La team DarkSATHI Li est dans la place.

Le jour du BAC ils vont tout

Your padawan name :

Répartition des tâches

	Classe	**Exercice** écrit	Sujet pratique(2022)
BRILLON Baptiste	TF	1	2
CARDON Dimitri	TH	2	4
CARTIGNIES Émély	TA	1	6
CHEVALIER Louis	TA	3	36
COLEAU Céline	TH	2	37
DELDALLE Corentin	TC	1	38
DELOEIL Timeo	TF	2	39
DEVIGNE Maxence	TB	3	40
DREUMONT Alessio	TB	1	2
DUBOIS Lucas	TH	2	4
DUEZ Alexis	TE	3	6
GRANSART Corentin	TG	1	36
HERBIN Thomas	TC	2	37
LEGRAND Rayan	TE	3	38
LOBRY Anthony	TB	2	39
NAFTEUR Alan	TG	3	40
ROUX Benjamin	TG	1	2

Les sujets de l'épreuve pratique sont téléchargeables depuis l'espace public de votre classe.

Exercice 1

Partie A : Expression correctement parenthésée

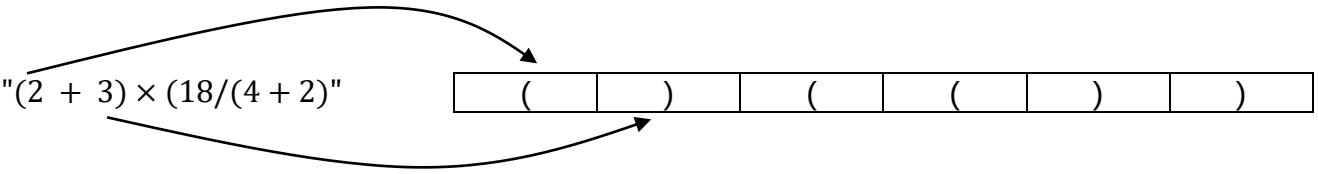
On veut déterminer si une expression arithmétique est correctement parenthésée.

Pour chaque parenthèse fermante "$)$" correspond une parenthèse précédemment ouverte "$($".

Exemples :

- L'expression arithmétique "$(2 + 3) \times (18/(4 + 2))$" est correctement parenthésée.
- L'expression arithmétique "$(2 + 3) \times (18/(4 + 2$" est non correctement parenthésée.

Pour simplifier les expressions arithmétiques, on enregistre, dans une structure de données, uniquement les parenthèses dans leur ordre d'apparition. On appelle expression simplifiée cette structure.



1 Indiquer si la phrase « les éléments sont maintenant retirés (pour être lus) de cette structure de données dans le même ordre qu'ils y ont été ajoutés lors de l'enregistrement » décrit le comportement d'une file ou d'une pile. Justifier.
Pour vérifier le parenthésage, on peut utiliser une variable `controleur` qui :

- est un nombre entier égal à 0 en début d'analyse de l'expression simplifiée ;
- augmente de 1 si l'on rencontre une parenthèse ouvrante "$($" ;
- diminue de 1 si l'on rencontre une parenthèse fermante "$)$".

Exemple : On considère l' expression simplifiée A : "$()(())$" Lors de l'analyse de l'expression A, `controleur` (initialement égal à 0) prend successivement pour valeur 1, 0, 1, 2, 1, 0. Le parenthésage est correct.

2 Écrire, pour chacune des 2 expressions simplifiées B et C suivantes, les valeurs successives prises par la variable `controleur` lors de leur analyse.
Expression simplifiée B : "$(((()())$"
Expression simplifiée C : "$(()))($"

3 L'expression simplifiée B précédente est mal parenthésée (parenthèses fermantes manquantes) car le `controleur` est différent de zéro en fin d'analyse.
L'expression simplifiée C précédente est également mal parenthésée (parenthèse fermante sans parenthèse ouvrante) car le `controleur` prend une valeur négative pendant l'analyse. Recopier et compléter uniquement les lignes 13 et 16 du code ci-dessous pour que la fonction `parenthesage_correct` réponde à sa description.

```
1  def parenthesage_correct(expression):
2      ''' fonction retournant True si l'expression arithmétique
3      simplifiée (str) est correctement parenthésée, False
4      sinon.
5      Condition: expression ne contient que des parenthèses
6      ouvrantes et fermantes'''
7      controleur = 0
8      for parenthese in expression: #pour chaque parenthèse
9          if parenthese == '(':
10             controleur = controleur + 1
11         else:# parenthese == ')'
12             controleur = controleur - 1
13             if controleur ... : # test 1 (à recopier et compléter)
14                 #parenthèse fermante sans parenthèse ouvrante
15                 return False
16     if controleur ... : # test 2 (à recopier et compléter)
17         return True #le parenthésage est correct
18     else:
19         return False #parenthèse(s) fermante(s) manquante(s)
```

Partie B : Texte correctement balisé

On peut faire l'analogie entre le texte simplifié des fichiers HTML (uniquement constitué de balises ouvrantes et fermantes) et les expressions parenthésées :

Par exemple, l'expression HTML simplifiée :
" < p >< strong >< em ></p > " est correctement balisée.

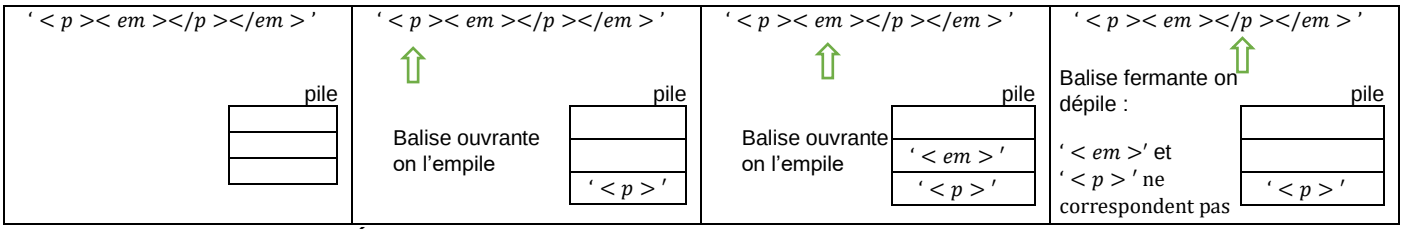
On ne tiendra pas compte dans cette partie des balises ne comportant pas de fermeture comme < br > ou < img ... >.

Afin de vérifier qu'une expression HTML simplifiée est correctement balisée, on peut utiliser une pile (initialement vide) selon l'algorithme suivant :

On parcourt successivement chaque balise de l'expression :

- lorsque l'on rencontre une balise ouvrante, on l'empile ;
- lorsque l'on rencontre une balise fermante :
  - si la pile est vide, alors l'analyse s'arrête : le balisage est incorrect ,
  - sinon, on dépile et on vérifie que les deux balises (la balise fermante rencontrée et la balise ouvrante dépilée) correspondent (c'est-à-dire ont le même nom) si ce n'est pas le cas, l'analyse s'arrête (balisage incorrect).

Exemple : État de la pile lors du déroulement de cet algorithme pour l'expression simplifiée
" < p >< em ></p > " qui n'est pas correctement balisée.



État de la pile lors du déroulement de l'algorithme

4. Cette question traite de l'état de la pile lors du déroulement de l'algorithme.
    a. Représenter la pile à chaque étape du déroulement de cet algorithme pour l'expression " < p >< em ></p > " (balisage correct).
    b. Indiquer quelle condition simple (sur le contenu de la pile) permet alors de dire que le balisage est correct lorsque toute l'expression HTML simplifiée a été entièrement parcourue, sans que l'analyse ne s'arrête.

5. Une expression HTML correctement balisée contient 12 balises. Indiquer le nombre d'éléments que pourrait contenir au maximum la pile lors de son analyse.

Exercice 2

1. Voici la définition d'une classe nommée `ArbreBinaire`, en Python :

```
1  class ArbreBinaire :
2      '''Construit un arbre binaire'''
3
4      def __init__(self, valeur):
5          '''Crée une instance correspondant à un état initial'''
6          self.valeur = valeur
7          self.enfant_gauche = None
8          self.enfant_droit = None
9
10     def insert_gauche(self, valeur):
11         '''Insérer valeur comme fils gauche'''
12         if self.enfant_gauche in None:
13             self.enfant_gauche = ArbreBinaire(valeur)
14         else :
15             new_node = ArbreBinaire(valeur)
16             new_node.enfant_gauche = self.enfant_gauche
17             self.enfant_gauche = new_node
18
19     def insert_droit(self, valeur):
20         '''Insérer valeur comme fils droit'''
21         if self.enfant_droit in None:
22             self.enfant_droit = ArbreBinaire(valeur)
23         else :
24             new_node = ArbreBinaire(valeur)
25             new_node.enfant_droit = self.enfant_droit
26             self.enfant_droit = new_node
27
28     def get_valeur(self):
29         '''Renvoie la valeur de la racine'''
30         return self.valeur
31
32     def get_gauche(self):
33         '''Renvoie le sous arbre gauche'''
34         return self.enfant_gauche
35
36     def get_droit(self):
37         '''Renvoie le sous arbre droit'''
38         return self.enfant_droit
```

   a. En utilisant la classe définie ci-dessus, donner un exemple d'attribut, puis un exemple de méthode.
   b. Après avoir défini la classe `ArbreBinaire`, on exécute les instructions Python suivantes :

```
r = ArbreBinaire(15)
r.insert_gauche(6)
r.insert_droit(18)
a = r.get_valeur()
b = r.get_gauche()
c = b.get_valeur()
```

Donner les valeurs associées aux variables a et c après l'exécution de ce code.

On utilise maintenant la classe ArbreBinaire pour implémenter un arbre binaire de recherche.

On utilisera la définition suivante : un arbre binaire de recherche est un arbre binaire, dans lequel :

- on peut comparer les valeurs des nœuds : ce sont par exemple des nombres entiers, ou des lettres de l'alphabet.
- si `x` est un nœud de cet arbre et `y` est un nœud du sous-arbre gauche de `x`, alors il faut que `y.valeur <= x.valeur`.
- si `x` est un nœud de cet arbre et `y` est un nœud du sous-arbre droit de `x`, alors il faut que `y.valeur => x.valeur`.

2. On exécute le code Python suivant. Représenter graphiquement l'arbre ainsi obtenu.

```
racine_r = ArbreBinaire(15)
racine_r.insert_gauche(6)
racine_r.insert_droit(18)
r_6 = racine_r.get_gauche()
r_6.insert_gauche(3)
r_6.insert_droit(7)
r_18 = racine_r.get_droit()
r_18.insert_gauche(17)
r_18.insert_droit(20)
r_3 = r_6.get_gauche()
r_3.insert_gauche(2)
```

3. On a représenté sur la figure 1 ci-dessous un arbre. Justifier qu'il ne s'agit pas d'un arbre binaire de recherche. Redessiner cet arbre sur votre copie en conservant l'ensemble des valeurs `{2,3,5,10,11,12,13}` pour les nœuds afin qu'il devienne un arbre binaire de recherche.

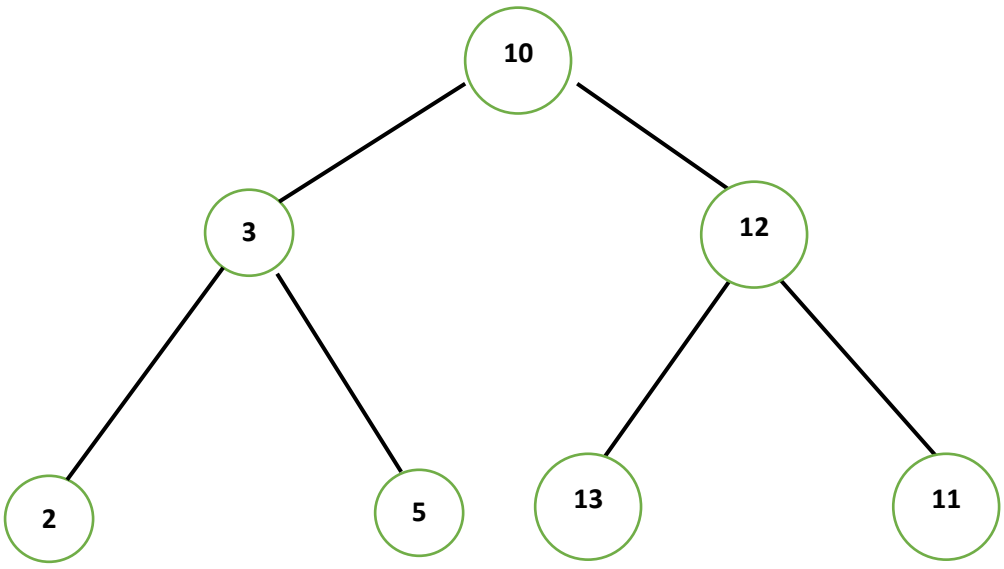


Figure 1

4. On considère qu'on a implémenté un objet `ArbreBinaire` nommé `A` représenté sur la figure 2.

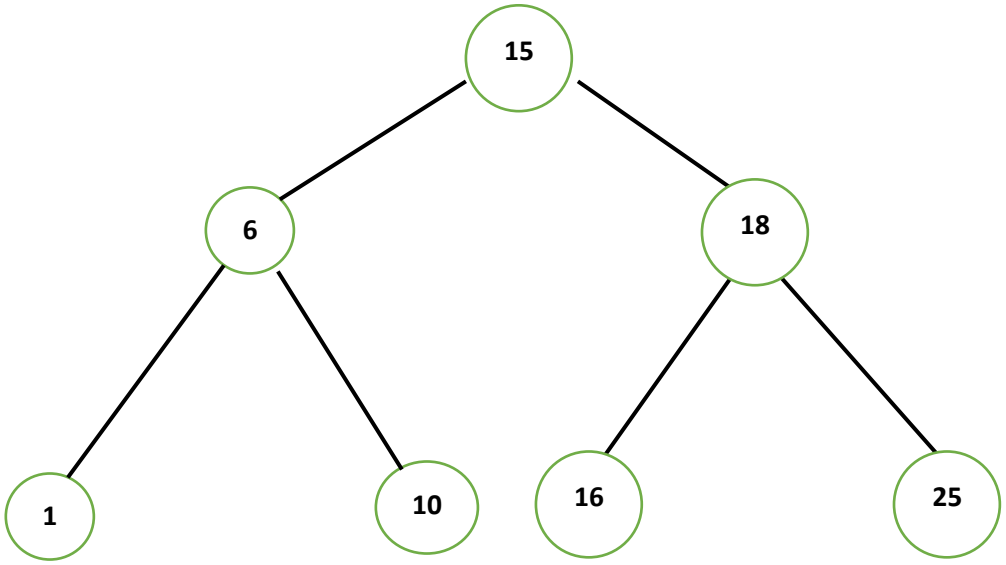


Figure 2

On définit la fonction `parcours_infixe` suivante, qui prend en paramètre un objet `ArbreBinaire` `T` et un second paramètre `parcours` de type liste.

```
1  def parcours_infixe(T, parcours):
2      ''' Affiche la liste des valeurs de l'arbre'''
3      if T is not None:
4          parcours_infixe(T.get_gauche(), parcours)
5          parcours_append(T.get_valeur())
6          parcours_infixe(T.get_droit(), parcours)
7      return parcours
```

Donner la liste renvoyée par l'appel suivant : `parcours_infixe(A,[])`.

Exercice 3

Dans un entrepôt de e-commerce, un robot mobile autonome exécute successivement les tâches qu'il reçoit tout au long de la journée.

La mémorisation et la gestion de ces tâches sont assurées par une structure de données.

1. Dans l'hypothèse où les tâches devraient être extraites de cette structure (pour être exécutées) dans le même ordre qu'elles ont été mémorisées, préciser si ce fonctionnement traduit le comportement d'une file ou d'une pile. Justifier.

En réalité, selon l'urgence des tâches à effectuer, on associe à chacune d'elles, lors de la mémorisation, un indice de priorité (nombre entier) distinct : il n'y a pas de valeur en double. Plus cet indice est faible, plus la tâche doit être traitée prioritairement. La structure de données retenue est assimilée à un arbre binaire de recherche (ABR) dans lequel chaque nœud correspond à une tâche caractérisée par son indice de priorité.

> Rappel : Dans un arbre binaire de recherche, chaque nœud est caractérisé par une valeur (ici l'indice de priorité), telle que chaque nœud du sous-arbre gauche a une valeur strictement inférieure à celle du nœud considéré, et que chaque nœud du sous-arbre droit possède une valeur strictement supérieure à celle-ci. Cette structure de données présente l'avantage de mettre efficacement en œuvre l'insertion ou la suppression de nœuds, ainsi que la recherche d'une valeur.

Par exemple, le robot a reçu successivement, dans l'ordre, des tâches d'indice de priorité 12, 6, 10, 14, 8 et 13. En partant d'un arbre binaire de recherche vide, l'insertion des différentes priorités dans cet arbre donne la figure 1.

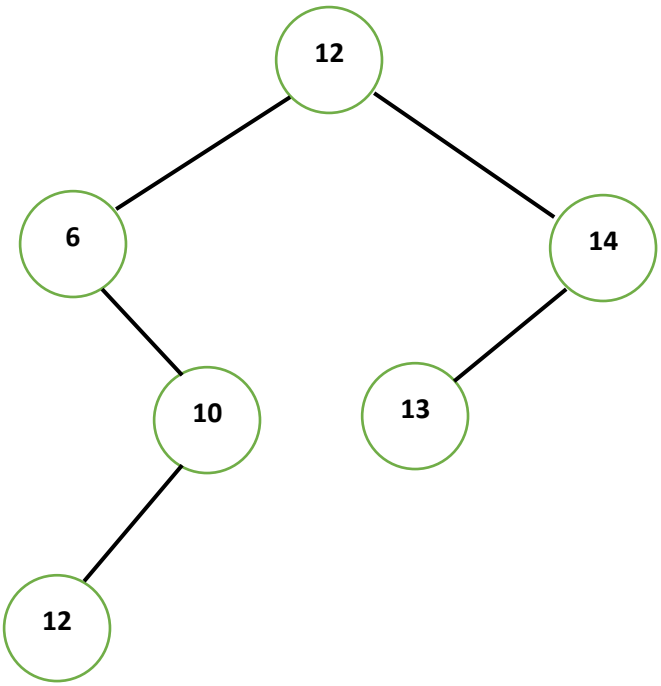


Figure 1 : Exemple d'un arbre binaire

2. En utilisant le vocabulaire couramment utilisé pour les arbres, préciser le terme qui correspond
   a. au nombre de tâches restant à effectuer, c'est-à-dire le nombre total de nœuds de l'arbre ;
   b. au nœud représentant la tâche restant à effectuer la plus ancienne ;
   c. au nœud représentant la dernière tâche mémorisée (la plus récente).

3. Lorsque le robot reçoit une nouvelle tâche, on déclare un nouvel objet, instance de la classe `Noeud`, puis on l'insère dans l'arbre binaire de recherche (instance de la classe `ABR`) du robot. Ces 2 classes sont définies comme suit :

```
1  class Noeud:
2      def __init__(self, tache, indice):
3          self.tache = tache #ce que doit accomplir le robot
4          self.indice = indice #indice de priorité (int)
5          self.gauche = ABR() #sous-arbre gauche vide (ABR)
6          self.droite = ABR() #sous-arbre droit vide (ABR)
7
8  class ABR:
9      #arbre binaire de recherche initialement vide
10     def __init__(self):
11         self.racine = None #arbre vide
12         #Remarque : si l'arbre n'est pas vide, racine est
13         #une instance de la classe Noeud
14
15     def est_vide(self):
16         """renvoie True si l'arbre autoréférencé est vide,
17         False sinon"""
18         return self.racine == None
19
20     def insere(self, nouveau_noeud):
21         """insere un nouveau noeud, instance de la classe
22         Noeud, dans l'ABR"""
23         if self.est_vide():
24             self.racine = nouveau_noeud
25         elif self.racine.indice …… nouveau_noeud.indice
26             self.racine.gauche.insere(nouveau_noeud)
27         else:
28             self.racine.droite.insere(nouveau_noeud)
```

   a. Donner les noms des attributs de la classe `Noeud`.
   b. Expliquer en quoi la méthode `insere` est dite récursive et justifier rapidement qu'elle se termine.
   c. Indiquer le symbole de comparaison manquant dans le test à la ligne 25 de la méthode `insere` pour que l'arbre binaire de recherche réponde bien à la définition de l'encadré « Rappel ».
   d. On considère le robot dont la liste des tâches est représentée par l'arbre de la figure 1. Ce robot reçoit, successivement et dans l'ordre, des tâches d'indice de priorité 11, 5, 16 et 7, sans avoir accompli la moindre tâche entretemps. Recopier et compléter la figure 1 après l'insertion de ces nouvelles tâches.

4. Avant d'insérer une nouvelle tâche dans l'arbre binaire de recherche, il faut s'assurer que son indice de priorité n'est pas déjà présent.

   Écrire une méthode `est_present` de la classe `ABR` qui répond à la description :

```
29
30      def est_present(self, indice_recherche) :
31          """renvoie True si l'indice de priorité indice_recherche
32          int) passé en paramètre est déjà l'indice d'un nœud
33          de l'arbre, False sinon"""
```

5. Comme le robot doit toujours traiter la tâche dont l'indice de priorité est le plus petit, on envisage un parcours infixe de l'arbre binaire de recherche.
   a. Donner l'ordre des indices de priorité obtenus à l'aide d'un parcours infixe de l'arbre binaire de recherche de la figure 1.
   b. Expliquer comment exploiter ce parcours pour déterminer la tâche prioritaire.

6. Afin de ne pas parcourir tout l'arbre, il est plus efficace de rechercher la tâche du nœud situé le plus à gauche de l'arbre binaire de recherche : il correspond à la tâche prioritaire.
   Recopier et compléter la méthode récursive `tache_prioritaire` de la classe `ABR`:

```
34
35      def tache_prioritaire(self):
36          """renvoie la tâche du noeud situé le plus
37          à gauche de l'ABR supposé non vide"""
38          if self.racine.…….est_vide():#pas de nœud plus à gauche
39              return self.racine.…….
40          else:
41              return self.racine.gauche.……()
```

7. Une fois la tâche prioritaire effectuée, il est nécessaire de supprimer le nœud correspondant pour que le robot passe à la tâche suivante :
   - Si le nœud correspondant à la tâche prioritaire est une feuille, alors il est simplement supprimé de l'arbre (cette feuille devient un arbre vide)
   - Si le nœud correspondant à la tâche prioritaire a un sous-arbre droit non vide, alors ce sous-arbre droit remplace le nœud prioritaire qui est alors écrasé, même s'il s'agit de la racine.

   Dessiner alors, pour chaque étape, l'arbre binaire de recherche (seuls les indices de priorités seront représentés) obtenu pour un robot, initialement sans tâche, et qui a, successivement dans l'ordre :
   - étape 1 : reçu une tâche d'indice de priorité 14 à accomplir
   - étape 2 : reçu une tâche d'indice de priorité 11 à accomplir
   - étape 3 : reçu une tâche d'indice de priorité 8 à accomplir
   - étape 4 : accompli sa tâche prioritaire
   - étape 5 : reçu une tâche d'indice de priorité 12 à accomplir
   - étape 6 : accompli sa tâche prioritaire
   - étape 7 : accompli sa tâche prioritaire
   - étape 8 : reçu une tâche d'indice de priorité 15 à accomplir
   - étape 9 : reçu une tâche d'indice de priorité 19 à accomplir
   - étape 10 : accompli sa tâche prioritaire

Programme de révision des vacances

Jour	Pratique(sujets 2023)	Thème écrit
1er week-end repos		
13/02/2023	1 - 2 - 3	Exercice 2 : Python, classes, itération, récursivité Lien 1
14/02/2023	4 - 5 - 6	Exercice 3 : bases de données Lien 1
15/02/2023	7 - 8 - 9	Exercice 2 : programmation, algorithmes de tri Lien 2
16/02/2023	10 - 11 - 12	Exercice 5 : réseau, protocoles de routage Lien 2
17/02/2023	13 - 14 - 15	Exercice 1 : programmation, algorithme, complexité Lien 3
2ème week-end repos		
20/02/2023	16 - 17 - 18	Exercice 4 : POO, Python Lien 4
21/02/2023	19 - 20 - 21	Exercice 3 : structures de donnée, programmation (jeu de la vie) Lien 5
22/02/2023	22 - 23 - 24	Exercice 1 : bases de données (Vacances autrement) Lien 6
23/02/2023	25 - 26 - 27	Exercice 5 : algorithmes, Python (TAKAZU) Lien 7
24/02/2023	28 - 29 - 30	Exercice 3 : binaire, routage Lien 8
3ème week-end repos		
QR Code		

- Lien 1 : https://kxs.fr/files/sujets/2022/ecrit/terminale-2022-rattrapage-jour-1.pdf
- Lien 2 : https://kxs.fr/files/sujets/2022/ecrit/terminale-2022-amerique-sud-jour-1.pdf
- Lien 3 : https://kxs.fr/files/sujets/2022/ecrit/terminale-2022-amerique-sud-jour-2.pdf
- Lien 4 : https://kxs.fr/files/sujets/2022/ecrit/terminale-2022-asie-jour-1.pdf
- Lien 5 : https://kxs.fr/files/sujets/2022/ecrit/terminale-2022-asie-jour-2.pdf
- Lien 6 : https://kxs.fr/files/sujets/2022/ecrit/terminale-2022-amerique-nord-jour-1.pdf
- Lien 7 : https://kxs.fr/files/sujets/2022/ecrit/terminale-2022-mayotte-jour-2.pdf
- Lien 8 : https://kxs.fr/files/sujets/2022/ecrit/terminale-2022-metropole-sujet-1.pdf