In [4]:
```python
# version finale du script encodage automatique

# import des librairies
import json
import os
import re
from lxml import etree
from lxml import etree as ET
import dateparser
import datetime


def generate_id(tag):
    """Génère un identifiant unique pour une balise en fonction de son type."""
    if tag not in ids:
        ids[tag] = 1
    else:
        ids[tag] += 1
    return f"{tag}{ids[tag]}"

def write_comment(filename):
    """fonction qui génère un commentaire lors du changement de page"""
    filename = os.path.basename(filename)
    comment_text = f"New page added from {filename}"
    comment_element = ET.Comment(comment_text)
    return comment_element


def create_teiheader():
    """ cette fonction crée le teiHeader """

    # fileDesc______________________________________________________________________

    fileDesc = ET.SubElement(teiHeader , "fileDesc")

    # Titre______________________________________________________________________

    global titleStmt
    titleStmt = ET.SubElement(fileDesc , "titleStmt")
    title_fr = ET.SubElement(titleStmt, "title", type ="main")
    title_fr.attrib["{http://www.w3.org/XML/1998/namespace}lang"] = "fr"
    title_fr.text = "Journal officiel de la République française. Débats parlementaires"
    title_en = ET.SubElement(titleStmt, "title", type ="main")
    title_en.attrib["{http://www.w3.org/XML/1998/namespace}lang"] = "en"
    title_en.text = "Official Journal of the French Republic. Parliamentary debates"

    title_sub_fr = ET.SubElement(titleStmt, "title", type ="sub")
```

```python
title_sub_fr.attrib["{http://www.w3.org/XML/1998/namespace}lang"] = "fr"
title_sub_fr.text = "Chambre des députés : compte rendu in-extenso"
title_sub_en = ET.SubElement(titleStmt, "title", type ="sub")
title_sub_en.attrib["{http://www.w3.org/XML/1998/namespace}lang"] = "en"
title_sub_en.text = "Chamber of Deputies: verbatim report"

# respStmt________________________________________________________

personnes = {"Brunel TCHEKELI": "id-hal", "Marie PUREN": "id-hal", "Pierre VERNUS": "id-hal", "Fanny LEBRETON" : "id-hal"}

# Boucle pour créer respStmt pour chaque personne
for personne, identifiant in personnes.items():
    respStmt = ET.SubElement(titleStmt , "respStmt")
    persName = ET.SubElement(respStmt , "persName")
    forename = ET.SubElement(persName , "forename")
    surname = ET.SubElement(persName , "surname")

    # Définir l'attribut xml:id pour l'élément ptr
    #ptr = ET.SubElement(persName, "ptr")
    #ptr.attrib["type"] = identifiant

     # Définir le texte pour forename et surname
    forename.text = personne.split()[0]  # Utiliser le prénom de la personne
    surname.text = personne.split()[1]  # Utiliser le nom de famille de la personne

    # Vérifier si la personne est "Brunel TCHEKELI"
    if personne == "Brunel TCHEKELI":
        # Ajouter l'élément ptr avec les attributs appropriés
        resp_fr = ET.SubElement(respStmt, "resp", {"{http://www.w3.org/XML/1998/namespace}lang": "fr"})
        resp_fr.text = "Amélioration du script de transformation du JSON en XML-TEI et ajout automatique des balises TEI par des script
        resp_en = ET.SubElement(respStmt, "resp", {"{http://www.w3.org/XML/1998/namespace}lang": "en"})
        resp_en.text = "Improved scripting for transforming JSON into XML-TEI and automatic addition of TEI tags by Python scripts"

    # Vérifier si la personne est " Fanny Lebreton"
    if personne == "Fanny LEBRETON":
        # Ajouter l'élément ptr avec les attributs appropriés
        resp_fr = ET.SubElement(respStmt, "resp", {"{http://www.w3.org/XML/1998/namespace}lang": "fr"})
        resp_fr.text = "Ajout automatique des balises TEI par des scripts Python"
        resp_en = ET.SubElement(respStmt, "resp", {"{http://www.w3.org/XML/1998/namespace}lang": "en"})
        resp_en.text = "Transformation from JSON to XML-TEI and automatic addition of TEI tags by Python scripts"


    # Vérifier si la personne est "Marie PUREN"
    if personne == "Marie PUREN":
        # Ajouter l'élément ptr avec les attributs appropriés
        ptr = ET.SubElement(persName, "ptr", type= identifiant, target=personne)
        ptr = ET.SubElement(persName, "ptr", type="orcid", target="0000-0001-5452-3913")
        resp_fr = ET.SubElement(respStmt, "resp", {"{http://www.w3.org/XML/1998/namespace}lang": "fr"})
        resp_fr.text = "agoda_schema.rng"
```

```python
            resp_en = ET.SubElement(respStmt, "resp", {"{http://www.w3.org/XML/1998/namespace}lang": "en"})
            resp_en.text = "agoda_schema.rng"

        # Vérifier si la personne est "Pierre VERNUS"
        if personne == "Pierre VERNUS" :
            # Ajouter l'élément ptr avec les attributs standard
            ptr = ET.SubElement(persName, "ptr", type=identifiant, target=personne)
            ptr = ET.SubElement(persName, "ptr", type="orcid", target="0000-0001-5452-3913")
            resp_fr = ET.SubElement(respStmt, "resp", {"{http://www.w3.org/XML/1998/namespace}lang": "fr"})
            resp_fr.text = "TEI Header"
            resp_en = ET.SubElement(respStmt, "resp", {"{http://www.w3.org/XML/1998/namespace}lang": "en"})
            resp_en.text = "TEI Header"


    # Création de l'élément funder
    funder = ET.SubElement(titleStmt, "funder")
    # Création des éléments orgName avec les attributs xml:lang correspondants
    orgName_fr = ET.SubElement(funder, "orgName", {"{http://www.w3.org/XML/1998/namespace}lang": "fr"})
    orgName_fr.text = "Bibliothèque nationale de France"
    orgName_en = ET.SubElement(funder, "orgName", {"{http://www.w3.org/XML/1998/namespace}lang": "en"})
    orgName_en.text = "National Library of France"

    # Ajout de extent avec les informations sur le nombre de page, le nombre de mots etc (plus tard)

    extent= ET.SubElement(fileDesc, "extent")
    measure_pages_fr = ET.SubElement(extent, "measure",  {"quantity": "1", "{http://www.w3.org/XML/1998/namespace}lang": "fr"})
    measure_pages_fr.text = "pages"
    measure_pages_en = ET.SubElement(extent, "measure",  {"quantity": "1", "{http://www.w3.org/XML/1998/namespace}lang": "en"})
    measure_pages_en.text = "pages"

    measure_utterances_en = ET.SubElement(extent, "measure",  {"quantity": "1", "{http://www.w3.org/XML/1998/namespace}lang": "en"})
    measure_utterances_en.text = "utterances"

    measure_words_en = ET.SubElement(extent, "measure", {"quantity": "1", "{http://www.w3.org/XML/1998/namespace}lang": "en"})
    measure_words_en.text = "words"

    global publicationStmt
    publicationStmt = ET.SubElement(fileDesc , "publicationStmt")
    publisher = ET.SubElement(publicationStmt, "publisher")
    publisher.text = "AGODA"
    authority = ET.SubElement(publicationStmt, "authority")
    authority.text = "Bnf Datalab"
    availability = ET.SubElement(publicationStmt, "availability", status="restricted", n="cc-by")
    licence = ET.SubElement(availability, "licence", target="https://creativecommons.org/licenses/by/4.0/")

    # ajout de la date _________________________________

    now = datetime.datetime.now()
    date = ET.SubElement(publicationStmt, "date", {"when": now.strftime("%Y-%m-%d")})
    # date générée automatiquement en utilisant la méthode now() de la classe datetime.datetime et
    # le format est défini avec strftime() en utilisant le modèle "AAAA-MM-JJ"

     # ajout de la dsourceDesc _________________________________
    sourceDesc = ET.SubElement(fileDesc , "sourceDesc")
    biblFull = ET.SubElement(sourceDesc, "biblFull")
    titleStmt_sDc = ET.SubElement(biblFull, "titleStmt")
    title_sDc = ET.SubElement(titleStmt_sDc, "title")
    title_sDc.text = '''Journal officiel de la République française. Débats parlementaires.
                           Chambre des députés : compte rendu in-extenso'''

    global publicationStmt_sDc
    publicationStmt_sDc = ET.SubElement(biblFull , "publicationStmt")
    publisher_sDc_fr = ET.SubElement(publicationStmt_sDc, "publisher", {"{http://www.w3.org/XML/1998/namespace}lang": "fr"})
    publisher_sDc_fr.text = " "

    publisher_sDc_en = ET.SubElement(publicationStmt_sDc, "publisher", {"{http://www.w3.org/XML/1998/namespace}lang": "en"})
    publisher_sDc_en.text = " "

    pubPlace_sDc = ET.SubElement(publicationStmt_sDc, "pubPlace")
    location_sDc = ET.SubElement(pubPlace_sDc, "location")
    country_sDc = ET.SubElement(location_sDc, "country", key="FR")
    settlement_sDc = ET.SubElement(location_sDc, "settlement", type="city")
    settlement_sDc.text = "Paris"
    # date____________(voir partie date-pub plus bas: la date est récupérée dans chaque fichier et est ajouté ici)

    distributor_sDc = ET.SubElement(publicationStmt_sDc, "distributor", facs="https://gallica.bnf.fr/ark:/12148/bpt6k477552f/f1")
    distributor_sDc.text = "Source gallica.bnf.fr / Bibliothèque nationale de France"
    availability_sDc = ET.SubElement(publicationStmt_sDc, "availability")
    licence_sDc = ET.SubElement(availability_sDc, "licence", {"target":"https://gallica.bnf.fr/edit/und/conditions-dutilisation-des-contenus
    licence_sDc_p1 = ET.SubElement(licence_sDc, "p" )
    licence_sDc_p1.text = "Les contenus accessibles sur le site Gallica sont pour la plupart des reproductions numériques d'œuvres tombées d
    licence_sDc_p2 = ET.SubElement(licence_sDc, "p" )
    licence_sDc_p2.text = "Ces contenus sont considérés, en vertu du code des relations entre le public et l’administration, comme étant des

    seriesStmt = ET.SubElement(biblFull, "seriesStmt")
    title_series = ET.SubElement(seriesStmt, "title" )
    title_series.text = "Journal Officiel de la République française"
    biblScope1 = ET.SubElement(seriesStmt, "biblScope")
    biblScope1.text = "Débats parlementaires"
    biblScope2 = ET.SubElement(seriesStmt, "biblScope")
    biblScope2.text = "Chambre des députés"
    idno = ET.SubElement(seriesStmt, "idno" , type="ISSN")
    idno.text = "1270-5942"



# EncodingDesc ______________________________________________________________________
```

```
    encodingDesc = ET.SubElement(teiHeader , "encodingDesc")

# profileDesc _______________________________________________________________________
    profileDesc = ET.SubElement(teiHeader , "profileDesc")
    langUsage = ET.SubElement(profileDesc, "langUsage")
    language = ET.SubElement(langUsage, "language", ident="fr")
    language.text = "Français"
    global setting_desc
    settingDesc = ET.SubElement(profileDesc, "settingDesc")
    setting_desc = ET.SubElement(settingDesc, "setting")
    name_desc1 = ET.SubElement(setting_desc, "name", type="place")
    name_desc1.text = "Palais Bourbon"
    name_desc2 = ET.SubElement(setting_desc, "name", type="city")
    name_desc2  = "Paris"
    name_desc3 = ET.SubElement(setting_desc, "name", type="country", key ="FR")
    name_desc3 = "France"
   # date_desc = voir partie date plus bas


# Ajout de la description du projet---------------------------------------------------------
    projectDesc = ET.SubElement(encodingDesc, "projectDesc")
    p1 = ET.SubElement(projectDesc, "p")
    p1.attrib["{http://www.w3.org/XML/1998/namespace}lang"] = "fr" # A cause des crochets et trop de "", écrire l'attibut de cette manière p
    ref1 = ET.SubElement(p1, "ref", target = "https://www.bnf.fr/fr/les-projets-de-recherche#bnf-agoda")
    ref1.text = "AGODA "
    ref1.tail = '''est un projet qui a pour objectif de rendre disponible au format XML-TEI les textes de débats parlementaires à la Chambre
    ref2 = ET.SubElement(p1, "ref", target ="https://github.com/mpuren/agoda/blob/ODD/documentation/agoda_odd.xml")
    ref2.text = "ODD "
    ref2.tail = "défini pour le projet à partir des "
    ref3 = ET.SubElement(p1, "ref", target = "https://github.com/clarin-eric/parla-clarin")
    ref3.text =  "recommandations produites par Parla-CLARIN. "
    ref3.tail = ''' Dans une optique de preuve de concept, la phase 1 du projet AGODA se concentre plus particulièrement
    sur la 5ème législature (1889-1893). Les textes encodés sont d'abord extraits des documents numérisés disponibles sur '''
    ref4 = ET.SubElement(p1, "ref", target = "https://gallica.bnf.fr/ark:/12148/cb328020951/date.item")
    ref4.text = "Gallica,"
    ref4.tail = ''' la bibliothèque numérique de la Biliothèque nationale de France, puis ils sont convertis
    en XML-TEI au moyen de scripts Python.'''

    # Version anglaise _______________________________________________________________

    p2 = ET.SubElement(projectDesc, "p")
    p2.attrib["{http://www.w3.org/XML/1998/namespace}lang"] = "en"
    ref1_2 = ET.SubElement(p2, "ref", target = "https://www.bnf.fr/fr/les-projets-de-recherche#bnf-agoda")
    ref1_2.text = "AGODA "
    ref1_2.tail = '''is a project that aims to make available in XML-TEI format the texts of parliamentary debates in the
    Chamber of Deputies during the Third Republic, following the'''
    ref2_2 = ET.SubElement(p2, "ref", target ="https://github.com/mpuren/agoda/blob/ODD/documentation/agoda_odd.xml")
    ref2_2.text = "ODD "
    ref2_2.tail = " defined for the project from the "
    ref2_2.tail = ''' defined for the project from the
    ref3_2 = ET.SubElement(p2, "ref", target = "https://github.com/clarin-eric/parla-clarin")
    ref3_2.text = " Parla-CLARIN recommendations "
    ref3_2.tail = ''' From a proof-of-concept perspective, phase 1 of the AGODA project focuses more specifically on the
    5th legislature (1889-1893). The encoded texts are first extracted from the digitised documents available on '''
    ref4_2 = ET.SubElement(p2, "ref", target = "https://gallica.bnf.fr/ark:/12148/cb328020951/date.item")
    ref4_2.text = "Gallica,"
    ref4_2.tail = '''  the digital library of the Biliothèque nationale de France, then they are converted into
    XML-TEI using Python scripts.'''

 # fin de la description du projet---------------------------------------------------------

# _____ fichier compilation__________________________________________________

# Chemin absolu du dossier contenant les fichiers à parcourir
dossier_json = os.path.join(os.getcwd(), "json_data")
dossier_xml = os.path.join(os.getcwd(), "xml_data")

# Créer un pattern regex pour extraire le numéro de page du nom de fichier
page_number_pattern = re.compile(r'^.*_p(\d+)\.json$')

# Récupérer la liste des fichiers JSON dans le dossier
fichiers = [f for f in os.listdir(dossier_json) if f.endswith('.json')]

# Trier les fichiers JSON en fonction du numéro de page
fichiers_json_tries = sorted(fichiers, key=lambda x: int(page_number_pattern.match(x).group(1)))



#global div_sitting, div, body, ids
ids = {"Partie_": 0, "n": 0, "s": 0, "note": 0}
# Création de l'élément racine et le squelette du XML
root = ET.Element("TEI", xmlns="http://www.tei-c.org/ns/1.0")
root.attrib["{http://www.w3.org/XML/1998/namespace}lang"] = "fr"

teiHeader = ET.SubElement(root, "teiHeader")
text_tei = ET.SubElement(root, "text")
body = ET.SubElement(text_tei, "body")
back = ET.SubElement(text_tei, "back")
div_sitting = ET.SubElement(body, "div", attrib={"type": "sitting"})
u_element = ET.Element("u")

# initialisation nécessaire pour quelques variables : Cette initialisation permet d'éviter
# des erreurs de type : "Variable is not defined"
divs_cibles = []
note_beg = None
added_segs = set()

current_parent = div_sitting
quote = None
```

```python
quote = None
note_voterlist = None
comment_note = None
comment_beg_note = None
seg_beg = None
u_beg_seg_beg = None
quote_seg_beg = None
quote_beg_seg_beg = None
signed = None
note_beg_seg = None

# Parcourir les fichiers JSON triés
for index, fichier_json in enumerate(fichiers_json_tries) :

    fichier_json = os.path.join(dossier_json, fichier_json)
    filename = os.path.basename(fichier_json)
    page_number = filename.split("_p0")[1].split(".")[0]
    xml_id = os.path.splitext(filename)[0]
    root.attrib["{http://www.w3.org/XML/1998/namespace}id"] = xml_id

    with open(fichier_json, "r", encoding="utf-8") as f:
        # Lire le contenu du fichier JSON
        data = json.load(f)

        bp_element = ET.Element("pb", attrib={"n": "{}".format(page_number)})
        # bp_element permet de réccupérer le numéro de page et de l'afficher sous forme <pb n="xxx"/>
        bp_element.addprevious(write_comment(filename)) # ajoute un commentaire avant le <pb n="xxx"/>

        if index == 0 : # si premier fichier , alors applique la fonction de création du teiHeader
            create_teiheader()

        for i in range(len(data)):

            if "comment" in data[i]:

                # ______________________Grandes divisions ___________________________________________________________

                if data[i]['comment'] == 'part head' or data[i]['comment'] == "head part" :

                    # Ajouter un élément 'part head' avec le contenu de la clé 'text_ocr'
                    div_part = ET.SubElement(div_sitting, "div", attrib={"type": "part", "corresp": "#pv"})
                    part_head = ET.SubElement(div_part, "head" )
                    part_head.text = data[i]['text_ocr']
                    div_part.addprevious(etree.Comment(generate_id("Partie_"))) # ajout de commentaire avant chaque partie suivi d'un id
                    divs_cibles.append(div_part)

                elif re.search(r"part1(?!-)", data[i]["comment"]):

                    div_part1 = ET.SubElement(div_sitting, "div", attrib={"type": "part"})
                    div_part1.addprevious(etree.Comment(generate_id("div_part")))
                    div_part1.addprevious(etree.Comment(generate_id("div_part")))
                    divs_cibles.append(div_part1)

                elif re.search(r"agenda", data[i]["comment"]):
                        # Ajouter un élément 'agenda' avec le contenu de la clé 'text_ocr'
                    div_agenda = ET.SubElement(div_sitting,"div", type="agenda")
                    div_agenda.addprevious(etree.Comment("Partie_Agenda"))
                    divs_cibles.append(div_agenda)
                    agenda_head = ET.SubElement(div_agenda, "head")
                    agenda_head.text = data[i]['text_ocr']

                elif re.search(r"\b(?<!-)appendices\b", data[i]["comment"]):
                    div_appendices = ET.SubElement(div_sitting, "div", attrib={"type": "appendices"})
                    div_appendices.addprevious(etree.Comment(generate_id("Partie_appendices_")))
                    divs_cibles.append(div_appendices)
                    head_appendices = ET.SubElement(div_appendices, "head")
                    head_appendices.text = data[i]['text_ocr']

                elif re.search(r"part1-appendices", data[i]["comment"]):
                    div_appendices1 = ET.SubElement(div_sitting, "div", attrib={"type": "appendices"})
                    div_appendices1.addprevious(etree.Comment(generate_id("Partie_appendices-1_")))
                    divs_cibles.append(div_appendices1)
                    head_appendices1 = ET.SubElement(div_appendices1, "head")
                    head_appendices1.text = data[i]['text_ocr']

                elif re.search(r"\b(?<!-)erratum\b", data[i]["comment"]) :
                    # Ajouter un élément 'erratum' avec le contenu de la clé 'text_ocr'
                    div_erratum = ET.SubElement(back, "div", attrib={"type": "erratum"})
                    div_erratum.addprevious(etree.Comment(generate_id("Partie_erratum_")))
                    divs_cibles.append(div_erratum)
                    head_annexe = ET.SubElement(div_erratum, "head")
                    label_annexe = ET.SubElement(head_annexe, "label")
                    label_annexe.text = data[i]['text_ocr']

                elif re.search(r"part1-erratum", data[i]["comment"]):
                    # Ajouter un élément 'part1-erratum' avec le contenu de la clé 'text_ocr'
                    div_erratum1 = ET.SubElement(back, "div", attrib={"type": "erratum"})
                    div_erratum1.addprevious(etree.Comment(generate_id("Partie-1_erratum_")))
                    divs_cibles.append(div_erratum1)
                    head_annexe1 = ET.SubElement(div_erratum, "head")
                    label_annexe1 = ET.SubElement(head_annexe, "label")
                    label_annexe1.text = data[i]['text_ocr']

                elif re.search(r"\b(?<!-)lists\b", data[i]["comment"]):
                    # Ajouter un élément 'lists' avec le contenu de la clé 'text_ocr'
                    div_lists = ET.SubElement(back, "div", attrib={"type": "lists"})
                    div_list.addprevious(etree.Comment(generate_id("Partie_lists_")))
                    head_lists = ET.SubElement(div_lists, "head")
                    label_lists = ET.SubElement(head_lists, "label")
                    label_lists.text = data[i]['text_ocr']
                    label_lists.text = data[i]['text_ocr']
                    divs_cibles.append(div_lists)

                elif re.search(r"part1-lists", data[i]["comment"]):
                     # Ajouter un élément 'part1-lists' avec le contenu de la clé 'text_ocr'
                    div_lists1 = ET.SubElement(back, "div", attrib={"type": "lists"})
                    div_list1.addprevious(etree.Comment(generate_id("Partie-1_lists_")))
                    divs_cibles.append(div_lists1)
                    head_lists1 = ET.SubElement(div_lists1, "head")
                    label_lists1 = ET.SubElement(head_lists1, "label")
                    label_lists1.text = data[i]['text_ocr']

                elif re.search(r"\b(?<!-)offices\b", data[i]["comment"]):
                     # Ajouter un élément 'offices' avec le contenu de la clé 'text_ocr'
                    div_offices = ET.SubElement(div_sitting, "div", attrib={"type": "offices"})
                    div_offices.addprevious(etree.Comment(generate_id("Partie_offices_")))
                    divs_cibles.append(div_offices)
                    head_offices = ET.SubElement(div_offices, "head")
                    head_offices.text = data[i]['text_ocr']

                elif re.search(r"part1-offices", data[i]["comment"]):
                    # Ajouter un élément 'part1-offices' avec le contenu de la clé 'text_ocr'
                    div_offices1 = ET.SubElement(div_sitting, "div", attrib={"type": "offices"})
                    div_offices1.addprevious(etree.Comment(generate_id("Partie-1_offices_")))
                    divs_cibles.append(div_offices1)
                    head_offices1 = ET.SubElement(div_offices1, "head")
                    head_offices1.text = data[i]['text_ocr']

                elif re.search(r"\b(?<!-)sitting\b", data[i]["comment"]) and re.search(r"contents", data[i]["comment"]):
                    div_content = ET.SubElement(div_sitting, "div", attrib={"type": "contents"})
                    div_content.addprevious(etree.Comment("SOMMAIRE"))
                    list_item = ET.SubElement(div_content, "list")

                elif re.search(r"other-sitting", data[i]["comment"]):
                    # Ajouter un élément 'other-sitting' avec le contenu de la clé 'text_ocr'
                    div_other_sitt = ET.SubElement(body, "div", attrib={"type": "other-sitting"})
                    div_other_sitt.addprevious(etree.Comment(generate_id("other-sitting_")))
                    divs_cibles.append(div_other_sitt)
                    head = ET.SubElement(div_other_sitt, "head")
                    head.text = data[i]['text_ocr']

                elif data[i]["comment"] == "voting":
                    div_voting = ET.SubElement(div_sitting, "div", attrib={"type": "voting"})
                    div_voting.addprevious(etree.Comment("Voting"))

                    head_voting = ET.SubElement(div_voting, "head")
                    label_voting = ET.SubElement(head_voting, "label")
                    label_voting.text = data[i]['text_ocr']
                    divs_cibles.append(div_voting)
```

```python
        elif re.search(r"voting1", data[i]["comment"]):
            # Ajouter un élément 'voting1' avec le contenu de la clé 'text_ocr'
            div_voting1 = ET.SubElement(div_sitting, "div", attrib={"type": "voting1"})
            div_voting1.addprevious(etree.Comment("Partie-1_voting"))
            head_voting1 = ET.SubElement(div_voting1, "head")
            label_voting1 = ET.SubElement(head_voting1, "label")
            label_voting1.text = data[i]['text_ocr']
            divs_cibles.append(div_voting1)

        elif re.search(r"rectification", data[i]["comment"]):
            # Ajouter un élément 'rectification' avec le contenu de la clé 'text_ocr'
            div_rectification = ET.SubElement(div_sitting, "div", attrib={"type": "rectification"})
            div_rectification.addprevious(etree.Comment(generate_id("Rectification_")))
            head_rectification = ET.SubElement(div_rectification, "head")
            head_rectification.text = data[i]['text_ocr']
            divs_cibles.append(div_rectification)

        elif re.search(r"\b(?<!-)petition\b", data[i]["comment"]):
             # Ajouter un élément 'petition' avec le contenu de la clé 'text_ocr'
            div_petition = ET.SubElement(back, "div", attrib={"type": "petition"})
            div_petition.addprevious(etree.Comment(generate_id("Petition_")))
            head_petition = ET.SubElement(div_petition, "head")
            label_petition = ET.SubElement(head_petition, "label")
            label_petition.text = data[i]['text_ocr']
            divs_cibles.append(div_petition)

        elif re.search(r"part1-petition", data[i]["comment"]):
            # Ajouter un élément 'part1-petition' avec le contenu de la clé 'text_ocr'
            div_petition1 = ET.SubElement(back, "div", attrib={"type": "petition"})
            div_petition1.addprevious(etree.Comment(generate_id("Partie-1_petition_")))
            divs_cibles.append(div_petition1)
            head_petition = ET.SubElement(div_petition1, "head")
            label_petition1 = ET.SubElement(head_petition1, "label")
            label_petition1.text = data[i]['text_ocr']

        elif data[i]['comment'] == "meeting-session meeting-legislature useless":
            # Ajouter un élément 'meeting-session meeting-legislature' avec le contenu de la clé 'text_ocr'
            meet_session = data[i]['text_ocr']

            try :
                # extrait le numéro de session
                num_session = int(meet_session.split(' ')[-1][:-4])
                num_legis = int(meet_session.split(' ')[0][:-1])

                # extrait le texte de la session
                texte_session = "Session " + meet_session.split('Session ')[1]  # ajoute le mot "Session"
                # extrait la première lettre du mot suivant "Session"
                lettre_session = texte_session.split('Session ')[1][0].upper()
                # créer la balise
                meeting_session = ET.SubElement(titleStmt, "meeting", n=f"E{num_session}", ana="#parla.lower\n#parla.session") # un
                meeting_session.text = f"{texte_session}"

                meet_legis = meet_session.split('—')
                meeting_legislature = ET.SubElement(titleStmt, "meeting", n=f"{num_legis}
                meeting_legislature.text = meet_legis[0]

                meeting_session_t = ET.SubElement(titleStmt, "meeting", n=f"E{num_session}", ana="#parla.lower\n#parla.session") #
                meeting_session_t.text = f"{texte_session}".strip(".")
                titleStmt.insert(4, meeting_session_t)

                meet_legis = meet_session.split('—')
                meeting_legislature_t = ET.SubElement(titleStmt, "meeting", n=f"{num_legis}L", ana="#parla.lower\n#parla.legislatur
                meeting_legislature_t.text = meet_legis[0].split(".")[0].replace("°", "e")
                titleStmt.insert(5, meeting_legislature_t)

            except ValueError as e:
                # Gérer l'exception ValueError ici
                print(f"Erreur lors de la conversion en entier : {e}")

        elif data[i]['comment'] == "meeting-sitting useless" or data[i]['comment'] == "meeting-sitting":
            # Ajouter un élément 'meeting-sitting' avec le contenu de la clé 'text_ocr'
            meet_sit = data[i]['text_ocr'].split('—')
            num_seance = meet_sit[1][0:3]
            meeting_sitting = ET.SubElement(titleStmt, "meeting", n=f"{num_seance}", ana="#parla.lower\n#parla.sitting")
            meeting_sitting.text = data[i]['text_ocr']
            meeting_sitting_t = ET.SubElement(titleStmt, "meeting", n=f"{num_seance}", ana="#parla.lower\n#parla.sitting")
            meeting_sitting_t.text = data[i]['text_ocr'].split("—")[1].strip(".").replace("°", "e").lower()
            titleStmt.insert(6, meeting_sitting_t)


        #__________________Commentaires et notes_____________________

        elif data[i]['comment'] == 'note-head':
            # Ajouter un élément 'note-head' avec le contenu de la clé 'text_ocr'
            note_head = ET.Element("note")
            note_head.text = data[i]['text_ocr']
            # Ajouter les éléments <note-head> à chaque div cible
            div_cible.append(note_head)

        elif re.search(r"voterslist-beginning", data[i]["comment"]):
            note_voterlist = ET.Element("note", attrib={"type": "voterslist"})
            voterlist = ET.SubElement(note_voterlist, "desc")
            voterlist.text = data[i]['text_ocr']
            div_cible.append(note_voterlist)

        if data[i]['comment'] == 'comment seg' or data[i]["comment"] == "seg comment" or data[i]['comment'] == 'comment' :
```

```python
            # Ajouter un élément 'comment seg' avec le contenu de la clé 'text_ocr'
            comment_note = ET.Element("note", attrib={"type": "comment"})
            comment_seg = ET.SubElement(comment_note, "seg")
            comment_seg.text = data[i]['text_ocr']
            div_cible.append(comment_note)
            if comment_note is not None:
                comment_note.tail = " "

        if data[i]['comment'] == 'comment-beginning seg':
            # Ajouter un élément 'comment-beginning seg' avec le contenu de la clé 'text_ocr'
            comment_beg_note = ET.Element("note", attrib={"type": "comment"})
            comment_beginning_seg = ET.SubElement(comment_beg_note, "seg")
            comment_beginning_seg.text = data[i]['text_ocr']
            div_cible.append(comment_beg_note)

        if data[i]['comment'] == 'comment-end seg':
            # Ajouter un élément 'comment-end seg' avec le contenu de la clé 'text_ocr'
            comment_end_seg = ET.Element("seg")
            comment_end_seg.text = data[i]['text_ocr']
            comment_beg_note.append(comment_end_seg)

            if comment_note is not None:
                comment_note.tail = " "
            #comment_note = ET.Element("note", attrib={"type": "comment"})

        if data[i]['comment'] == 'note seg' or data[i]['comment'] == 'note-seg':
            # Ajouter un élément 'note seg' avec le contenu de la clé 'text_ocr'
            note_seg = ET.Element("seg")
            note_seg.text = data[i]['text_ocr']
            comment_note.append(note_seg)
            div_cible.append(comment_note)

        if data[i]['comment'] == 'result':
            # Ajouter un élément 'result' avec le contenu de la clé 'text_ocr'
            note_result = ET.Element("note", attrib={"type": "result"})
            note_result.text = data[i]['text_ocr']
            div_cible.append(note_result)

        if re.search(r"note-beginning", data[i]["comment"]):
            global note_beg
            note_beg = ET.SubElement(div_cible, "note", attrib={"type": "numbersannounced"})
            global note_beg_seg
            note_beg_seg = ET.SubElement(note_beg, "seg")
            note_beg_seg.text = data[i]['text_ocr']

        if re.search(r"note-end", data[i]["comment"]) and "div-end" not in data[i]["comment"] :
            note_end_seg = ET.Element("seg")
            note_end_seg.text = data[i]['text_ocr']
            if note_beg_seg is None:
```

```
                        pass
                    else :
                        note_beg_seg.append(note_end_seg)

                if data[i]['comment'] == 'signed seg back':
                    # Ajouter un élément 'signed seg back' avec le contenu de la clé 'text_ocr'

                    signed = ET.Element("signed")
                    signed_seg_back = ET.SubElement(signed, "seg")
                    signed_seg_back.text = data[i]['text_ocr']
                    div_cible.append(signed)

                if "seg note-end div-end" in data[i]['comment']:
                    note_end_div_end = ET.Element("seg")
                    note_end_div_end.text = data[i]['text_ocr']
                    note_voterlist.append(note_end_div_end)

                #__________________Items_____________________


                elif re.search(r"\bitem(?!-)\b", data[i]["comment"]):
                    item = ET.SubElement(list_item, "item")
                    item.text = data[i]["text_ocr"]

                elif re.search(r"item-list", data[i]["comment"]):
                    item = ET.SubElement(list_item, "item")
                    item.text = data[i]["text_ocr"]


                #__________________ajout des Utterances, des seg et tables_____________________

                if data[i]["comment"] == "u-beginning seg" :
                    u_beg = ET.Element("seg")
                    u_beg.text = data[i]['text_ocr']
                    u_element.append(u_beg)

                if data[i]["comment"] == "part1 u-beginning seg" :
                    u_beg = ET.Element("seg")
                    u_beg.text = data[i]['text_ocr']
                    u_element.append(u_beg)
                    div_part1.append(u_element)

                if data[i]["comment"] == "u-beginning seg-beginning" :
                    global u_beg_seg_beg
                    u_beg_seg_beg = ET.Element("seg")
                    u_beg_seg_beg.text = data[i]['text_ocr']
                    u_element.append(u_beg_seg_beg)

                if data[i]["comment"] == "u-beginning seg-beginning incident" :
```

```python
            u_seg_beg_inc = ET.Element("seg")
            u_seg_beg_inc.text = data[i]['text_ocr']
            u_element.append(u_seg_beg_inc)

        if data[i]["comment"] == "u-end seg-end incident" or data[i]["comment"] == "seg-end incident" :

            text_ocr = data[i]["text_ocr"]

            # Rechercher l'indice de la première occurrence de "(" et de ")" dans le texte
            start_index = text_ocr.find("(")
            end_index = text_ocr.find(")")

            if start_index != -1 and end_index != -1 and start_index < end_index:
                # Extraire la partie du texte entre les guillemets
                incident_text = text_ocr[start_index :end_index+1]

                # Créer un élément <incident> et y ajouter le texte extrait

                incident_tag = ET.Element("incident")
                incident_desc = ET.SubElement(incident_tag, "desc")
                incident_desc.text = incident_text
                incident_tag.tail = text_ocr[end_index + 1:]

                 # Insérer l'élément <incident> dans la bonne position en utilisant les méthodes d'insertion

                if u_beg_seg_beg is not None:
                    u_beg_seg_beg.text += "" + text_ocr[:start_index]
                    u_beg_seg_beg.insert(1, incident_tag)
                    incident_tag.tail = text_ocr[end_index + 1:]
                    u_element.append(u_beg_seg_beg)
                else :
                    u_element.text = text_ocr[:start_index]
                    u_element.append(incident_tag)
                    u_element.tail = text_ocr[end_index + 1:]

            if u_element is not None:
                u_element.tail = " "
            u_element = ET.Element("u")

        if data[i]["comment"] == "u-beginning seg quote" :
            seg_cas = ET.Element("seg")
            text_ocr = data[i]["text_ocr"]

            # Rechercher l'indice de la première occurrence de "«" et de "»" dans le texte
            start_index = text_ocr.find("«")
            end_index = text_ocr.find("»")

            if start_index != -1 and end_index != -1 and start_index < end_index:
                # Extraire la partie du texte entre les guillemets
                quote_text = text_ocr[start_index :end_index+1]

                # Créer un élément <quote> et y ajouter le texte extrait
                quote_seg = ET.SubElement(seg_cas, "quote")
                quote_seg.text = quote_text

                 # Insérer l'élément <quote> dans la bonne position en utilisant les méthodes d'insertion
                seg_cas.text = text_ocr[:start_index]
                seg_cas.insert(1, quote_seg)
                quote_seg.tail = text_ocr[end_index + 1:]

                u_element.append(seg_cas)

            else:
                # Aucun guillemet trouvé ou l'ordre est incorrect, utiliser le texte tel quel
                seg_cas.text = text_ocr
            u_element.append(seg_cas)


        if data[i]["comment"] == "quote-beginning seg" or data[i]["comment"] == "seg quote-beginning" :

            quote_beg_seg = ET.Element("seg")
            quote_beg_seg.text = data[i]['text_ocr']
            if quote is not None :
                quote.append(quote_beg_seg)

        if "quote-beginning" in data[i]["comment"] and "seg-beginning" in data[i]["comment"] :
            quote_seg_beg = ET.Element("quote")
            global quote_beg_seg_beg
            quote_beg_seg_beg = ET.SubElement(quote_seg_beg,"seg")
            quote_beg_seg_beg.text = data[i]['text_ocr']
            u_element.append(quote_seg_beg)


        if  "quote-end" in data[i]["comment"] and "seg-end" in data[i]["comment"] :
            if quote_beg_seg_beg is not None:
                quote_beg_seg_beg.text += "" + "\n"+ data[i]['text_ocr']

        if "u-beginning" in data[i]["comment"]:
            # Créer un nouvel élément 'u' et le définir comme parent actuel
            current_parent = u_element

        elif "quote-beginning" in data[i]["comment"]:
            # Créer un nouvel élément 'quote' et le définir comme parent actuel
            quote = ET.Element("quote")
            current_parent.append(quote)
            current_parent = quote

        elif "quote-beginning" in data[i]["comment"] and "seg-beginning" in data[i]["comment"]:
```

```
                    # Créer un nouvel élément 'quote' et le définir comme parent actuel
                    current_parent.append(quote_seg_beg)
                    current_parent = quote_seg_beg

                elif "quote-end" in data[i]["comment"]:
                    # Retourner à l'élément parent précédent s'il s'agit de la balise "quote"
                    if current_parent is quote:
                        current_parent = current_parent.getparent()

                elif "voterslist-beginning" in data[i]["comment"]:
                    # Créer un nouvel élément 'voterslist' et le définir comme parent actuel
                    current_parent.append(note_voterlist)
                    current_parent = note_voterlist

                elif data[i]['comment'] == 'comment seg' or data[i]["comment"] == "seg comment" or data[i]['comment'] == 'comment' :
                    current_parent.append(comment_note)
                    #current_parent = comment_note

                elif "u-end" in data[i]["comment"] or "note-end" in data[i]["comment"]:
                    # Retourner à l'élément parent précédent
                    if current_parent is not None:
                        current_parent = current_parent.getparent()

                elif 'comment-beginning' in data[i]['comment'] :
                    if comment_beg_note is not None :
                        current_parent.append(comment_beg_note)
                        current_parent = comment_beg_note

                elif 'comment-end' in data[i]['comment'] :
                    if current_parent is not None:
                        current_parent = current_parent.getparent()

                if data[i]["comment"] == "seg":
                    seg = ET.Element("seg")
                    seg.text = data[i]['text_ocr']

                    if i >= 0 and "comment" in data[i-1] and ("quote-end" in data[i-1]["comment"] or "table" in data[i-1]["comment"] or "si
                        while i < len(data) - 1 and "comment" in data[i+1] and "u-end" not in data[i]["comment"]:
                            if data[i]["comment"] == "seg":
                                if data[i]['text_ocr'] not in added_segs:  # Vérifier si l'élément <seg> existe déjà dans l'ensemble
                                    seg = ET.Element("seg")
                                    seg.text = data[i]['text_ocr']
                                    u_element.append(seg)
                                    added_segs.add(data[i]['text_ocr'])  # Ajouter l'élément à l'ensemble des éléments ajoutés
                            i += 1
                    if i >= 0 and "comment" in data[i-1] and ("seg incident" in data[i-1]["comment"] or "seg quote incident" in data[i-1]["
                        while i < len(data) - 1 and "comment" in data[i+1] and "u-end" not in data[i]["comment"]:
                            if data[i]["comment"] == "seg":
                                if data[i]['text_ocr'] not in added_segs:  # Vérifier si l'élément <seg> existe déjà dans l'ensemble
                                    seg = ET.Element("seg")
                                    seg.text = data[i]['text_ocr']
                                    u_element.append(seg)
                                    added_segs.add(data[i]['text_ocr'])  # Ajouter l'élément à l'ensemble des éléments ajoutés
                            i += 1

                    if current_parent is u_element:
                        # Ajouter les balises 'seg' à l'élément 'u'
                        if u_element is not None:
                            u_element.append(seg)

                    elif current_parent is quote:
                        # Ajouter les balises 'seg' à l'élément 'quote'
                        quote.append(seg)
                        u_element.append(quote)

                    elif current_parent is note_voterlist:
                        # Ajouter les balises 'seg' à l'élément 'voterslist'
                        note_voterlist.append(seg)
                        div_cible.append(note_voterlist)

                    elif current_parent is comment_note :
                        comment_note.append(seg)
                        div_cible.append(comment_note)

                    elif current_parent is comment_beg_note:
                        comment_beg_note.append(seg)
                        div_cible.append(comment_beg_note)

                    elif current_parent is signed:
                        # Ajouter les balises 'seg' à l'élément 'signed'
                        signed.append(seg)
                        div_cible.append(signed)

                if data[i]['comment'] == 'table':
                    # Ajouter un élément 'table' avec le contenu de la clé 'text_ocr'
                    table = ET.Element("table")
                    row = ET.SubElement(table, "row")
                    cell = ET.SubElement(row, "cell")
                    cell.text = data[i]['text_ocr']

                    if current_parent is u_element:
                        # Ajouter les balises 'seg' à l'élément 'u'
                        if u_element is not None:
                            u_element.append(table)

                    if current_parent is quote:
                        # Ajouter les balises 'seg' à l'élément 'quote'
                        quote.append(table)
```

```
                        quote.append(table)

                    if current_parent is note_voterlist:
                        # Ajouter les balises 'seg' à l'élément 'voterslist'
                        note_voterlist.append(table)

                if data[i]["comment"] == "seg-beginning" :
                    global seg_beg
                    seg_beg = ET.Element("seg")
                    seg_beg.text = data[i]['text_ocr']

                    if current_parent is u_element:
                        # Ajouter les balises 'seg' à l'élément 'u'
                        if u_element is not None and seg_beg is not None:
                            u_element.append(seg_beg)

                    if current_parent is quote:
                        # Ajouter les balises 'seg' à l'élément 'quote'
                        quote.append(seg_beg)

                    if current_parent is note_voterlist:
                        # Ajouter les balises 'seg' à l'élément 'voterslist'
                        note_voterlist.append(seg_beg)

                if data[i]["comment"] == "seg-end" :
                    if seg_beg is not None :
                        seg_beg.text += " " + data[i]['text_ocr']

                if data[i]['comment'] == 'desc':
                    # Ajouter un élément 'desc' avec le contenu de la clé 'text_ocr'
                    voterlist_desc = ET.SubElement(note_voterlist, "desc")
                    voterlist_desc.text = data[i]['text_ocr']


# ___________________________________Fin ___seg ___


                if data[i]["comment"]== "seg-end quote-end" :
                        if quote_beg_seg is not None :
                            quote_beg_seg.text += " " + data[i]['text_ocr']

                if data[i]["comment"] == "u seg" :
                    u_seg = ET.Element("seg")
                    u_seg.text = data[i]['text_ocr']
                    u_element.append(u_seg)

                    if u_element is not None:
                        u_element.tail = " "
                    u_element = ET.Element("u")
```

```python
if data[i]["comment"] == "u seg quote" or data[i]["comment"] == "u-end seg quote" or data[i]["comment"] == "seg quote":
    u_seg_quote = ET.Element("seg")

    text_ocr = data[i]["text_ocr"]
    # Rechercher l'indice de la première occurrence de "«" et de "»" dans le texte
    start_index = text_ocr.find("«")
    end_index = text_ocr.find("»")

    if start_index != -1 and end_index != -1 and start_index < end_index:
        # Extraire la partie du texte entre les guillemets
        quote_text = text_ocr[start_index :end_index+1]

        # Créer un élément <quote> et y ajouter le texte extrait
        quote_seg = ET.SubElement(u_seg_quote, "quote")
        quote_seg.text = quote_text

         # Insérer l'élément <quote> dans la bonne position en utilisant les méthodes d'insertion
        u_seg_quote.text = text_ocr[:start_index]
        u_seg_quote.insert(1, quote_seg)
        quote_seg.tail = text_ocr[end_index + 1:]
        u_element.append(u_seg_quote)

    else:
    # Aucun guillemet trouvé ou l'ordre est incorrect, utiliser le texte tel quel
        u_seg_quote.text = text_ocr
        u_element.append(u_seg_quote)

    if u_element is not None:
        u_element.tail = " "
    u_element = ET.Element("u")


if data[i]["comment"] == "u seg incident" :

    text_ocr = data[i]["text_ocr"]

    # Rechercher l'indice de la première occurrence de "(" et de ")" dans le texte
    start_index = text_ocr.find("(")
    end_index = text_ocr.find(")")

    if start_index != -1 and end_index != -1 and start_index < end_index:
        # Extraire la partie du texte entre les guillemets
        incident_text = text_ocr[start_index :end_index+1]

        # Créer un élément <incident> et y ajouter le texte extrait
        u_seg_inc = ET.Element("seg")
        incident_tag = ET.SubElement(u_seg_inc, "incident")
        incident_desc = ET.SubElement(incident_tag, "desc")
```

```
                        incident_desc.text = incident_text

                         # Insérer l'élément <incident> dans la bonne position en utilisant les méthodes d'insertion
                        u_seg_inc.text = text_ocr[:start_index]
                        u_seg_inc.insert(1, incident_tag)
                        incident_tag.tail = text_ocr[end_index + 1:]
                        u_element.append(u_seg_inc)

                    else :
                        u_seg_inc = ET.Element("seg")
                        incident_tag = ET.SubElement(u_seg_inc, "incident")
                        incident_desc = ET.SubElement(incident_tag, "desc")
                        incident_desc.text = data[i]['text_ocr']
                        u_element.append(u_seg_inc)

                    if u_element is not None:
                        u_element.tail = " "
                    u_element = ET.Element("u")


                if data[i]["comment"] == "u-beginning seg incident" :
                    text_ocr = data[i]["text_ocr"]

                    # Rechercher l'indice de la première occurrence de "(" et de ")" dans le texte
                    start_index = text_ocr.find("(")
                    end_index = text_ocr.find(")")

                    if start_index != -1 and end_index != -1 and start_index < end_index:
                        # Extraire la partie du texte entre les guillemets
                        incident_text = text_ocr[start_index :end_index+1]

                        # Créer un élément <incident> et y ajouter le texte extrait
                        u_seg_inc = ET.Element("seg")
                        incident_tag = ET.SubElement(u_seg_inc, "incident")
                        incident_desc = ET.SubElement(incident_tag, "desc")
                        incident_desc.text = incident_text

                         # Insérer l'élément <incident> dans la bonne position en utilisant les méthodes d'insertion
                        u_seg_inc.text = text_ocr[:start_index]
                        u_seg_inc.insert(1, incident_tag)
                        incident_tag.tail = text_ocr[end_index + 1:]
                        u_element.append(u_seg_inc)

                    else :
                        u_seg_inc = ET.Element("seg")
                        incident_tag = ET.SubElement(u_seg_inc, "incident")
                        incident_desc = ET.SubElement(incident_tag, "desc")
                        incident_desc.text = data[i]['text_ocr']
                        u_element.append(u_seg_inc)
```

```python
        if data[i]["comment"] == "seg incident" :

            text_ocr = data[i]["text_ocr"]

            # Rechercher l'indice de la première occurrence de "(" et de ")" dans le texte
            start_index = text_ocr.find("(")
            end_index = text_ocr.find(")")

            if start_index != -1 and end_index != -1 and start_index < end_index:
                # Extraire la partie du texte entre les guillemets
                incident_text = text_ocr[start_index :end_index+1]

                # Créer un élément <incident> et y ajouter le texte extrait
                seg_inci = ET.Element("seg")
                incident_tag = ET.SubElement(seg_inci, "incident")
                incident_desc = ET.SubElement(incident_tag, "desc")
                incident_desc.text = incident_text

                 # Insérer l'élément <incident> dans la bonne position en utilisant les méthodes d'insertion
                seg_inci.text = text_ocr[:start_index]
                seg_inci.insert(1, incident_tag)
                incident_tag.tail = text_ocr[end_index + 1:]
                u_element.append(seg_inci)

        if data[i]["comment"] == "u-end seg-end" or data[i]["comment"] == "seg-end u-end" :

            u_beg_seg_beg.text += "" + data[i]['text_ocr']
            u_element.append(u_beg_seg_beg)

            if u_element is not None:
                u_element.tail = " "
            u_element = ET.Element("u")


        if data[i]["comment"] == "u-end seg incident" or data[i]["comment"] == "seg incident u-end":
            u_end_seg_inc = ET.Element("seg")
            text_ocr = data[i]["text_ocr"]

            # Rechercher l'indice de la première occurrence de "(" et de ")" dans le texte
            start_index = text_ocr.find("(")
            end_index = text_ocr.find(")")

            if start_index != -1 and end_index != -1 and start_index < end_index:
                # Extraire la partie du texte entre parentèses
                incident_text = text_ocr[start_index :end_index+1]

                # Créer un élément <incident> et y ajouter le texte extrait
                incident_tag = ET.Element("incident")
                incident_desc = ET.SubElement(incident_tag, "desc")
                incident_desc.text = incident_text

                 # Insérer l'élément <incident> dans la bonne position en utilisant les méthodes d'insertion

                u_end_seg_inc.text = text_ocr[:start_index]
                u_end_seg_inc.insert(1, incident_tag)
                incident_tag.tail = text_ocr[end_index + 1:]
                u_element.append(u_end_seg_inc)

                if u_element is not None:
                    u_element.tail = " "
                u_element = ET.Element("u")


        if data[i]["comment"] == "seg quote incident":
            seg_quote = ET.Element("seg")
            text_ocr = data[i]["text_ocr"]

            # Rechercher l'indice de la première occurrence de "«" et de "»" dans le texte
            start_index = text_ocr.find("«")
            end_index = text_ocr.find("»")

            start_index1 = text_ocr.find("(")
            end_index1 = text_ocr.find(")")

            if start_index != -1 and end_index != -1 and start_index < end_index:
                # Extraire la partie du texte entre les guillemets
                quote_text = text_ocr[start_index:end_index+1]
                incident_text = text_ocr[start_index1:end_index1+1]
                # Créer un élément <quote> et y ajouter le texte extrait
                quote_seg = ET.Element("quote")
                quote_seg.text = quote_text

                quote_incident = ET.Element("incident")
                quote_incident_desc = ET.SubElement(quote_incident, "desc")
                quote_incident_desc.text = incident_text

                # Insérer l'élément <quote> dans la bonne position en utilisant les méthodes d'insertion
                seg_quote.text = text_ocr[:start_index1]
                seg_quote.append(quote_seg)
                seg_quote.append(quote_incident)
                quote_incident.tail = text_ocr[end_index1 + 1:]

            u_element.append(seg_quote)


        if data[i]["comment"] == "seg-end quote" :
            seg_end_quote = ET.Element("seg")
            seg_end_quote.text = data[i]['text_ocr']

            text_ocr = data[i]["text_ocr"]
            # Rechercher l'indice de la première occurrence de "«" et de "»" dans le texte
            start_index = text_ocr.find("«")
            end_index = text_ocr.find("»")

            if start_index != -1 and end_index != -1 and start_index < end_index:
                # Extraire la partie du texte entre les guillemets
                quote_text = text_ocr[start_index :end_index+1]

                # Créer un élément <quote> et y ajouter le texte extrait
                quote_end_seg = ET.SubElement(seg_end_quote, "quote")
                quote_end_seg.text = quote_text

                 # Insérer l'élément <quote> dans la bonne position en utilisant les méthodes d'insertion
                seg_end_quote.text = text_ocr[:start_index]
                seg_end_quote.insert(1, quote_seg)
                quote_end_seg.tail = text_ocr[end_index + 1:]

            if seg_beg is not None :
                seg_beg.text += "" + seg_end_quote.text
                #u_element.append(seg_beg)

        if data[i]["comment"] == "quote-end seg" :

            quote_end_seg = ET.Element("seg")
            quote_end_seg.text = data[i]['text_ocr']
            if quote is not None :
                quote.append(quote_end_seg)

        if data[i]["comment"] == "opening seg" :
            # Ajouter un élément 'opening seg' avec le contenu de la clé 'text_ocr'
            opening_note = ET.Element("note", { "{http://www.w3.org/XML/1998/namespace}id": "CR_" + filename.split("_")[3] + "_" +
            opening_seg = ET.SubElement(opening_note, "seg", { "{http://www.w3.org/XML/1998/namespace}id": generate_id("s")})
            opening_seg.text = data[i]['text_ocr']
            div_cible.append(opening_note)

        elif data[i]['comment'] == 'closing seg':
            # Ajouter un élément 'closing seg' avec le contenu de la clé 'text_ocr'
            note_closing = ET.Element("note", {"type": "closing", "{http://www.w3.org/XML/1998/namespace}id": generate_id("note")})
            closing_seg = ET.SubElement(note_closing, "seg")
            closing_seg.text = data[i]['text_ocr']
            div_cible.append(note_closing)

        if 'page-number' in data[i]['comment']:

            if current_parent is u_element:
```

```
                        # Ajouter les balises 'seg' à l'élément 'u'
                        u_element.append(bp_element)

                    elif current_parent is quote:
                        # Ajouter les balises 'seg' à l'élément 'quote'
                        quote.append(bp_element)

                    elif current_parent is quote_seg_beg:
                        # Ajouter les balises 'seg' à l'élément 'quote'
                        quote_seg_beg.append(bp_element)

                    elif current_parent is note_voterlist:
                        # Ajouter les balises 'seg' à l'élément 'voterslist'
                        note_voterlist.append(bp_element)


                    elif current_parent is comment_note :
                        comment_note.append(bp_element)


                    elif current_parent is comment_beg_note:
                        comment_beg_note.append(bp_element)


                    else :
                        for div_cible in divs_cibles:
                            div_cible.append(bp_element)


                for div_cible in divs_cibles:
                    if div_cible is not None:
                        div_cible.append(u_element)
                    if comment_note is not None :
                        div_cible.append(comment_note)




xml_tree = ET.ElementTree(root)

final_output = filename.split("_p0")[0] + "_compiled" + ".xml"
xml_filename = os.path.join(dossier_xml, final_output )

with open(os.path.join(dossier_xml, xml_filename), "wb") as xml_file:
     root = xml_tree.getroot()

     instruction1 = ' href="agoda_schema.rng" type="application/xml" schematypens="http://purl.oclc.org/dsdl/schematron"'
     model_instr1 = ET.ProcessingInstruction("xml-model", instruction1)
```

```python
     root.addprevious(model_instr1)

     instruction2 = ' href="agoda_schema.rng" type="application/xml" schematypens="http://relaxng.org/ns/structure/1.0"'
     model_instr2 = ET.ProcessingInstruction("xml-model", instruction2)
     root.addprevious(model_instr2)

     xml_tree.write(xml_file, encoding="utf-8", xml_declaration=True, pretty_print=True)



# ___________________Fin du script en haut________Nettoyage, ci-dessous_______________________________________________________

# suppressions des tirets suivis de retours à la ligne

# Chemin d'accès au dossier contenant les fichiers XML
dossier_xml = os.path.join(os.getcwd(), "xml_data")

# Parcourir les fichiers XML dans le dossier
for nom_fichier in os.listdir(dossier_xml):
    if nom_fichier.endswith(".xml"):
        chemin_fichier = os.path.join(dossier_xml, nom_fichier)

        # Lire le contenu du fichier XML
        with open(chemin_fichier, "r") as fichier:
            contenu = fichier.read()

        # Supprimer les tirets suivis de retours à la ligne entre deux mots
        contenu_modifie = re.sub(r"(\w)-\n(\w)", r"\1\2", contenu)
        contenu_modifie = re.sub("-\n", "", contenu_modifie)
        contenu_modifie = re.sub(r"-\s", "", contenu_modifie)


        # Écrire le contenu modifié dans le fichier
        with open(chemin_fichier, "w") as fichier_modifie:
            fichier_modifie.write(contenu_modifie)

        #print(f"Le fichier '{nom_fichier}' a été modifié.")
```

In [ ]:

In [ ]: