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

Centres étrangers - juin 2021 - sujet 1

Exercice 1 (POO - 4 points)

Dans cet exercice, on étudie une méthode de chiffrement de chaînes de caractères alphabétiques. Pour des raisons historiques, cette méthode de chiffrement est appelée *code de César*. On considère que les messages ne contiennent que les lettres capitales de l'alphabet `ABCDEFGHIJKLMNOPQRSTUVWXYZ` et la méthode de chiffrement utilise un nombre entier fixé appelé la clé de chiffrement.

1. Soit la classe `CodeCesar` définie ci-dessous :

```
class CodeCesar:
    def __init__(self, cle):
        self.cle = cle
        self.alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

    def decale(self, lettre):
        num1 = self.alphabet.find(lettre)
        num2 = num1 + self.cle
        if num2 >= 26:
            num2 = num2 - 26
        if num2 < 0:
            num2 = num2 + 26
        nouvelle_lettre = self.alphabet[num2]
        return nouvelle_lettre
```

   On rappelle que la méthode `str.find` renvoie l'indice (index), dans la chaîne de caractères `str`, de la lettre `lettre` passée en paramètre.

   Représenter le résultat d'exécution du code `Python` suivant :

```
code1 = CodeCesar(3)
print(code1.decale('A'))
print(code1.decale('X'))
```

2. La méthode de chiffrement du « code César » consiste à décaler les lettres du message dans l'alphabet d'un nombre de rangs fixé par la clé. Par exemple, avec la clé 3, toutes les lettres sont décalées de 3 rangs vers la droite : le `A` devient le `D`, le `B` devient le `E`, etc.
   Ajouter une méthode `cryptage` dans la classe `CodeCesar` définie à la question précédente, qui reçoit en paramètre une chaîne de caractères `texte` (le message à crypter) et qui retourne une chaîne de caractères (le message crypté). Cette méthode doit crypter la chaîne `texte` avec la clé de l'objet de la classe `CodeCesar` qui a été instancié. Par exemple :

```
>>> code1 = CodeCesar(3)
>>> code1.cryptage("NSI")
'QVL'
```

3. Ecrire un programme qui :
   - ⋆ demande de saisir la clé de chiffrement
   - ⋆ crée un objet de classe `CodeCesar`
   - ⋆ demande de saisir le texte à chiffrer
   - ⋆ affiche le texte chiffré en appelant la méthode `cryptage`
4. On ajoute à la classe `CodeCesar` la méthode `transforme` qui prend en argument une chaîne de caractère `texte` :

```
def transforme(self, texte):
    self.cle = -self.cle
    message = self.cryptage(texte)
    self.cle = -self.cle
    return message
```

On exécute la ligne suivante : `print(CodeCesar(10).transforme("PSX"))`
Que va-t-il s'afficher ? Expliquer votre réponse.

Exercice 2 (Dictionnaires - 4 points)

Une ville souhaite gérer son parc de vélos en location partagée. L'ensemble de la flotte de vélos est stocké dans une table de données représentée en langage `Python` par un dictionnaire contenant des associations de type `id_velo` : `dict_velo` où `id_velo` est un nombre entier compris entre 1 et 199 qui correspond à l'identifiant unique du vélo et `dict_velo` est un dictionnaire dont les clés sont : `"type"`, `"etat"`, `"station"`.

Les valeurs associées aux clés `"type"`, `"etat"`, `"station"` de `dict_velo` sont de type chaînes de caractères ou nombre entier :

- `"type"` : chaîne de caractères qui peut prendre la valeur `"electrique"` ou `"classique"`
- `"état"` : nombre entier qui peut prendre la valeur 1 si le vélo est disponible, 0 si le vélo est en déplacement, -1 si le vélo est en panne
- `"station"` : chaînes de caractères qui identifie la station où est garé le vélo.

Dans le cas où le vélo est en déplacement ou en panne, `"station"` correspond à celle où il a été dernièrement stationné.

Voici un extrait de la table de données :

```
flotte = {12 : {"type" : "electrique", "etat" : 1, "station" : "Prefecture"},
          80 : {"type" : "classique", "etat" : 0, "station" : "Saint-Leu"},
          45 : {"type" : "classique", "etat" : 1, "station" : "Baraban"},
          41 : {"type" : "classique", "etat" : -1, "station" : "Citadelle"},
          26 : {"type" : "classique", "etat" : 1, "station" : "Coliseum"},
          28 : {"type" : "electrique", "etat" : 0, "station" : "Coliseum"},
          74 : {"type" : "electrique", "etat" : 1, "station" : "Jacobins"},
          13 : {"type" : "classique", "etat" : 0, "station" : "Citadelle"},
          83 : {"type" : "classique", "etat" : -1, "station" : "Saint-Leu"},
          22 : {"type" : "electrique", "etat" : -1, "station" : "Joffre"}}
```

`flotte` étant une variable globale du programme.

Toutes les questions de cet exercice se réfèrent à l'extrait de la table `flotte` fourni cidessus. L'annexe 1 présente un rappel sur les dictionnaires en langage `Python`.

1. (a) Que renvoie l'instruction `flotte[26]` ?
   (b) Que renvoie l'instruction `flotte[80]["etat"]` ?
   (c) Que renvoie l'instruction `flotte[99]["etat"]` ?
2. Voici le script d'une fonction :

```
def proposition(choix):
    for v in flotte:
        if flotte[v]["type"] == choix and flotte[v]["etat"] == 1:
            return flotte[v]["station"]
```

   (a) Quelles sont les valeurs possibles de la variable `choix` ?
   (b) Expliquer ce que renvoie la fonction lorsque l'on choisit comme paramètre l'une des valeurs possibles de la variable `choix`.
3. (a) Ecrire un script en langage `Python` qui affiche les identifiants (`id_velo`) de tous les vélos disponibles à la station `"Citadelle"`.
   (b) Ecrire un script en langage `Python` qui permet d'afficher l'identifiant (`id_velo`) et la station de tous les vélos électriques qui ne sont pas en panne.
4. On dispose d'une table de données des positions GPS de toutes les stations, dont un extrait est donné ci-dessous. Cette table est stockée sous forme d'un dictionnaire. Chaque élément du dictionnaire est du type : `'nom de la station' : (latitude, longitude)`

```
stations = {'Prefecture' : (49.8905, 2.2967),
            'Saint-Leu' : (49.8982, 2.3017),
            'Coliseum' : (49.8942, 2.2874),
            'Jacobins' : (49.8912, 2.3016)}
```

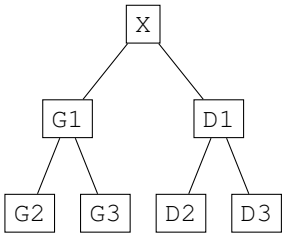
   On admet que l'on dispose d'une fonction `distance` prenant en paramètre deux tuples `d1` et `d2` données par leur coordonnées GPS, et renvoyant un nombre entier correspondant à distance en mètres entre deux positions. Par exemple, l'instruction

`distance((49.8905, 2.2967), (49.8912, 2.3016))` renvoie `9591`

Ecrire une fonction qui prend en paramètre les coordonnées GPS de l'utilisateur sous forme d'un tuple et qui renvoie, pour chaque station située à moins de 800 mètres de l'utilisateur : le nom de la station, la distance entre l'utilisateur et la station, et les identifiants des vélos disponibles dans cette station.

Une station où aucun vélo n'est disponible ne doit pas être renvoyée.

Exercice 3 (Arbres binaires de recherche - 4 points)
Un arbre binaire est soit vide, soit un nœud qui a une valeur et au plus deux fils (le sous-arbre gauche et le sous-arbre droit).



`X` est un nœud, sa valeur est `X.valeur`

`G1` est le fils gauche de `X`, noté `X.fils_gauche`

`D1` est le fils droit de `X`, noté `X.fils_droit`

Un arbre binaire de recherche est ordonné de la manière suivante. Pour chaque nœud `X` :

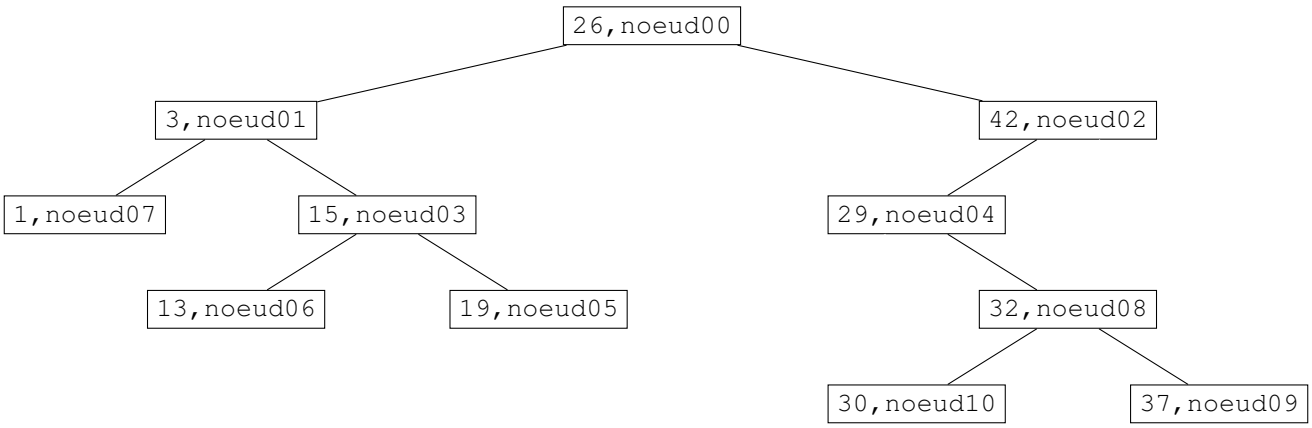
- les valeurs de tous les nœuds du sous-arbre gauche sont strictement inférieures à la valeur du nœud `X`;
- les valeurs de tous les nœuds du sous-arbre droit sont supérieures ou égales à la valeur du nœud `X`.

Ainsi, par exemple, toutes les valeurs des nœuds `G1`, `G2` et `G3` sont strictement inférieures à la valeur du nœud `X` et toutes les valeurs des nœuds `D1`, `D2` et `D3` sont supérieures ou égales à la valeur du nœud `X`.

Voici un exemple d'arbre binaire de recherche dans lequel on a stocké dans cet ordre les valeurs :

```
[26, 3, 42, 15, 29, 19, 13, 1, 32, 37, 30]
```

L'étiquette d'un nœud indique la valeur du nœud suivie du nom du nœud.
Les nœuds ont été nommés dans l'ordre de leur insertion dans l'arbre ci-dessous.
`'29,noeud04'` signifie que le nœud nommé `noeud04` possède la valeur `29`.



1. On insère la valeur 25 dans l'arbre, dans un nouveau nœud nommé `noeud11`.
   Recopier l'arbre binaire de recherche étudié et placer la valeur 25 sur cet arbre en coloriant en rouge le chemin parcouru.
   Préciser sous quel nœud la valeur 25 sera insérée et si elle est insérée en fils gauche ou en fils droit, et expliquer toutes les étapes de la décision.
2. Préciser toutes les valeurs entières que l'on peut stocker dans le nœud fils gauche du `noeud04` (vide pour l'instant), en respectant les règles sur les arbres binaires de recherche.
3. Voici un algorithme récursif permettant de parcourir et d'afficher les valeurs de l'arbre :

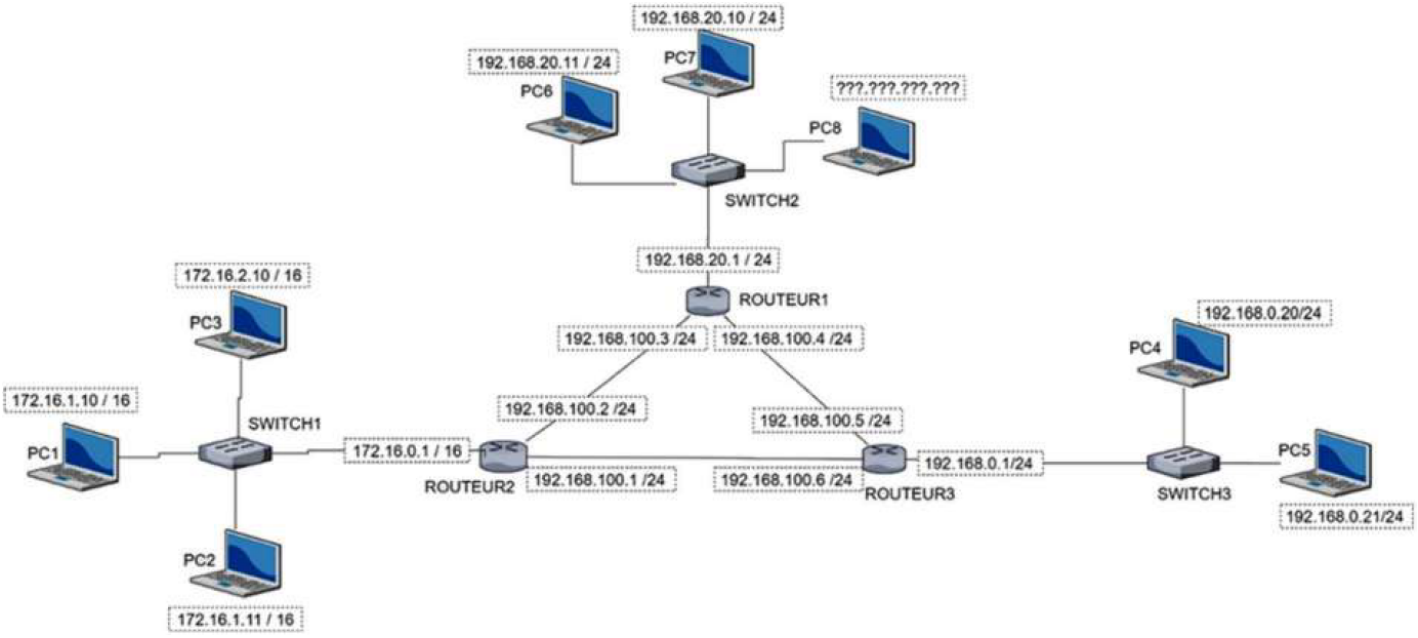
```
fonction Parcours(A)
    # A est un arbre binaire de recherche
    Afficher(A.valeur)
    Parcours(A.fils_gauche)
    Parcours(A.fils_droit)
fin fonction
```

   (a) Ecrire la liste de toutes les valeurs dans l'ordre où elles seront affichées.
   (b) Choisir le type de parcours d'arbres binaires de recherche réalisé parmi les propositions suivantes : préfixe, suffixe ou infixe.
4. En vous inspirant de l'algorithme précédent, écrire un algorithme `Parcours2` permettant de parcourir et d'afficher les valeurs de l'arbre `A` dans l'ordre croissant.

Exercice 4 (Réseaux - 4 points)
Soit un réseau informatique dont le schéma structurel simplifié est représenté cidessous. Il est composé de 8 PC, 3 switchs, et 3 routeurs.

Dans cet exercice, on utilisera l'adressage CIDR composé d'une adresse IPv4 et d'une indication sur le masque de sous réseau.
Par exemple : `172.16.1.10/16` signifie :

- Adresse IP : 172.16.1.10
- Masque de sous-réseau en notation CIDR : 16



Partie A : étude de l'adressage IP

1. Sur l'annexe 2, encadrer tous les sous-réseaux présents dans le réseau global donné le document réponse.
2. Etude du PC7 dont l'adresse IP est : `192.168.20.10/24`
   (a) Combien d'octets sont nécessaires pour composer une adresse IP(V4) ?
   (b) Compléter la ligne 2 du tableau de l'annexe 3 en convertissant la notation décimale de l'adresse IP en notation binaire.
   La notation CIDR `/16` pour une adresse IP signifie que le masque de sous-réseau a les 16 bits de poids fort de son adresse IP à la valeur `1`, c'est-à-dire : `1111 1111.1111 1111.0000 0000.0000 0000`
   (c) Compléter la ligne 3 du tableau de l'annexe 3 en donnant le codage binaire du masque de sous-réseau en notation CIDR `/24`.
   (d) En déduire, à la ligne 4 du tableau de l'annexe 3, l'écriture décimale pointée du masque de sous-réseau.
   (e) On rappelle que l'adresse du réseau s'obtient en réalisant un `ET` logique bit à bit entre l'adresse IP du PC7 et le masque de sous-réseau.
      i. Compléter la ligne 5 du tableau de l'annexe 3 avec l'adresse binaire du réseau.
      ii. Compléter la ligne 6 du tableau de l'annexe 3 avec l'adresse décimale du réseau.
3. Connexion du PC8 au réseau : répondre au questionnaire donné dans l'annexe 3 en cochant la ou les bonnes réponses.

Partie B : une fonction pour convertir une adresse IP en décimal pointé en notation binaire.
On dispose de la fonction `dec_bin` :

- qui prend en paramètre d'entrée un nombre entier compris entre 0 et 255
- qui renvoie une liste de huit éléments correspondant à la conversion du nombre en écriture décimale en notation binaire. Chaque élément de cette liste est de type entier.

Exemples d'exécution de la fonction `dec_bin` :

- `dec_bin(10)` renvoie la liste `[0,0,0,0,1,0,1,0]`
- `dec_bin(255)` renvoie la liste `[1,1,1,1,1,1,1,1]`

Ecrire une fonction en langage `Python` que l'on appellera `IP_bin` qui :

- prend en paramètre d'entrée une liste de quatre entiers compris entre 0 et 255 correspondant à l'adresse IP en notation décimale
- renvoie une liste de quatre listes correspondant à l'adresse IP en notation binaire.

La fonction `IP_bin` peut faire appel à la fonction `dec_bin`.

Exemple d'exécution de la fonction `IP_bin` :

```
>>> IP_bin([192,168,0,1])
[[1,1,0,0,0,0,0,0],[1,0,1,0,1,0,0,0],[0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,1]]
```

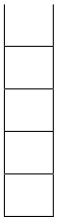
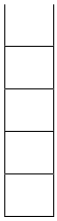
Exercice 5 (Piles - 4 points)

Dans cet exercice, on considère une pile d'entiers positifs. On suppose que les quatre fonctions suivantes ont été programmées préalablement en langage `Python` :

- `empiler(P, e)` : ajoute l'élément `e` sur la pile `P` ;
- `depiler(P)` : enlève le sommet de la pile `P` et renvoie la valeur de ce sommet ;
- `est_vide(P)` : renvoie `True` si la pile `P` est vide et `False` sinon ;
- `creer_pile()` : renvoie une pile vide.

Dans cet exercice, seule l'utilisation de ces quatre fonctions sur la structure de données pile est autorisée.

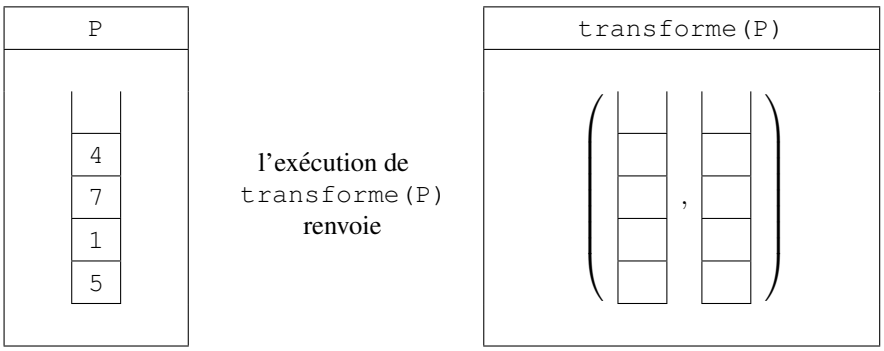
1. Recopier le schéma ci-dessous et le compléter sur votre copie en exécutant les appels de fonctions donnés. On écrira ce que renvoie la fonction utilisée dans chaque cas, et on indiquera `None` si la fonction ne renvoie aucune valeur.

	Etape 0 Pile d'origine `P`	Etape 1 `empiler(P,8)`	Etape 2 `depiler(P)`	Etape 3 `est_vide(P)`
				
Valeur renvoyée		.....	.....	.....

2. On propose la fonction ci-dessous, qui prend en argument une pile `P` et renvoie un couple de piles :

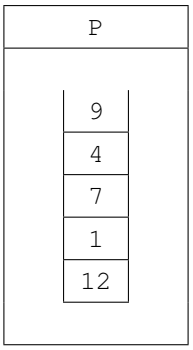
```
def transforme(P):
    Q = creer_pile()
    while not est_vide(P):
        v = depile(P)
        empile(Q,v)
    return (P,Q)
```

Recopier et compléter sur votre copie le document ci-dessous



3. Ecrire en langage `Python` une fonction `maximum` recevant une pile `P` comme argument et renvoyant la valeur maximale de cette pile. On ne s'interdit pas qu'après exécution de la fonction, la pile `P` soit vide.

On souhaite à présent connaître le nombre d'éléments d'une pile `P` à l'aide de la fonction `taille` recevant cette pile en paramètre. Par exemple, l'exécution de `taille(P)` avec la pile `P` ci-dessous renvoie l'entier `5`.



4. (a) Proposer une stratégie écrite en langage naturel et/ou expliquée à l'aide de schémas, qui permette de mettre en place une telle fonction.
   (b) Donner le code `Python` de cette fonction `taille(P)` (on pourra utiliser les cinq fonctions déjà programmées).

Annexe 1 (exercice 2)

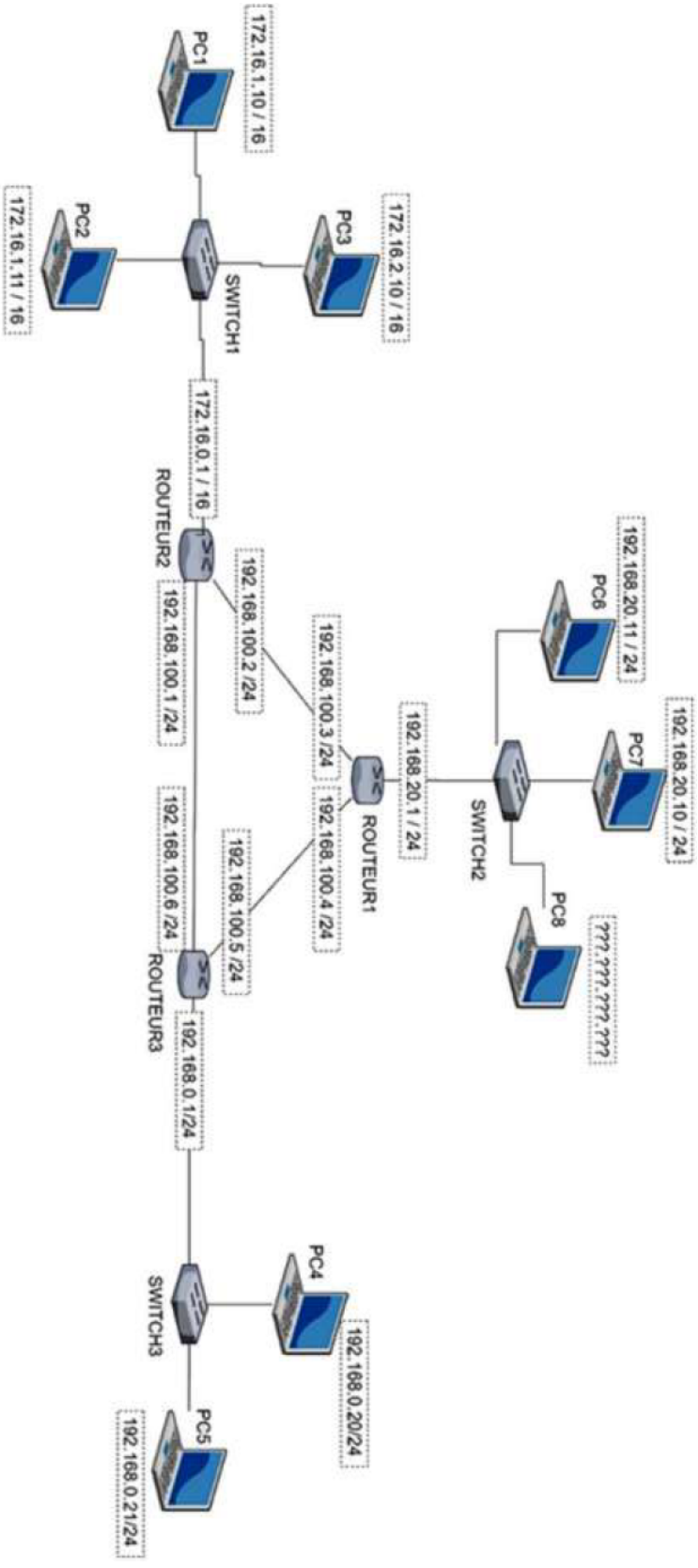
(à ne pas rendre avec la copie)

Action	**Instruction et syntaxe**
Créer un dictionnaire vide	`dico = {}`
Obtenir un élément d'un dictionnaire existant à partir de sa clé Renvoie une erreur si la clé n'existe pas dans le dictionnaire	`dico[cle]`
Modifier la valeur d'un élément d'un dictionnaire à partir de sa clé	`dico[cle] = nouvelle_valeur`
Ajouter un élément dans un dictionnaire existant	`dico[nouvelle_cle] = valeur`
Supprimer et obtenir un élément d'un dictionnaire à partir de sa clé	`dico.pop(cle)`
Tester l'appartenance d'un élément à un dictionnaire (renvoie un booléen)	`cle in dico`
Objet itérable contenant les clés (ce n'est pas un objet de type `list`)	`dico.keys()`
Objet itérable contenant les valeurs (ce n'est pas un objet de type `list`)	`dico.values()`
Objet itérable contenant les couples (clé,valeur)	`dico.items()`
Afficher les associations clé :valeur du dictionnaire `dico`	`for cle in dico:` `  print(cle, dico[cle])`

Annexe 2 (exercice 4)

(à rendre avec la copie)

Partie A, question 1



Annexe 3 (exercice 4)

(à rendre avec la copie)

Partie A, question 2

Adresse IP (V4) du PC7	192								168								20								10							
	1	1	0	0	0	0	0	0									0	0	0	1	0	1	0	0								
Masque de sous-réseau																																
	255																															
Pour obtenir l'adresse réseau binaire, on réalise un `ET` logique entre chaque bit de l'adresse IP (ligne 2) et du masque de sous-réseau (ligne 3)																																
Adresse du réseau																																
									168																							

Partie A, question 3
On désire connecter le PC8 au réseau précédent. Parmi les propositions suivantes, cochez les adresses IP possibles pour le PC8 :

- ☐ `192.168.20.0`
- ☐ `192.256.20.11`
- ☐ `192.168.20.30`
- ☐ `192.168.20.230`
- ☐ `192.168.20.260`
- ☐ `192.168.27.11`