BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2021

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 16 pages numérotées de 1/16 à 16/16.

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

Exercice 1

Notion abordée : programmation objet.

Cryptage selon le « Code de César »

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(lettre)` renvoie l'indice (index) de la `lettre` dans la chaîne de caractères `str`

Représenter le résultat d'exécution du code Python suivant :

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

1. On applique la méthode decale('A') sur l'instance de classe code1 :
   - code1.cle = 3
   - num1 = 0
   - num2 = 0 + 3 = 3
   - nouvelle_lettre = ''ABCDEFGHIJKLMNOPQRSTUVWXYZ''[3] = 'D'
   - l'affichage donne la lettre 'D'

   On applique la méthode decale('X') sur l'instance de classe code1 :
   - code1.cle = 3
   - num1 = 23
   - num2 = 23 + 3 = 26
   - num2 = 26 – 26 = 0
   - nouvelle_lettre = ''ABCDEFGHIJKLMNOPQRSTUVWXYZ''[0] = 'A'
   - l'affichage donne la lettre 'A'

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(self, texte)` dans la classe `CodeCesar` définie à la question précédente, qui reçoit en paramètre une chaîne de caractères (le message à crypter) et qui retourne une chaîne de caractères (le message crypté).

Cette méthode `cryptage(self, texte)` doit crypter la chaîne `texte` avec la clé de l'objet de la classe `CodeCesar` qui a été instancié.

Exemple :

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

2. On utilise la méthode précédente. Cela donne :

```
def cryptage(self,texte) :
    sortie = ""
    for c in texte :
        sortie += self.decale(c)
    return sortie
```

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`

3. Le programme est :

```
cle = int(input("Saisir le clé de chiffrement : "))
code = CodeCesar(cle)
texte = input("Saisir le texte à afficher : ")
print(code.cryptage(texte))
```

4. On ajoute la méthode `transforme(texte)` à la classe `CodeCesar` :

```
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.

4. On applique la méthode transforme(' PSX') avec une clé égale à 10. Ce qui donne :
   - cle = -10 , d'où un décalage de 10 vers la gauche
   - pour 'P' : num1 = 15 et donc num2 = 5 et 'P' devient 'F'
   - pour 'S' : num1 = 18 et donc num2 = 8 et 'S' devient 'I'
   - pour 'X' : num1 = 23 et donc num2 = 13 et 'X' devient 'N'
   - message = 'FIN'

L'exécution de la ligne entraine donc l'affichage du mot 'FIN'.

Exercice 2

Notion abordée : structures de données (dictionnaires)

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.

1.

1.a. Que renvoie l'instruction `flotte[26]` ?
1.b. Que renvoie l'instruction `flotte[80]["etat"]` ?
1.c. Que renvoie l'instruction `flotte[99]["etat"]` ?

1. flotte[26] = {'type' : 'classique' , 'etat' : 1 , 'station' :'Coliseum }
   flotte[80]['etat'] = 0
   flotte[99]['etat] renvoie une erreur car le dictionnaire flotte ,'a pas de clé égale à 99.

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"]
```

2.a. **Quelles sont** les valeurs possibles de la variable `choix` ?
2.b. **Expliquer** ce que renvoie la fonction lorsque l'on choisit comme paramètre l'une des valeurs possibles de la variable `choix`.

2. Dans cette fonction, la variable v prend comme valeurs les clés du dictionnaire : 12, 80, 45, ….
   a. La variable choix peut à priori prendre n'importe quelle valeur. Pour que la condition flotte[v]['type'] == choix puisse être vraie choix doit prendre comme valeur 'electrique' ou 'classique'.

b. Si choix = ‘electrique’ , la fonction renvoie ‘Prefecture’. Si choix = ‘classique’, la fonction renvoie ‘Baraban’ . Le premier élément du dictionnaire qui satisfait aux 2 conditions entraine un arrêt du script de par la présence de l’instruction return. Si choix pren une valeur différente, la fonction ne renvoie rien.

3.

3.a. Écrire un script en langage Python qui affiche les identifiants (`id_velo`) de tous les vélos disponibles à la station "`Citadelle`".
3.b. Écrire 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.

3. On a :
   a. Le script peut être le suivant :

```
for id_velo in flotte :
    if flotte[id_velo]['station'] == 'Citadelle' :
        print(id_velo)
```

   b. Le script peut être le suivant :

```
for id_velo in flotte :
    if flotte[id_velo]['etat'] != -1 :
        print(id_velo , flotte[id_velo]['station'])
```

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(p1, p2)` permettant de renvoyer la distance en mètres entre deux positions données par leurs coordonnées GPS (latitude et longitude).

Cette fonction prend en paramètre deux tuples représentant les coordonnées des deux positions GPS et renvoie un nombre entier représentant cette distance en mètres.

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

Écrire 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 ;
- les identifiants des vélos disponibles dans cette station.

Une station où aucun vélo n’est disponible ne doit pas être affichée.

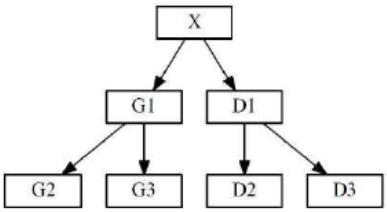
4. Un script possible de cette fonction est :

```
def plus_proche(p_utilisateur) :
    for s in stations :
        p = stations[s]
        d = distance(p_utilisateur , p)
        temp = []
        if d < 800 :
            for id_velo in flotte :
                if flotte[id_velo]['etat'] == 1 and flotte[id_velo]['station'] == s :
                    temp.append(id_velo)
            if temp != [] :
                print(f"station : {s} ; distance = {d} ; vélos libres : " , end=" ")
                for id_velo in temp : print(id_velo , end = " ")
                print()
```

Exercice 3

Notion abordée : les arbres binaires de recherche.

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.

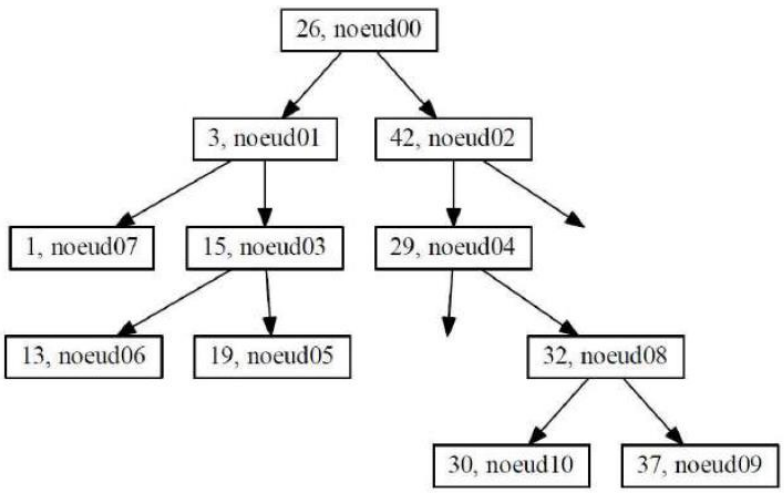
Voici un exemple d'arbre binaire de recherche dans lequel on a stocké dans cet ordre les valeurs :

[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.

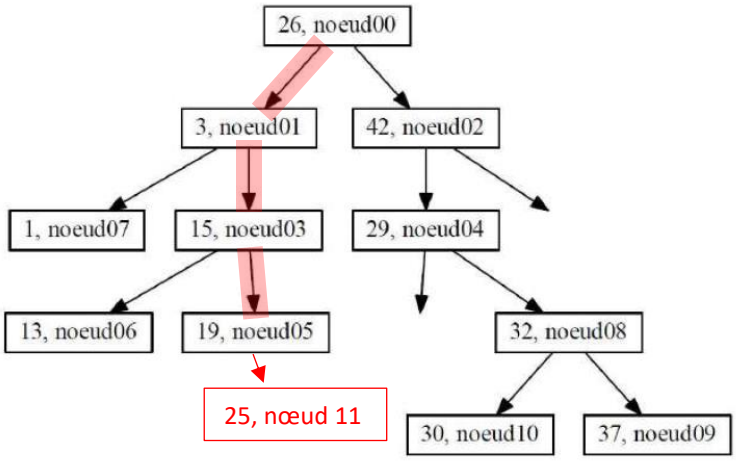
Les nœuds ont été nommés dans l'ordre de leur insertion dans l'arbre ci-dessous.

`'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 nœud11.
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.

1. On a :



En partant de la racine , les étapes du placement sont :

- 25 < 26 , donc 25 est placé dans le sous-arbre fils_gauche de 26
- 25 > 3 , donc 25 est placé dans le sous-arbre fils_droit de 3
- 25 > 15 , donc 25 est placé dans le sous-arbre fils_droit de 15
- 25 > 19 , donc 25 est placé dans le sous-arbre fils_droit de 19

2. **Préciser** toutes les valeurs entières que l'on peut stocker dans le nœud fils gauche du nœud04 (vide pour l'instant), en respectant les règles sur les arbres binaires de recherche ?

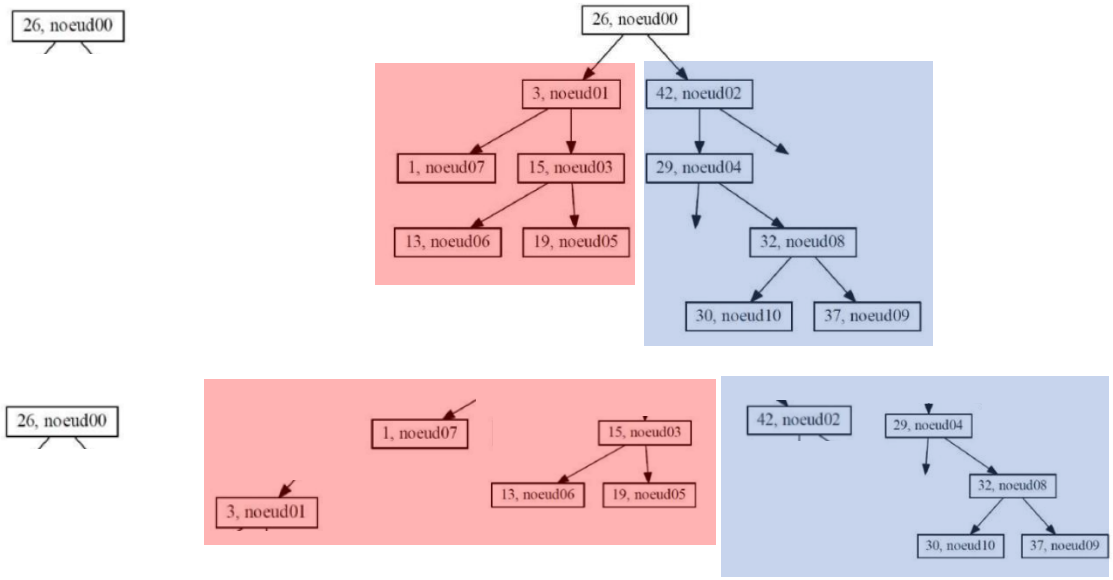
2. Le nœud 04 a la valeur 29. La valeur du nœud fils_gauche du nœud 04 doit donc être strictement inférieure à 29. Le nœud 04 est fils_gauche du nœud 02, qui est lui-même fis_droit du nœud racine de valeur 26. Ainsi la valeur du nœud fils_gauche du nœud 04 doit être strictement supérieure à 26. Les valeurs entières possibles sont ainsi : 27 et 28.

3. Voici un algorithme récursif permettant de parcourir et d'afficher les valeurs de l'arbre :

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

3.a. **Écrire** la liste de toutes les valeurs dans l'ordre où elles seront affichées.
3.b. **Choisir** le type de parcours d'arbres binaires de recherche réalisé parmi les propositions suivantes : Préfixe, Suffixe ou Infixe

3. a. On affiche d'abord la valeur du nœud racine, puis on a une récursivité sur le sous-arbre gauche, puis sur le sous-arbre-droit. Cela donne :



Les valeurs des nœuds visités seront dans l'ordre :

26 3 1 15 13 19 42 29 32 30 37

On a donc la liste qui est : 26, 3 ,1 ,15 ,13 ,19 ,42 ,29 ,32 ,30 ,37

4. b. Le parcours de l'arbre est ici un parcours **préfixe**.

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.

5. Pour afficher les valeurs dans l'ordre croissant, on reprend l'algorithme précédent en ayant une récursivité sur le sous-arbre fils_gauche, puis en affichant la valeur du nœud racine, puis en ayant une récursivité sur le sous-arbre fils-droit. On a donc :
Ce parcours donne ici : 1, 3 ; 13, 15, 19, 26, 29, 30, 32, 37, 42

```
Parcours2(A)
    Parcours(A.fils_gauche)
    Afficher(A.valeur)
    Parcours(A.fils_droit)
```

Etude d'un réseau informatique

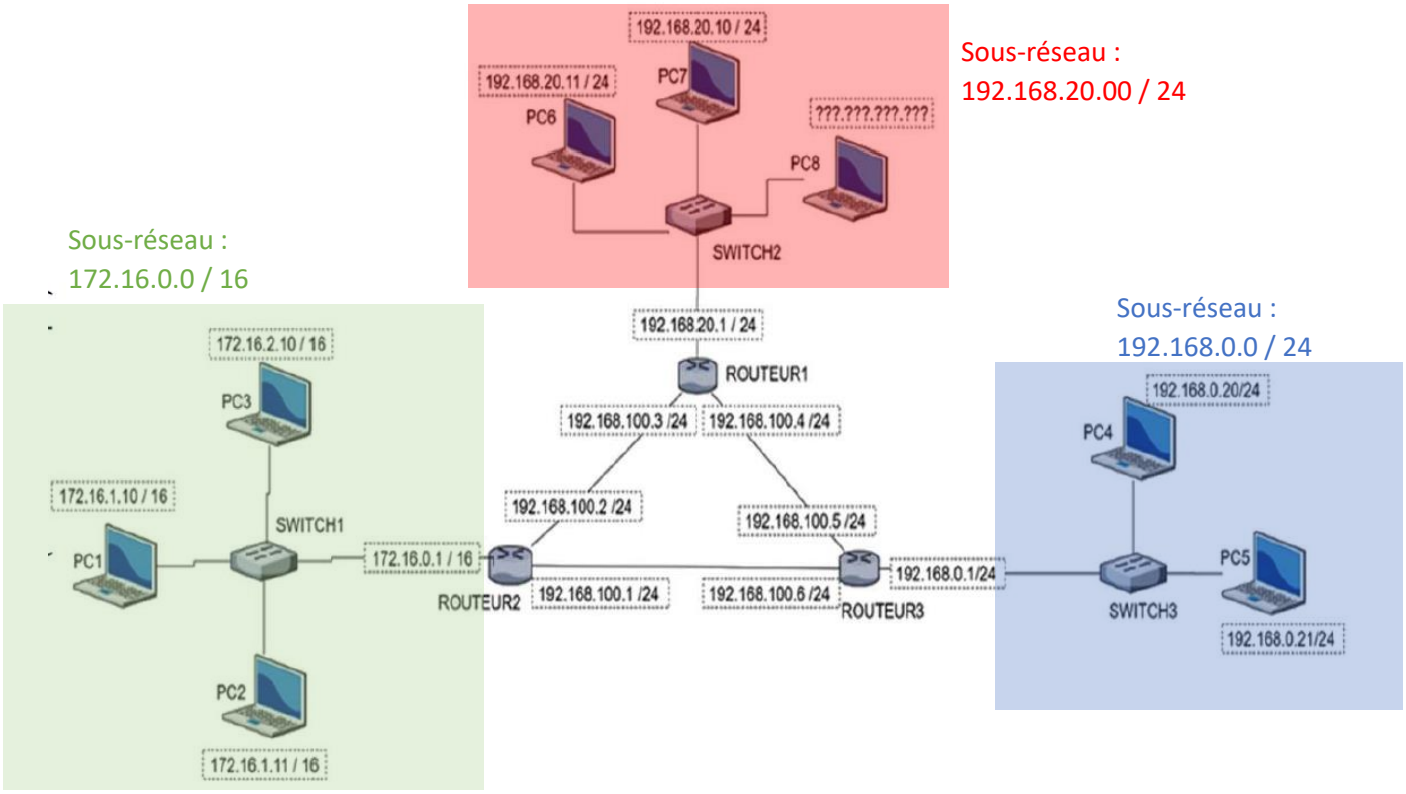
Soit un réseau informatique dont le schéma structurel simplifié est représenté ci-dessous. Il est composé de 8 PC, 3 switchs, et 3 routeurs.

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 : ETUDE DE L'ADRESSAGE IP

1. Sur le document réponse 1 de l'exercice 4, encadrer tous les sous-réseaux présents dans le réseau global sur le document réponse.



2. Etude du PC7 dont l'adresse IP est : 192.168.20.10 / 24.

2.a. Combien d'octets sont nécessaires pour composer une adresse IP(V4) ?

2.b. Compléter la ligne 2 du tableau du document réponse 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: 11111111.11111111.00000000.00000000.

2.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.

2.d. En déduire, à la ligne 4 du tableau de l'annexe, l'écriture décimale pointée du masque de sous-réseau.

L'adresse du réseau peut s'obtenir 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 2 avec l'adresse binaire du réseau.

ii. Compléter la ligne 6 du tableau avec l'adresse décimale du réseau.

2. On a :
   a. Une adresse Ip(v4) est composée de 4 octets.
   b. On complète le document réponse :
   168 = **1** x 128 + **0** x 64 + **1** x 32 + **0** x 16 + **1** x 8 + **0** x 4 + **0** x 2 + **0** x 1 . Donc $(168)_{10} = (1010\ 1000)_2$
   10 = **0** x 128 + **0** x 64 + **0** x 32 + **0** x 16 + **1** x 8 + **0** x 4 + **1** x 2 + **0** x 1 . Donc $(168)_{10} = (0000\ 1010)_2$
   255 = **1** x 128 + **1** x 64 + **1** x 32 + **1** x 16 + **1** x 8 + **1** x 4 + **1** x 2 + **1** x 1 . Donc $(255)_{10} = (1111\ 1111)_2$

Adresse IP (V4) du PC7	Ligne 1	192								168								20								10					
	Ligne 2	1	1	0	0	0	0	0	0	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	0	0	0	1	0	1	0	0	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**0**
Masque de sous réseau	Ligne 3	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	Ligne 4	255								**255**								**255**								**0**							
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 (ligne3)																																	
Adresse du réseau	Ligne 5	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	Ligne 6	**192**								168								**20**								**0**							

3. Connexion du PC8 au réseau
Répondre au questionnaire sur le document réponse joint en cochant la ou les bonnes réponses.

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

- [X] 192.168.20.0
- [ ] 192.256.20.11
- [X] 192.168.20.30
- [X] 192.168.20.230
- [ ] 192.168.20.260
- [ ] 192.168.27.11

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 retourne une **liste** de 8 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)` retourne la liste `[0,0,0,0,1,0,1,0]`
  - `dec_bin(255)` retourne 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 4 entiers compris entre 0 et 255 correspondant à l'adresse IP en notation décimale
- **retourne** une liste de 4 listes correspondant à l'adresse IP en notation binaire.

Si on ajoute une série d'assertion, on peut obtenir :

```
def IP_bin(ip_dec) :
    assert type(ip_dec) == list , "L'argument doit être une liste"
    assert len(ip_dec) == 4 , "La liste en argument n'a pas le bon format"
    assert ip_dec[0] < 256 , "La liste en argument n'a pas le bon format"
    assert ip_dec[1] < 256 , "La liste en argument n'a pas le bon format"
    assert ip_dec[2] < 256 , "La liste en argument n'a pas le bon format"
    assert ip_dec[3] < 256 , "La liste en argument n'a pas le bon format"
    ip_bin = []
    for n_dec in ip_dec :
        n_bin = dec_bin(n_dec)
        ip_bin.append(n_bin)
    return ip_bin
```

Exercice 5

Notion abordée : structures de données : les piles.

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 retourne la valeur de ce sommet ;
`est_vide(P)` : retourne `True` si la pile est vide et `False` sinon ;
`creer_pile()` : retourne une pile vide.

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

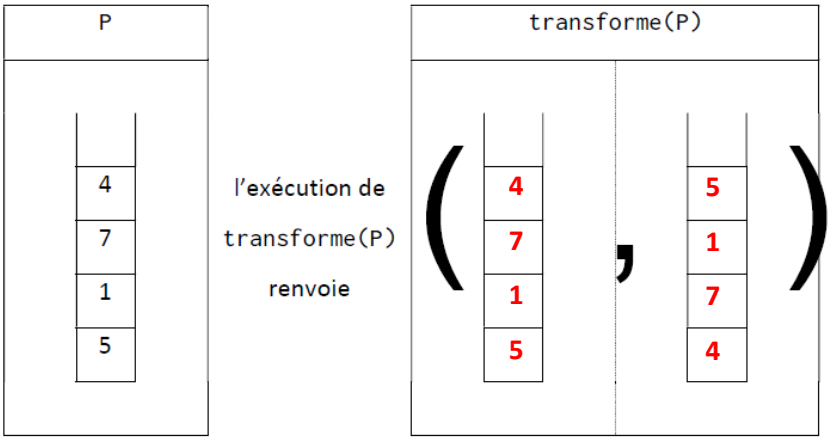
1. **Recopier** le schéma ci-dessous et le compléter sur votre copie en exécutant les appels de fonctions donnés. On écrira ce que renvoie la fonction utilisée dans chaque cas, et on indiquera `None` si la fonction ne retourne aucune valeur.

	Etape 0 Pile d'origine P	Etape 1 empiler(P,8)	Etape 2 depiler(P)	Etape 3 est_vide(P)
	4 7 1 5	8 4 7 1 5	4 7 1 5	4 7 1 5
Retour de la fonction		..... **None**	..... **8**	..... **False**

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

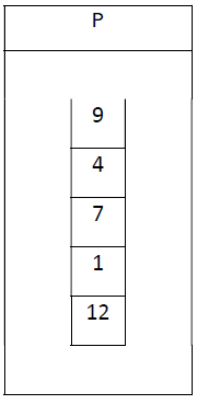
```
def transforme(P) :
  Q = creer_pile()
  while not est_vide(P) :
    v = depile(P)
    empile(Q,v)
  return (P,Q)
```

Recopier et compléter sur votre copie le document ci-dessous



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

On souhaite connaître le nombre d'éléments d'une pile à l'aide de la fonction `taille(P)`



`taille(P)` retournera donc l'entier 5

4.

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.

4.b. Donner le code Python de cette fonction `taille(P)` (on pourra utiliser les cinq fonctions déjà programmées).

Codes proposées :

```
def maximum(P) :
    Q = creer_pile()
    if est_vide(P) : return
    v = depile(P)
    empile(Q,v)
    max = v
    while not est_vide(P) :
        v = depile(P)
        empile(Q,v)
        if v > max : max = v
    return max
```

```
def taille(P) :
    n = 0
    Q = creer_pile()
    while not est_vide(P) :
        v = depile(P)
        empile(Q,v)
        n += 1
    while not est_vide(Q) :
        v = depile(Q)
        empile(P,v)
    return n
```