Bac Général Maths + NSI (Mathématiques et Numérique et sciences informatiques)

SUJET D'EXAMEN (SESSION OFFICIELLE)

Numérique et sciences informatiques - Session 2021

Sujet de l'examen national tel que publié pour la session indiquée. Source : sujets nationaux publiés par le ministère chargé de l'examen et les services académiques. Document mis à disposition par FormaV pour la révision.

Toutes les annales et leurs corrigés : https://bac-maths-nsi.fr/annales-sujets-et-corriges

BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2021

NUMÉRIQUE ET SCIENCES INFORMATIQUES

Jour 2

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 13 pages numérotées de 1/13 à 13/13.

Le candidat traite au choix 3 exercices parmi les 5 exercices proposés

Chaque exercice est noté sur 4 points.

EXERCICE 1 (4 points)

Principaux thèmes abordés : protocoles de communication, architecture d'un réseau et protocoles de routage.

Les parties A et B sont indépendantes.

Partie A : Réseau

1. Parmi les termes ci-dessous, préciser celui qui désigne l'ensemble des règles de communication utilisées pour réaliser un service particulier sur le réseau ?

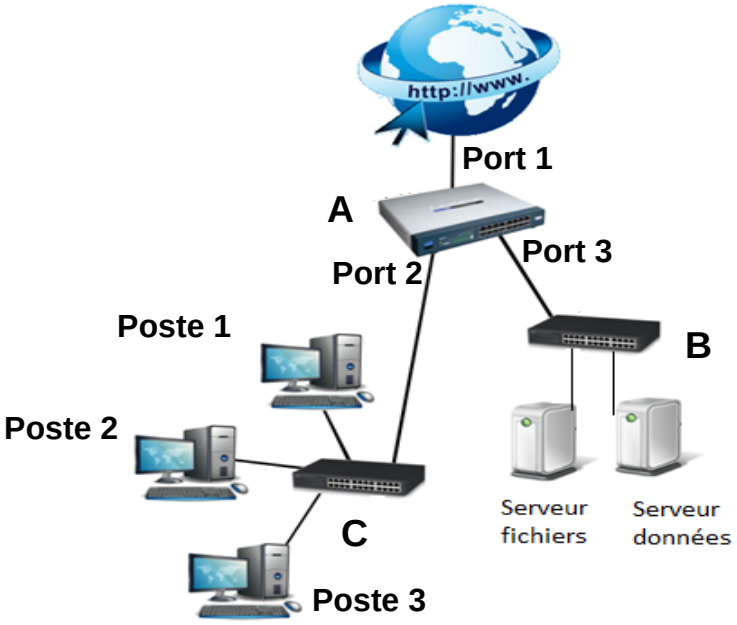
Architecture	Protocole	Paquet

2. On considère le schéma réseau de l'entreprise Lambda :

Parmi les quatre propositions suivantes (Routeur, Commutateur (Switch), Contrôleur WIFI et Serveur), préciser celle qui correspond à :

a) L'élément A

b) L'élément B



3. En reprenant le schéma de la question **2.** et le tableau d'adressage du réseau de l'entreprise Lambda, recopier sur votre copie et compléter la ligne du tableau du poste 3 :

Matériel	Adresse IP	Masque	Passerelle
Routeur Port 1	172.16.0.1	255.255.0.0	
Routeur Port 2	192.168.11.1	255.255.255.0	
Routeur Port 3	192.168.11.254	255.255.255.0	
Serveur fichiers	192.168.11.10	255.255.255.0	192.168.11.1
Serveur données	192.168.11.11	255.255.255.0	192.168.11.1
Poste 1	192.168.11.20	255.255.255.0	192.168.11.1
Poste 2	192.168.11.21	255.255.255.0	192.168.11.1
Poste 3			

Partie B : Routage réseaux

L'extrait de la table de routage d'un routeur R1 est donné ci-dessous :

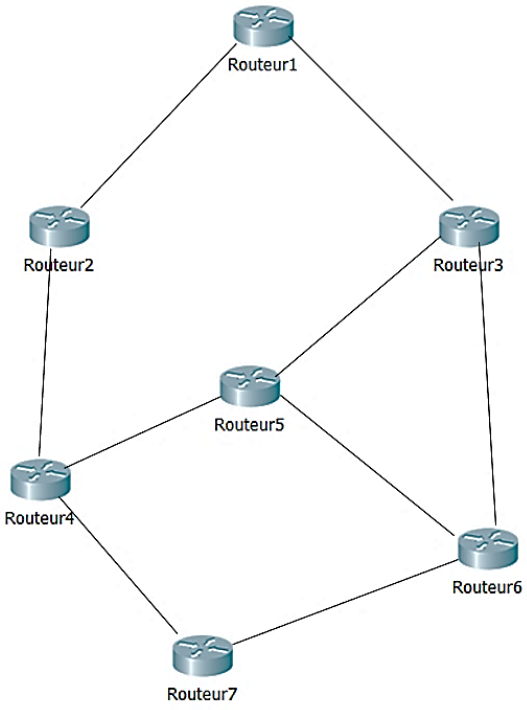
Réseau IP destination		**Passerelle**	**Interface Machine ou Port**	**Métrique (distance)**
Réseau IP	Masque			
10.0.0.0	255.0.0.0	10.0.0.1	10.0.0.1	0
172.16.0.0	255.255.0.0	172.16.0.1	172.16.0.1	0
192.168.0.0	255.255.255.0	192.168.0.1	192.168.0.1	0
11.0.0.0	255.0.0.0	192.168.0.2	192.168.0.1	1
172.17.0.0	255.255.0.0	192.168.0.2	192.168.0.1	1
172.18.0.0	255.255.0.0	172.15.0.2	172.15.0.1	1
192.168.1.0	255.255.255.0	192.168.0.2	192.168.0.1	1
192.168.2.0	255.255.255.0	172.15.0.2	172.15.0.1	1

1. Indiquer sur votre copie les adresses IP du(des) réseau(x) directement connectés à ce routeur.

2. Indiquer sur votre copie l'interface utilisée pour transférer les paquets contenant les adresses IP destination suivantes :

Adresse IP destination	**Interface Machine ou Port**
192.168.1.55	
172.18.10.10	

3. On considère un réseau selon le schéma ci-contre.



Recopier sur votre copie et compléter la table de routage simplifiée du Routeur1 (R1) (ci-dessous) en prenant comme métrique le nombre de routeurs à « traverser » avant d'atteindre le réseau de la machine destinataire.

Table de routage simplifiée du Routeur1		
Routeur destination	***Métrique***	***Route***
R2 : Routeur2	*0*	*R1 – R2*
…		

EXERCICE 2 (4 points)

Principaux thèmes abordés : structure de données (tableaux, dictionnaires) et langages et programmation (spécification).

Objectif de l'exercice :

Les Aventuriers du Rail© est un jeu de société dans lequel les joueurs doivent construire des lignes de chemin de fer entre différentes villes d'un pays.

La carte des liaisons possibles dans la région Occitanie est donnée en **annexe 1 de l'exercice 2**. Dans l'**annexe 2 de l'exercice 2**, les liaisons possédées par le joueur 1 sont en noir, et celles du joueur 2 en blanc. Les liaisons en gris sont encore en jeu.

Codages des structures de données utilisées :

- **Liste des liaisons d'un joueur :** Toutes les liaisons directes (sans ville intermédiaire) construites par un joueur seront enregistrées dans une variable de type "**tableau de tableaux**".

  Le joueur 1 possède les lignes directes **"**Toulouse-Muret", "Toulouse-Montauban", "Gaillac-St Sulpice" et "Muret-Pamiers" (liaisons indiquées en noir dans l'**annexe 2 de l'exercice 2**). Ces liaisons sont mémorisées dans la variable ci-contre.

```
liaisonsJoueur1 = [
 ["Toulouse","Muret"],
 ["Toulouse","Montauban"],
 ["Gaillac","St Sulpice"],
 ["Muret","Pamiers"]
]
```

  Remarque : *Seules les liaisons directes existent, par exemple* ***["Toulouse","Muret"]*** *ou* ***["Muret","Toulouse"]****. Par contre, le tableau* ***["Toulouse","Mazamet"]*** *n'existe pas, puisque la ligne Toulouse-Mazamet passe par Castres.*

- **Dictionnaire associé à un joueur :** On code la liste des villes et des trajets possédée par un joueur en utilisant un **dictionnaire de tableaux**. Chaque **clef de ce dictionnaire** est une ville de départ, et chaque **valeur** est un **tableau** contenant les villes d'arrivée possibles en fonction des liaisons possédées par le joueur.

  Le **dictionnaire de tableaux** du joueur 1 est donné ci-contre :

```
DictJoueur1 = {
    "Toulouse":["Muret","Montauban"],
    "Montauban":["Toulouse"],
    "Gaillac":["St Sulpice"],
    "St Sulpice":["Gaillac"],
    "Muret":["Toulouse","Pamiers"],
    "Pamiers":["Muret"]}
```

1. Expliquer pourquoi la liste des liaisons suivante n'est pas valide :

```
tableauliaisons = [["Toulouse","Auch"], ["Luchon","Muret"],
["Quillan","Limoux"] ]
```

2. Cette question concerne le joueur n°2 *(Rappel : les liaisons possédées par le joueur n°2 sont représentées par un rectangle blanc dans l'**annexe 2 de l'exercice 2**)*.

a) Donner le tableau `liaisonsJoueur2`, des liaisons possédées par le joueur n°2.

b) Recopier et compléter le dictionnaire suivant, associé au joueur n°2 :

```
DictJoueur2 = {
    "Toulouse":["Castres","Castelnaudary"],
    …
}
```

3. À partir du tableau de tableaux contenant les liaisons d'un joueur, on souhaite construire le dictionnaire correspondant au joueur. Une première proposition a abouti à la fonction **`construireDict`** ci-dessous.

```
1   def construireDict(listeLiaisons):
2     """
3     listeLiaisons est un tableau de tableaux représentant la
4     liste des liaisons d'un joueur comme décrit dans le problème
5     """
6     Dict={}
7     for liaison in listeLiaisons :
8       villeA = liaison[0]
9       villeB = liaison[1]
10      if not villeA in Dict.keys() :
11        Dict[villeA]=[villeB]
12      else :
13        destinationsA = Dict[villeA]
14        if not villeB in destinationsA :
15          destinationsA.append(villeB)
16    return Dict
```

a) Écrire sur votre copie un assert dans la fonction **`construireDict`** qui permet de vérifier que la **`listeLiaisons`** n'est pas vide.

b) Sur votre copie, donner le résultat de cette fonction ayant comme argument la variable **`liaisonsJoueur1`** donnée dans l'énoncé et expliquer en quoi cette fonction ne répond que partiellement à la demande.

c) La fonction `construireDict`, définie ci-dessus, est donc partiellement inexacte.

Compléter la fonction `construireDict` pour qu'elle génère bien l'ensemble du dictionnaire de tableaux correspondant à la liste de liaisons données en argument. À l'aide des numéros de lignes, on précisera où est inséré ce code.

EXERCICE 3 (4 points)

Principaux thèmes abordés : bases de données (modèle relationnel, base de données relationnelle et langage SQL).

Dans notre monde, l'information a de plus en plus de valeur et d'importance mais nous sommes de plus en plus confrontés à l'infobésité.

Considérons l'utilisation des données issues de la table de Mendeleïev (tableau périodique des éléments). Il est contraignant de faire des recherches sur des moteurs dédiés à chaque fois qu'une valeur est nécessaire (masse volumique, rayon de covalence, point de fusion…).

Les lignes 3, 4 et 5 de cette table Mendeleïev ont permis de construire, en **annexe 1 de l'exercice 3,** une base de données des différents atomes correspondants.

1. Donner le nom du langage informatique utilisé pour accéder aux données dans une base de données ?

2. a) Lister les différents attributs des tables ATOMES et VALENCE en précisant le type du domaine de chacun.

b) Déterminer si des attributs de la table ATOMES peuvent avoir un rôle de clé primaire et/ou de clé étrangère. Justifier.

c) Donner le schéma relationnel pour les deux tables ATOMES et VALENCE.

3. Donner les réponses des deux requêtes suivantes :

a) `SELECT nom FROM ATOMES WHERE L='3' ORDER BY Sym`

b) `SELECT DISTINCT C FROM ATOMES`

4. Donner la requête SQL:

a) Pour afficher le nom et la masse atomique des atomes.

b) Pour afficher le symbole des atomes dont la couche de valence est s.

5. On a remarqué une erreur de saisie dans la table ATOMES, la masse atomique de l'argon (Ar) n'est pas 29,948 $g.mol^{-1}$ mais 39,948 $g.mol^{-1}$. Écrire la requête SQL pour corriger cette erreur de saisie.

EXERCICE 4 (4 points)

Principaux thèmes abordés : structure de données (programmation objet) et langages et programmation (spécification).

Une entreprise fabrique des yaourts qui peuvent être soit nature (sans arôme), soit aromatisés (fraise, abricot ou vanille).

Pour pouvoir traiter informatiquement les spécificités de ce produit, on va donc créer une classe *Yaourt* qui possèdera un certain nombre d'attributs :

- Son genre : nature ou aromatisé
- Son arôme : fraise, abricot, vanille ou aucun
- Sa date de durabilité minimale (DDM) exprimée par un entier compris entre 1 et 365 (on ne gère pas les années bissextiles). Par exemple, si la DDM est égale à 15, la date de durabilité minimale est le 15 janvier.

On va créer également des méthodes permettant d'interagir avec l'objet `Yaourt` pour attribuer un arôme ou récupérer un genre par exemple. On peut représenter cette classe par le tableau de spécifications ci-dessous :

Yaourt	
ATTRIBUTS: - genre - arome - duree	**METHODES**: - Construire(arome,duree) - Obtenir_Arome() - Obtenir_Genre() - Obtenir_Duree() - Attribuer_Arome(arome) - Attribuer_Duree(duree) - Attribuer_Genre(arome)

1. La classe `Yaourt` est déclarée en Python à l'aide du mot-clé class :

```
class Yaourt:
    """ Classe définissant un yaourt caractérisé par :
    - son arome
    - son genre
    - sa durée de durabilité minimale"""
```

L'annexe 1 de l'exercice 4 donne le code existant et l'endroit des codes à produire dans les questions suivantes.

a) Quelles sont les assertions à prévoir pour vérifier que l'arôme et la durée correspondent bien à des valeurs acceptables. Il faudra aussi expliciter les commentaires qui seront renvoyés.

Pour rappel :

- L'arôme doit prendre comme valeur 'fraise', 'abricot', 'vanille' ou 'aucun'.
- Sa date de durabilité minimale (DDM) est une valeur positive.

b) Pour créer un yaourt, on exécutera la commande suivante :

```
Mon_Yaourt = Yaourt('fraise',24)
```

Quelle valeur sera affectée à l'attribut genre associé à `Mon_Yaourt` ?

c) Écrire en python une fonction `GetArome(self)`, renvoyant l'arôme du yaourt créé.

2. On appelle mutateur une méthode permettant de modifier un ou plusieurs attributs d'un objet.

Sur votre copie, écrire en Python le mutateur `SetArome(self,arome)` permettant de modifier l'arôme du yaourt.

On veillera à garder une cohérence entre l'arôme et le genre.

3. On veut créer une pile contenant le stock de yaourts. Pour cela il faut tout d'abord créer une pile vide :

```
def creer_pile():
    pile = [ ]
    return pile
```

a) Créer une fonction `empiler(p,Yaourt)` qui renvoie la pile p après avoir ajouté un objet de type `Yaourt` à la fin.

b) Créer une fonction `depiler(p)` qui renvoie l'objet à dépiler.

c) Créer une fonction `estVide(p)` qui renvoie `True` si la pile est vide et `False` sinon.

d) Qu'affiche le bloc de commandes suivantes ci-dessous ?

```
mon_Yaourt1 = Yaourt('aucun',18)
mon_Yaourt2 = Yaourt('fraise',24)
ma_pile = creer_pile()
empiler(ma_pile, mon_Yaourt1)
empiler(ma_pile, mon_Yaourt 2)
print(depiler(ma_pile).GetDuree())
print(estVide(ma_pile))
```

EXERCICE 5 (4 points)

Principaux thèmes abordés : Traitement de données en tables (CSV) et langages et programmation (spécification).

Afin d'améliorer l'ergonomie d'un logiciel de traitement des inscriptions dans une université, un programmeur souhaite exploiter l'intelligence artificielle pour renseigner certains champs par auto-complétion. Il s'intéresse au descripteur « genre » (masculin, féminin ou indéterminé). Pour cela il propose d'exploiter les dernières lettres du prénom pour proposer automatiquement le genre.

Pour vérifier son hypothèse, il récupère un fichier CSV associant plus de 60 000 prénoms du monde entier au genre de la personne portant ce prénom. En utilisant seulement la dernière lettre, le taux de réussite de sa démarche est de 72,9% avec la fonction définie ci-dessous :

```
1  def genre(prenom):
2      liste_M = ['f', 'd', 'c', 'b', 'o', 'n', 'm', 'l', 'k',
3                 'j', 'é', 'h', 'w', 'v', 'u', 't', 's', 'r',
4                 'q', 'p', 'i', 'þ', 'z', 'x', 'ç', 'ö', 'ã',
5                 'â', 'ï', 'g']
6      liste_F = ['e', 'a', 'ä', 'ü', 'y', 'ë']
7
8      if prenom[len(prenom)-1].lower() in liste_M :
9          return "M"
10     elif prenom[len(prenom)-1].lower() in liste_F :
11         return "F"
12     else :
13         return "I"


   #  Pour rappel, C.lower() convertit le caractère C en minuscule.
```

1. Appropriation

a) Expliquer ce qu'est un fichier CSV.

b) Donner le type de l'argument **`prenom`** de la fonction **`genre,`** et le type de la réponse renvoyée.

2. Développement

Pour effectuer son étude sur les prénoms à partir du fichier CSV, le programmeur souhaite utiliser la bibliothèque csv.

a) La bibliothèque csv est un module natif du moteur python.

Donner, dans ce cas, l'instruction d'importation.

b) Au cours du développement de son projet, le programmeur souhaite insérer une assertion sur l'argument donné à la fonction.

Proposer une assertion sur le type de l'argument qui corrige une erreur lorsque le type ne correspond pas à celui attendu.

c) Avant le déploiement de sa solution, le programmeur décide de rendre sa fonction plus robuste.

Pour cela il veut remplacer l'assertion proposée dans la question **2.b)** par une gestion de l'argument pour éviter toutes erreurs empêchant la poursuite du programme.

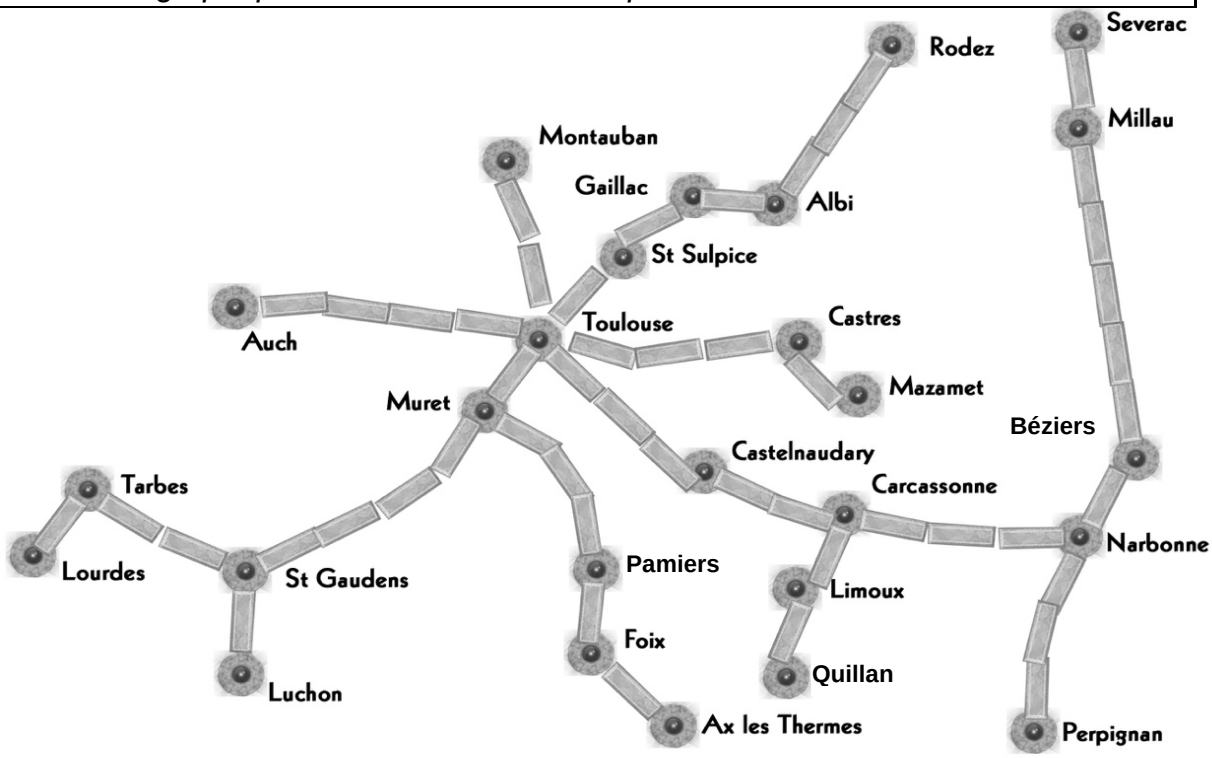
Proposer alors une ou plusieurs instructions en Python utilisant l'argument afin de s'assurer que la fonction se termine quel que soit le type de l'argument.

3. Améliorations

En prenant en compte les deux dernières lettres du prénom, il parvient à augmenter son taux de réussite à 74,4%. Pour cela, son étude du fichier CSV lui permet de créer deux listes : **`liste_M2`** pour les terminaisons de deux lettres associées aux prénoms masculins et **`liste_F2`** pour les prénoms féminins.
Sur votre copie, recopier et modifier la structure conditionnelle (lignes 8 à 13) de la fonction **`genre`** afin de prendre en compte les terminaisons de deux lettres des listes **`liste_M2`** et **`liste_F2`**.

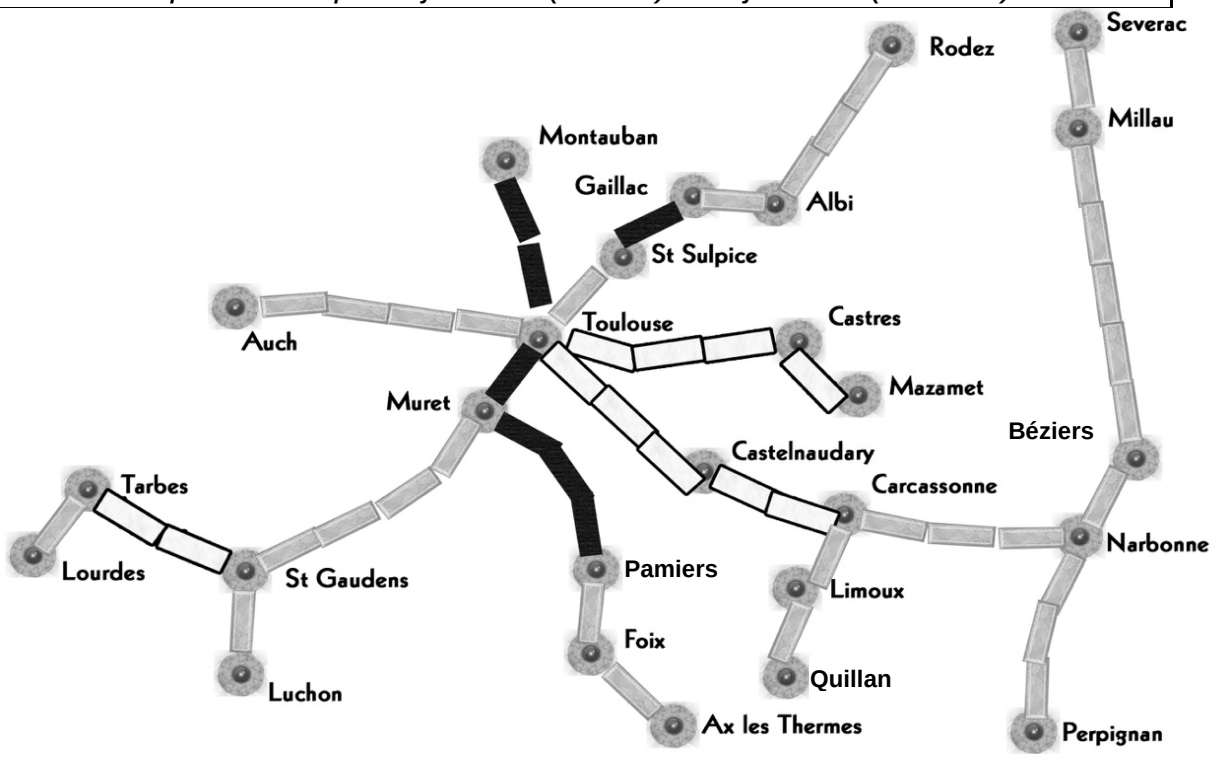
Exercice 2 - Annexe 1

graphique 1 : Extrait des liaisons possibles en Occitanie



Exercice 2 - Annexe 2

Liaisons possédées par le joueur 1 (en noir) et le joueur 2 (en blanc)



Exercice 3 - Annexe 1

Relation « ATOMES »

Z	nom	Sym	L	C	masse_atom
11	sodium	Na	3	1	22,9897693
12	magnesium	Mg	3	2	24,305
13	aluminium	Al	3	13	26,9815386
14	silicium	Si	3	14	28,0855
15	phosphore	P	3	15	30,973762
16	soufre	S	3	16	32,065
17	chlore	Cl	3	17	35,453
18	argon	Ar	3	18	**29,948**
19	potassium	K	4	1	39,0983
20	calcium	Ca	4	2	40,078
21	scandium	Sc	4	3	44,955912
22	titane	Ti	4	4	47,867
23	vanadium	V	4	5	50,9415
24	chrome	Cr	4	6	51,9961
25	manganese	Mn	4	7	54,938045
26	fer	Fe	4	8	55,845
27	cobalt	Co	4	9	58,933195
28	nickel	Ni	4	10	58,6934
29	cuivre	Cu	4	11	63,546
30	zinc	Zn	4	12	65,409
31	gallium	Ga	4	13	69,723
32	germanium	Ge	4	14	72,64
33	arsenic	As	4	15	74,9216
34	selenium	Se	4	16	78,96
35	brome	Br	4	17	79,904
36	krypton	Kr	4	18	83,798
37	rubidium	Rb	5	1	85,4678
38	strontium	Sr	5	2	87,62
39	yttrium	Y	5	3	88,90585
40	zirconium	Zr	5	4	91,224
41	niobium	Nb	5	5	92,90638
42	molybdene	Mo	5	6	95,94
43	technetium	Tc	5	7	98
44	ruthenium	Ru	5	8	101,07
45	rhodium	Rh	5	9	102,9055
46	palladium	Pd	5	10	106,42
47	argent	Ag	5	11	107,8682
48	cadmium	Cd	5	12	112,411
49	indium	In	5	13	114,818
50	etain	Sn	5	14	118,71
51	Antimoine	Sb	5	15	121,76
52	Tellure	Te	5	16	127,6
53	Iode	I	5	17	126,90447
54	Xenon	Xe	5	18	131,293

Relation « VALENCE »

Col	Couche
1	s
2	s
3	d
4	d
5	d
6	d
7	d
8	d
9	d
10	d
11	d
12	d
13	p
14	p
15	p
16	p
17	p
18	p

Z : Numéro atomique ;
Sym : symbole ;
L : lignes ;
C ou Col : Colonne ;
Couche : Couche de valence

Exercice 4 - Annexe 1

```
class Yaourt:

    def __init__(self,arome,duree):
    # **** Assertions : question 1.a. à compléter sur votre copie
        self.__arome = arome
        self.__duree = duree
        if arome == 'aucun':
            self.__genre = 'nature'
        else:
            self.__genre = 'aromatise'


    # **** Méthode GetArome(self) question 1.c. à compléter sur
    votre copie

    def GetDuree(self):
        return(self.__duree)

    def GetGenre(self):
        return ( self.__genre)

    def SetDuree(self,duree):
    # **** Mutateur de durée
        if duree > 0 :
            self.__duree = duree

    # **** Mutateur d'arôme SetArome(self,arome) - question 2. à
compléter sur votre copie

    def __SetGenre(self,arome):
        if arome == 'aucun':
            self.__genre = 'nature'
        else:
            self.__genre = 'aromatise'
```