';', '<', '=', '>', '?', '@']`

3. La variable `jeu_caracteres` de la fonction `gen_mdp` est une liste contenant tous les caractères autorisés pour fabriquer un nouveau mot de passe. Recopier et compléter la ligne 13 du code de la fonction `gen_mdp`, puis écrire les autres lignes de code afin de créer la variable `jeu_caracteres`. Le nombre de lignes de code à insérer est laissé à l'appréciation du candidat.

4. La documentation de la fonction `randint` indique que `randint(a, b)` renvoie un entier aléatoire compris entre les entiers `a` et `b` incluant les deux extrémités `a` et `b`. Recopier et compléter la ligne 21 du code de la fonction `gen_mdp`.

5. Un site demande de créer un mot de passe possédant obligatoirement les caractéristiques suivantes :

   - le mot de passe doit contenir au moins 12 caractères ;
   - le mot de passe doit contenir au moins un caractère spécial et au moins une lettre minuscule.

   Expliquer en quoi le code de la fonction `gen_mdp` peut renvoyer un mot de passe qui ne répond pas aux exigences du site.

Partie B

Dans cette partie, on pourra utiliser les clauses du langage SQL pour :

- construire des requêtes d'interrogation à l'aide de `SELECT`, `FROM`, `WHERE` (avec les opérateurs logiques `AND`, `OR`), `JOIN ... ON` ;
- construire des requêtes d'insertion et de mise à jour à l'aide de `UPDATE`, `INSERT`, `DELETE` ;
- affiner les recherches à l'aide de `DISTINCT`, `ORDER BY`.

Alice souhaite mettre en œuvre une base de données composée des deux tables `compte` et `site` dont des extraits sont donnés dans les tableaux suivants.

compte			
mot_de_passe	utilisateur	renouvellement	id_site
Asrtg!Myfj	aliceB24	2022-06-30	1
@rDfohpj!&	aliceB24	2021-03-12	2
GxRGDxc(u-PM	alice_B@votremailp.me	2018-10-14	4
Ghcj=+f*AZs	alice1276	2022-06-30	3
cYFgt!:Ehr;	alice_B2@votremailp.me	2022-06-30	4

site		
id	nom_site	url
1	Vosnotes	https://logi-educ.net/vosnotes/eleve.html
2	Banque Perso	https://www.banqueperso.fr/connexion.html
3	Elec verte	https://espace-client.ev.fr/login
4	Votremailp	https://account.votremailp.me/login

- L'attribut `mot_de_passe` est une clé primaire de la table `compte`.
- L'attribut `id` est une clé primaire de la table `site`.
- L'attribut `renouvellement`, correspondant au dernier renouvellement du mot de passe, est une chaîne de caractères de format AAAA-MM-JJ. Ainsi un mot de passe renouvelé le 21 février 2025 correspond à un attribut `renouvellement` de `'2025-02-21'`.

- Dans la table `compte`, l'attribut `id_site` est une clé étrangère référençant l'attribut `id` de la table `site`.

6. Expliquer en quoi il n'est pas possible, pour Alice, d'avoir le même mot de passe pour deux sites différents.

7. Écrire la requête SQL permettant d'afficher toutes les URL enregistrées dans la base de données.

8. La Banque Perso a demandé à Alice de renouveler son mot de passe. Elle remplace le mot de passe `'@rDfohpj!&'` par `'yhTS?d@UTJe'`. Écrire la requête SQL permettant de faire la modification dans la base de données. Dans cette question, on ignorera l'attribut `renouvellement` et on ne cherchera pas à le mettre à jour.

On rappelle que le langage SQL utilise l'ordre lexicographique (ordre du dictionnaire) pour comparer deux éléments de type texte. Selon cet ordre, on a par exemple `'3'` supérieur à `'1'` et `'python'` est inférieur à `'sql'`.

9. À la date du 20 mars 2025, Alice a décidé de lister tous les mots de passe qui n'ont pas été renouvelés depuis plus d'un an. Écrire la requête SQL permettant de donner la liste des `id_site` concernés.

10. Donner une raison pour laquelle Alice a préféré choisir le format AAAA-MM-JJ pour l'attribut `renouvellement` plutôt que le format JJ-MM-AAAA.

11. Écrire une requête SQL permettant d'afficher tous les utilisateurs et les mots de passe du ou des sites de nom `'Votremailp'` dont l'identifiant est supposé non connu. Le résultat devra être affiché par ordre chronologique de date de renouvellement de mot de passe.

12. Pour réaliser le projet, il aurait été possible de regrouper toutes les informations dans une seule table mais Alice a choisi d'utiliser deux tables : les tables `site` et `compte`. Donner un avantage impliqué par le choix d'Alice.

Partie C

Dans cette partie, on s'intéresse à la sécurité du gestionnaire de mots de passe d'Alice. Alice utilise un système d'exploitation multi-utilisateurs basé sur Linux et elle a les droits de lecture, d'écriture et d'exécution dans son répertoire personnel `/home`.

La base de données est stockée dans un fichier `gestionnaire.db`. Alice décide de chiffrer ce fichier. Elle crée un fichier `chiffrement.py` dans lequel elle écrit le code d'une fonction Python `chiffrement` dont la documentation est donnée ci-après.

```
1 def chiffrement(f_source, f_dest, f_cle):
2     '''Crée un fichier f_dest contenant les données
3     du fichier f_source chiffrées selon la clé
4     contenue dans le fichier f_cle.
```

```
5       f_source (str) : chemin vers le fichier à chiffrer
6       f_dest (str) : chemin vers le fichier chiffré
7       f_cle (str) : chemin vers le fichier contenant la clé
8       '''
```

On donne l'arborescence de fichiers utilisée par Alice sur la Figure 2.

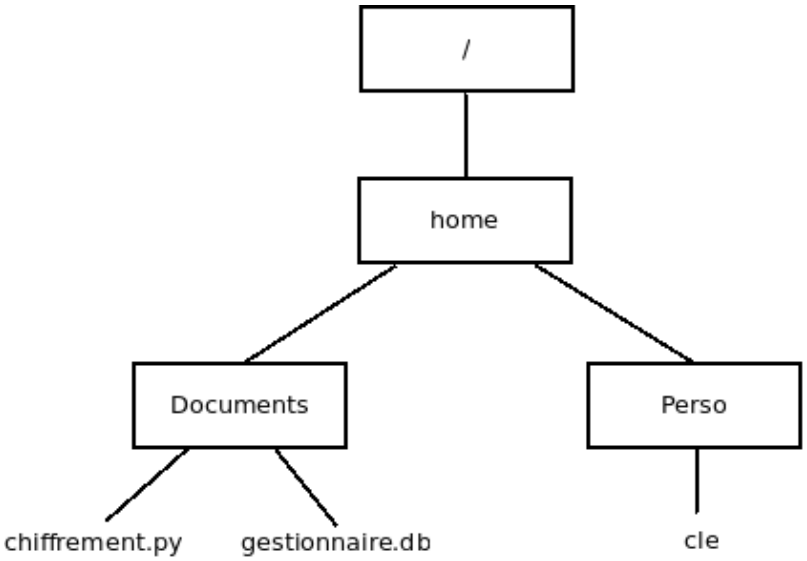

Figure 2. Arborescence de fichiers

Alice ouvre un interpréteur Python dans le répertoire `Documents`, qui est ainsi le répertoire courant (ou répertoire de travail) et exécute le contenu du fichier `chiffrement.py`.

13. Écrire l'appel à la fonction `chiffrement` permettant, à l'aide de la clé contenue dans le fichier `cle`, de chiffrer le fichier `gestionnaire.db` en créant le fichier chiffré `secret.db` dans le répertoire `Perso`.

L'algorithme de chiffrement consiste à appliquer l'opération OU-exclusif bit à bit entre le fichier source et le fichier contenant la clé de chiffrement.

Le premier bit du fichier chiffré est le résultat du OU-exclusif entre le premier bit du fichier source et le premier bit du fichier clé. De même, le deuxième bit est obtenu par le OU-exclusif entre les deuxièmes bits de ces deux fichiers et ainsi de suite.

On suppose, pour simplifier, que le fichier contenant la clé est de la même taille que le fichier source, qui est donc aussi la taille du fichier chiffré.

On rappelle ci-après la table de vérité du OU-exclusif, noté XOR.

Table de vérité du XOR		
a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

On rappelle que l'hexadécimal correspond à l'écriture des entiers dans la base 16. Les chiffres de cette base sont notés `0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F`.

14. Donner le résultat, en écriture hexadécimale, de l'opération OU-exclusif bit à bit entre deux octets dont les écritures hexadécimales sont respectivement `A3` et `59`.

15. Montrer que, pour tous bits `a` et `b`, on a la propriété suivante : `(a XOR b) XOR b = a`.

16. Indiquer, en justifiant, si le chiffrement utilisé par Alice est symétrique ou asymétrique.

Alice exécute la commande `ls -l secret.db` et obtient la réponse donnée ci-dessous :

```
-rw-r--r-- 1 alice eleves 42480 mars 25 11:57 secret.db
```

17. Justifier le fait qu'un attaquant a le champ libre pour tenter un déchiffrement du fichier `secret.db` et expliquer ce que devrait faire Alice pour corriger ce problème.

Partie D

On donne ci-dessous quatre bonnes pratiques proposées par le gouvernement en matière de gestion des mots de passe.

- P1 : Utilisez un mot de passe différent pour chaque accès (messagerie, banque en ligne, comptes de réseaux sociaux, etc.) : en cas de compromission de l'un de vos comptes, cela évitera l'effet boule de neige.
- P2 : Créez un mot de passe suffisamment long, complexe et inattendu, de 8 caractères minimum, contenant des minuscules, des majuscules, des chiffres et des caractères spéciaux.
- P3 : Ne communiquez jamais votre mot de passe à un tiers : aucune organisation ou personne de confiance ne vous demandera de lui communiquer votre mot de passe.

- P4 : Utilisez un gestionnaire de mots de passe : pas simple de retenir tous ses codes de connexion ! Heureusement des outils de type « coffres forts de mots de passe » existent. Ces derniers mémorisent tous vos mots de passe et vous permettent d'en générer de manière aléatoire.

Source : d'après https://cyber.gouv.fr/bonnes-pratiques-protegez-vous

18. En justifiant votre réponse, préciser en quoi les choix d'Alice dans la conception de son gestionnaire de mots de passe respectent ou non chacune des quatre bonnes pratiques mentionnées.