以下はイタリア語版のstreamlit cloud アプリ用のpythonコード群である。アプリにおいては、1つ目のコードの”main.py”が中心として実行され、2つ目のコードの”Pagina per generare file JSON per sostituire testi in esperanto con stringhe (kanji).py”はstreamlit特有のpagesフォルダの中のコードとして実行される。3つ目(esp\_text\_replacement\_module.py)、4つ目(esp\_replacement\_json\_make\_module.py)のコードはアプリ内で用いるpythonモジュールを多数収録したコードである。

これらのコードをじっくり読み込んで、内容を深く理解した上で、このアプリのイタリア語圏のGUIユーザーに向けた、このアプリの使い方の丁寧でわかりやすい説明書をイタリア語でじっくり丁寧に作成してほしい。大変負荷の高いタスクですので、回答が複数回に渡っても一向に構いません。あなたが、これぞこのアプリのイタリア語圏のGUIユーザーにとって完璧な説明書だと思うものをじっくり丁寧に書き上げていただきたいと思っています。

## main.py(1つ目)

# main.py (メインの Streamlit アプリ/機能拡充版202502)

import streamlit as st

import re

import io

import json

import pandas as pd # 必要なら使う

from typing import List, Dict, Tuple, Optional

import streamlit.components.v1 as components

import multiprocessing

#=================================================================

# Streamlit で multiprocessing を使う際、PicklingError 回避のため

# 明示的に 'spawn' モードを設定する必要がある。

#=================================================================

try:

multiprocessing.set\_start\_method("spawn")

except RuntimeError:

pass # すでに start method が設定済みの場合はここで無視する

#=================================================================

# エスペラント文の(漢字)置換・ルビ振りなどを行う独自モジュールから

# 関数をインポートする。

# esp\_text\_replacement\_module.py内に定義されているツールをまとめて呼び出す

#=================================================================

from esp\_text\_replacement\_module import (

x\_to\_circumflex,

x\_to\_hat,

hat\_to\_circumflex,

circumflex\_to\_hat,

replace\_esperanto\_chars,

import\_placeholders,

orchestrate\_comprehensive\_esperanto\_text\_replacement,

parallel\_process,

apply\_ruby\_html\_header\_and\_footer

)

#=================================================================

# Streamlit の @st.cache\_data デコレータを使い、読み込み結果をキャッシュして

# JSONファイルのロード高速化を図る。大きなJSON(50MB程度)を都度読むと遅いので、

# ここで呼び出す関数をキャッシュする作り。

#=================================================================

@st.cache\_data

def load\_replacements\_lists(json\_path: str) -> Tuple[List, List, List]:

"""

JSONファイルをロードし、以下の3つのリストをタプルとして返す:

1) replacements\_final\_list

2) replacements\_list\_for\_localized\_string

3) replacements\_list\_for\_2char

"""

with open(json\_path, 'r', encoding='utf-8') as f:

data = json.load(f)

replacements\_final\_list = data.get(

"全域替换用のリスト(列表)型配列(replacements\_final\_list)", []

)

replacements\_list\_for\_localized\_string = data.get(

"局部文字替换用のリスト(列表)型配列(replacements\_list\_for\_localized\_string)", []

)

replacements\_list\_for\_2char = data.get(

"二文字词根替换用のリスト(列表)型配列(replacements\_list\_for\_2char)", []

)

return (

replacements\_final\_list,

replacements\_list\_for\_localized\_string,

replacements\_list\_for\_2char,

)

#=================================================================

# Streamlit ページの見た目設定

# page\_title: ブラウザタブに表示されるタイトル

# layout="wide" で横幅を広く使えるUIにする

#=================================================================

st.set\_page\_config(

page\_title="Strumento di sostituzione di caratteri (kanji) per il testo in esperanto",

layout="wide"

)

# タイトル部分（GUI表示部分のみイタリア語に）

st.title("Sostituzione del testo in esperanto con kanji o annotazioni in HTML (versione estesa)")

st.write("---")

#=================================================================

# 1) JSONファイル (置換ルール) をロード

# (デフォルトを使うか、ユーザーがアップロードするかの選択)

#=================================================================

json\_options = ["デフォルトを使用する", "アップロードする"]

selected\_option = st.radio(

"Come gestire il file JSON? (lettura del file JSON per la sostituzione)",

json\_options,

format\_func=lambda x: "Usa il file JSON predefinito" if x == "デフォルトを使用する" else "Carica un file"

)

with st.expander("Scarica un file JSON di esempio (per la sostituzione)"):

json\_file\_path = './Appの运行に使用する各类文件/最终的な替换用リスト(列表)(合并3个JSON文件).json'

with open(json\_file\_path, "rb") as file\_json:

btn\_json = st.download\_button(

label="Scarica il file JSON di esempio",

data=file\_json,

file\_name="file\_JSON\_esempio.json",

mime="application/json"

)

replacements\_final\_list: List[Tuple[str, str, str]] = []

replacements\_list\_for\_localized\_string: List[Tuple[str, str, str]] = []

replacements\_list\_for\_2char: List[Tuple[str, str, str]] = []

# JSONファイルの読み込み方を分岐

if selected\_option == "デフォルトを使用する":

default\_json\_path = "./Appの运行に使用する各类文件/最终的な替换用リスト(列表)(合并3个JSON文件).json"

try:

(replacements\_final\_list,

replacements\_list\_for\_localized\_string,

replacements\_list\_for\_2char) = load\_replacements\_lists(default\_json\_path)

st.success("Il file JSON predefinito è stato caricato con successo.")

except Exception as e:

st.error(f"Impossibile caricare il file JSON predefinito: {e}")

st.stop()

else:

uploaded\_file = st.file\_uploader(

"Carica un file JSON (in un formato combinato con 3 liste).json", type="json"

)

if uploaded\_file is not None:

try:

combined\_data = json.load(uploaded\_file)

replacements\_final\_list = combined\_data.get(

"全域替换用のリスト(列表)型配列(replacements\_final\_list)", []

)

replacements\_list\_for\_localized\_string = combined\_data.get(

"局部文字替换用のリスト(列表)型配列(replacements\_list\_for\_localized\_string)", []

)

replacements\_list\_for\_2char = combined\_data.get(

"二文字词根替换用のリスト(列表)型配列(replacements\_list\_for\_2char)", []

)

st.success("Il file JSON è stato caricato con successo.")

except Exception as e:

st.error(f"Impossibile leggere il file JSON caricato: {e}")

st.stop()

else:

st.warning("Nessun file JSON è stato caricato. Interruzione del processo.")

st.stop()

#=================================================================

# 2) placeholders (占位符) の読み込み

# %...% や @...@ で囲った文字列を守るために使用する文字列群を読み込む

#=================================================================

placeholders\_for\_skipping\_replacements: List[str] = import\_placeholders(

'./Appの运行に使用する各类文件/占位符(placeholders)\_%1854%-%4934%\_文字列替换skip用.txt'

)

placeholders\_for\_localized\_replacement: List[str] = import\_placeholders(

'./Appの运行に使用する各类文件/占位符(placeholders)\_@5134@-@9728@\_局部文字列替换结果捕捉用.txt'

)

st.write("---")

#=================================================================

# 設定パラメータ (UI) - 高度な設定

# 並列処理 (multiprocessing) を利用できるかどうかのスイッチと、

# 同時プロセス数の選択

#=================================================================

st.header("Parametri avanzati (elaborazione parallela)")

with st.expander("Apri le impostazioni per l'elaborazione parallela"):

st.write("""

Qui puoi definire il numero di processi da eseguire in parallelo

durante la sostituzione dei caratteri (kanji).

""")

use\_parallel = st.checkbox("Usa l'elaborazione parallela", value=False)

num\_processes = st.number\_input(

"Numero di processi simultanei",

min\_value=2, max\_value=4, value=4, step=1

)

st.write("---")

#=================================================================

# 例: 出力形式の選択

# (HTMLルビ形式・括弧形式・文字列のみ など)

#=================================================================

options = {

'HTML格式\_Ruby文字\_大小调整': 'HTML格式\_Ruby文字\_大小调整',

'HTML格式\_Ruby文字\_大小调整\_汉字替换': 'HTML格式\_Ruby文字\_大小调整\_汉字替换',

'HTML格式': 'HTML格式',

'HTML格式\_汉字替换': 'HTML格式\_汉字替换',

'括弧(号)格式': '括弧(号)格式',

'括弧(号)格式\_汉字替换': '括弧(号)格式\_汉字替换',

'替换后文字列のみ(仅)保留(简单替换)': '替换后文字列のみ(仅)保留(简单替换)'

}

# ユーザーに見せるイタリア語ラベルをマッピング

options\_italian\_labels = {

'HTML格式\_Ruby文字\_大小调整': "Formato HTML con annotazioni (ruby) e regolazione delle dimensioni",

'HTML格式\_Ruby文字\_大小调整\_汉字替换': "Formato HTML con annotazioni (ruby), regolazione delle dimensioni e sostituzione dei kanji",

'HTML格式': "Formato HTML",

'HTML格式\_汉字替换': "Formato HTML con sostituzione dei kanji",

'括弧(号)格式': "Formato con parentesi",

'括弧(号)格式\_汉字替换': "Formato con parentesi e sostituzione dei kanji",

'替换后文字列のみ(仅)保留(简单替换)': "Mantieni solo il testo sostituito (sostituzione semplice)"

}

display\_options = list(options.keys())

selected\_display = st.selectbox(

"Seleziona il formato di output (identico a quello definito nel file JSON di sostituzione):",

display\_options,

format\_func=lambda key: options\_italian\_labels[key]

)

format\_type = options[selected\_display]

processed\_text = ""

#=================================================================

# 4) 入力テキストのソースを選択 (手動入力 or ファイルアップロード)

#=================================================================

source\_options = ["手動入力", "ファイルアップロード"]

st.subheader("Fonte del testo di input")

source\_option = st.radio(

"Come vuoi fornire il testo di input?",

source\_options,

format\_func=lambda x: "Inserimento manuale" if x == "手動入力" else "Carica un file"

)

uploaded\_text = ""

if source\_option == "ファイルアップロード":

text\_file = st.file\_uploader("Carica un file di testo (codifica UTF-8)", type=["txt", "csv", "md"])

if text\_file is not None:

uploaded\_text = text\_file.read().decode("utf-8", errors="replace")

st.info("Il file di testo è stato caricato con successo.")

else:

st.warning("Nessun file di testo è stato caricato. Passa all'inserimento manuale o carica un file.")

#=================================================================

# フォーム: 実行ボタン(送信/キャンセル)を配置

# - テキストエリアにエスペラント文を入力してもらう

#=================================================================

with st.form(key='profile\_form'):

if uploaded\_text:

initial\_text = uploaded\_text

else:

initial\_text = st.session\_state.get("text0\_value", "")

text0 = st.text\_area(

"Inserisci qui il testo in esperanto",

height=150,

value=initial\_text

)

st.markdown("""Circondando una parte del testo con il simbolo \*\*%\*\*

(es. `%<testo fino a 50 caratteri>%`), quella parte \*\*non verrà sostituita\*\*

e rimarrà invariata nel risultato finale.""")

st.markdown("""Allo stesso modo, circondando una parte del testo con il simbolo \*\*@\*\*

(es. `@<testo fino a 18 caratteri>@`), quella parte verrà sostituita in modo

\*\*localizzato\*\* (limitato) all'interno di quel frammento.""")

letter\_type = st.radio(

'Scegli la forma di visualizzazione dei caratteri speciali dell\'esperanto nel risultato',

('上付き文字', 'x 形式', '^形式'),

format\_func=lambda x: (

"Accento sulla lettera (ĉ → c + ˆ)" if x == "上付き文字"

else ("Formato con x (ĉ → cx)" if x == "x 形式" else "Formato con ^ (ĉ → c^)")

)

)

submit\_btn = st.form\_submit\_button('Invia')

cancel\_btn = st.form\_submit\_button("Annulla")

if cancel\_btn:

st.warning("Operazione annullata.")

st.stop()

if submit\_btn:

st.session\_state["text0\_value"] = text0

if use\_parallel:

processed\_text = parallel\_process(

text=text0,

num\_processes=num\_processes,

placeholders\_for\_skipping\_replacements=placeholders\_for\_skipping\_replacements,

replacements\_list\_for\_localized\_string=replacements\_list\_for\_localized\_string,

placeholders\_for\_localized\_replacement=placeholders\_for\_localized\_replacement,

replacements\_final\_list=replacements\_final\_list,

replacements\_list\_for\_2char=replacements\_list\_for\_2char,

format\_type=format\_type

)

else:

processed\_text = orchestrate\_comprehensive\_esperanto\_text\_replacement(

text=text0,

placeholders\_for\_skipping\_replacements=placeholders\_for\_skipping\_replacements,

replacements\_list\_for\_localized\_string=replacements\_list\_for\_localized\_string,

placeholders\_for\_localized\_replacement=placeholders\_for\_localized\_replacement,

replacements\_final\_list=replacements\_final\_list,

replacements\_list\_for\_2char=replacements\_list\_for\_2char,

format\_type=format\_type

)

if letter\_type == '上付き文字':

processed\_text = replace\_esperanto\_chars(processed\_text, x\_to\_circumflex)

processed\_text = replace\_esperanto\_chars(processed\_text, hat\_to\_circumflex)

elif letter\_type == '^形式':

processed\_text = replace\_esperanto\_chars(processed\_text, x\_to\_hat)

processed\_text = replace\_esperanto\_chars(processed\_text, circumflex\_to\_hat)

processed\_text = apply\_ruby\_html\_header\_and\_footer(processed\_text, format\_type)

#=================================================================

# =========================================

# フォーム外の処理: 結果表示・ダウンロード

# =========================================

#=================================================================

if processed\_text:

MAX\_PREVIEW\_LINES = 250

lines = processed\_text.splitlines()

if len(lines) > MAX\_PREVIEW\_LINES:

first\_part = lines[:247]

last\_part = lines[-3:]

preview\_text = "\n".join(first\_part) + "\n...\n" + "\n".join(last\_part)

st.warning(

f"Il testo è molto lungo (numero totale di linee: {len(lines)}). "

"Verrà mostrata un'anteprima parziale (prime 247 linee e ultime 3 linee)."

)

else:

preview\_text = processed\_text

if "HTML" in format\_type:

tab1, tab2 = st.tabs(["Anteprima HTML", "Risultato (codice HTML)"])

with tab1:

components.html(preview\_text, height=500, scrolling=True)

with tab2:

st.text\_area("Codice HTML generato :", preview\_text, height=300)

else:

tab3\_list = st.tabs(["Testo risultante"])

with tab3\_list[0]:

st.text\_area("Risultato :", preview\_text, height=300)

download\_data = processed\_text.encode('utf-8')

st.download\_button(

label="Scarica il risultato",

data=download\_data,

file\_name="risultato\_sostituzione.html",

mime="text/html"

)

st.write("---")

st.title("Ligilo-oj(URL-oj)")

st.markdown("""

#### Ligilo-oj de la aplikaĵo en aliaj lingvaj versioj (Esperanto, English, 日本語, 中文, 한국어, Русский, español, italiano, français, Deutsch, العربية, हिन्दी, polski, Tiếng Việt, Bahasa Indonesia; entute 14 lingvoj) ⇓

Esperanta versio

https://esperanto-kanji-converter-and-ruby-annotation-tool-esperanto.streamlit.app/

English version

https://esperanto-kanji-converter-and-ruby-annotation-tool-english.streamlit.app/

日本語版

https://esperanto-kanji-converter-and-ruby-annotation-tool.streamlit.app/

中文版

https://esperanto-hanzi-converter-and-ruby-annotation-tool-chinese-dgw.streamlit.app/

한국어 버전

https://esperanto-kanji-converter-and-ruby-annotation-tool-korean-yrrx.streamlit.app/

Русская версия

https://esperanto-kanji-converter-and-ruby-annotation-tool-russian.streamlit.app/

Versión en español

https://esperanto-kanji-converter-and-ruby-annotation-tool-spanish.streamlit.app/

\*\*Versione italiana\*\*

https://esperanto-kanji-converter-and-ruby-annotation-tool-italian.streamlit.app/

Version française

https://esperanto-kanji-converter-and-ruby-annotation-tool-french.streamlit.app/

Deutsche Version

https://esperanto-kanji-converter-and-ruby-annotation-tool-german.streamlit.app/

إصدار عربي

https://esperanto-kanji-converter-and-ruby-annotation-tool-arabic.streamlit.app/

हिन्दी संस्करण

https://esperanto-kanji-converter-and-ruby-annotation-tool-hindi.streamlit.app/

Polska wersja

https://esperanto-kanji-converter-and-ruby-annotation-tool-polish.streamlit.app/

Phiên bản tiếng Việt

https://esperanto-kanji-converter-and-ruby-annotation-tool-vietnamese.streamlit.app/

Versi Bahasa Indonesia

https://esperanto-kanji-converter-and-ruby-annotation-tool-indonesian.streamlit.app/

#### Uzadaj instrukcioj de la aplikaĵo (README.md en la GitHub-deponejo) ⇓

Esperanta versio

https://github.com/TakafumiYamauchi/Esperanto-Kanji-Converter-and-Ruby-Annotation-Tool-Esperanto

English version

https://github.com/TakafumiYamauchi/Esperanto-Kanji-Converter-and-Ruby-Annotation-Tool-English

日本語版

https://github.com/Takatakatake/Esperanto-Kanji-Converter-and-Ruby-Annotation-Tool-

中文版

https://github.com/Takatakatake/Esperanto-Hanzi-Converter-and-Ruby-Annotation-Tool-Chinese

한국어 버전

https://github.com/Takatakatake/Esperanto-Kanji-Converter-and-Ruby-Annotation-Tool-Korean

Русская версия

https://github.com/TakafumiYamauchi/Esperanto-Kanji-Converter-and-Ruby-Annotation-Tool-Russian

Versión en español

https://github.com/TakafumiYamauchi/Esperanto-Kanji-Converter-and-Ruby-Annotation-Tool-Spanish

\*\*Versione italiana\*\*

https://github.com/TakafumiYamauchi/Esperanto-Kanji-Converter-and-Ruby-Annotation-Tool-Italian

Version française

https://github.com/TakafumiYamauchi/Esperanto-Kanji-Converter-and-Ruby-Annotation-Tool-French

Deutsche Version

https://github.com/TakafumiYamauchi/Esperanto-Kanji-Converter-and-Ruby-Annotation-Tool-German

إصدار عربي

https://github.com/TakafumiYamauchi/Esperanto-Kanji-Converter-and-Ruby-Annotation-Tool-Arabic

हिन्दी संस्करण

https://github.com/TakafumiYamauchi/Esperanto-Kanji-Converter-and-Ruby-Annotation-Tool-Hindi

Polska wersja

https://github.com/TakafumiYamauchi/Esperanto-Kanji-Converter-and-Ruby-Annotation-Tool-Polish

Phiên bản tiếng Việt

https://github.com/TakafumiYamauchi/Esperanto-Kanji-Converter-and-Ruby-Annotation-Tool-Vietnamese

Versi Bahasa Indonesia

https://github.com/TakafumiYamauchi/Esperanto-Kanji-Converter-and-Ruby-Annotation-Tool-Indonesian

""")

## Pagina per generare file JSON per sostituire testi in esperanto con stringhe (kanji).py(2つ目)

#############################

# Pagina per generare file JSON per sostituire testi in esperanto con stringhe (kanji).py (Streamlit特有のpagesフォルダに入れるコード)

#

# 【概要】

# このページでは、「エスペラント文の漢字(イタリア語)置換」で使う

# 最終的な JSON ファイル(合并3個JSONファイル)を生成する機能を提供します。

#

# 【主な流れ】

# 1) CSVファイルを取り込み(エスペラント語根→漢字/イタリア語訳の対応)

# 2) エスペラント単語の語根分解法などを記述したJSONファイルを取り込み

# 3) 必要に応じて並列処理を設定

# 4) ボタン押下で、「置換用の巨大JSON」を最終生成&ダウンロード可能

#

# 【内部処理のポイント】

# - CSVによる「語根→翻訳」情報を独自のsafe\_replace()を使って(placeholder経由で)置換

# - 動詞の活用形(as,is,os...)や接頭辞/接尾辞(2文字語根)などを自動的に展開して

# 文字数に応じた「置換優先順位」を設定

# - カスタムのJSON(語根分解法 / 置換後文字列)も合体してさらに展開

# - 最終的に「全域替换用のリスト」「局部文字替換用のリスト」「二文字词根替换用リスト」の3つを合并

# - ダウンロードボタンで JSONファイルとして取得

#############################

# Pagina per generare file JSON per sostituire testi in esperanto con stringhe (kanji).py (Streamlit特有のpagesフォルダに入れるコード)

# こちらは「置換用JSONファイルを自分で作成したい」場合に利用するツールです。

# main.pyで使う「置換用JSON(合并3個JSONファイル)」を生成するための処理がまとまっています。

# Pagina per generare file JSON per sostituire testi in esperanto con stringhe (kanji).py (Streamlit特有のpagesフォルダに入れるコード)

# こちらは「置換用JSONファイルを自分で作成したい」場合に利用するツールです。

# (main.pyで使う「合并3個JSONファイル」形式の置換用JSONを生成するための処理をまとめています)

import streamlit as st

import pandas as pd

import io

import os

import re

import json

import streamlit as st

from typing import List, Dict, Tuple, Optional

import multiprocessing

from io import StringIO

import streamlit.components.v1 as components

#---------------------------------------------------------------------

# esp\_text\_replacement\_module.py と esp\_replacement\_json\_make\_module.py

# から、必要な関数をインポートして利用します。

# これら2つのモジュールは、エスペラント文字変換・ルビ付与・並列置換など、

# 主に裏方処理(ユーティリティ)を提供する仕組みになっています。

#---------------------------------------------------------------------

from esp\_text\_replacement\_module import (

convert\_to\_circumflex, # エスペラントの文字(ĉ等)形式に変換する関数(cx/c^→ĉなど)

safe\_replace, # (old→placeholder→new)の段階置換を安全に行う関数

import\_placeholders, # プレースホルダ文字列をファイルから読み込む関数

apply\_ruby\_html\_header\_and\_footer # HTMLのルビ表示用ヘッダ/フッタを付加する関数

)

from esp\_replacement\_json\_make\_module import (

convert\_to\_circumflex, # 同じ名前の関数(こちらも字上符に変換)

output\_format, # ルビや括弧形式などの出力フォーマットを生成

import\_placeholders, # プレースホルダを読み込む（同名の関数だが別モジュール）

capitalize\_ruby\_and\_rt, # <ruby>タグ内部の文字を大文字化(冒頭のみ)する関数

process\_chunk\_for\_pre\_replacements, # 並列処理で一括置換する下請け関数

parallel\_build\_pre\_replacements\_dict,# 大量データの置換を並列化して辞書化する関数

remove\_redundant\_ruby\_if\_identical # 重複ルビ(親文字と同一の場合)を取り除く関数

)

#---------------------------------------------------------------------

# 以下は動詞接尾辞や特殊接尾辞などを扱うための変数群です。

#---------------------------------------------------------------------

verb\_suffix\_2l = {

'as':'as', 'is':'is', 'os':'os', 'us':'us','at':'at','it':'it','ot':'ot',

'ad':'ad','iĝ':'iĝ','ig':'ig','ant':'ant','int':'int','ont':'ont'

}

AN=[['dietan', '/diet/an/', '/diet/an'], ['afrikan', '/afrik/an/', '/afrik/an'], ['movadan', '/mov/ad/an/', '/mov/ad/an'], ['akcian', '/akci/an/', '/akci/an'], ['montaran', '/mont/ar/an/', '/mont/ar/an'], ['amerikan', '/amerik/an/', '/amerik/an'], ['regnan', '/regn/an/', '/regn/an'], ['dezertan', '/dezert/an/', '/dezert/an'], ['asocian', '/asoci/an/', '/asoci/an'], ['insulan', '/insul/an/', '/insul/an'], ['azian', '/azi/an/', '/azi/an'], ['ŝtatan', '/ŝtat/an/', '/ŝtat/an'], ['doman', '/dom/an/', '/dom/an'], ['montan', '/mont/an/', '/mont/an'], ['familian', '/famili/an/', '/famili/an'], ['urban', '/urb/an/', '/urb/an'], ['popolan', '/popol/an/', '/popol/an'], ['dekan', '/dekan/', '/dek/an'], ['partian', '/parti/an/', '/parti/an'], ['lokan', '/lok/an/', '/lok/an'], ['ŝipan', '/ŝip/an/', '/ŝip/an'], ['eklezian', '/eklezi/an/', '/eklezi/an'], ['landan', '/land/an/', '/land/an'], ['orientan', '/orient/an/', '/orient/an'], ['lernejan', '/lern/ej/an/', '/lern/ej/an'], ['enlandan', '/en/land/an/', '/en/land/an'], ['kalkan', '/kalkan/', '/kalk/an'], ['estraran', '/estr/ar/an/', '/estr/ar/an'], ['etnan', '/etn/an/', '/etn/an'], ['eŭropan', '/eŭrop/an/', '/eŭrop/an'], ['fazan', '/fazan/', '/faz/an'], ['polican', '/polic/an/', '/polic/an'], ['socian', '/soci/an/', '/soci/an'], ['societan', '/societ/an/', '/societ/an'], ['grupan', '/grup/an/', '/grup/an'], ['ligan', '/lig/an/', '/lig/an'], ['nacian', '/naci/an/', '/naci/an'], ['koran', '/koran/', '/kor/an'], ['religian', '/religi/an/', '/religi/an'], ['kuban', '/kub/an/', '/kub/an'], ['majoran', '/major/an/', '/major/an'], ['nordan', '/nord/an/', '/nord/an'], ['paran', 'paran', '/par/an'], ['parizan', '/pariz/an/', '/pariz/an'], ['parokan', '/parok/an/', '/parok/an'], ['podian', '/podi/an/', '/podi/an'], ['rusian', '/rus/i/an/', '/rus/ian'], ['satan', '/satan/', '/sat/an'], ['sektan', '/sekt/an/', '/sekt/an'], ['senatan', '/senat/an/', '/senat/an'], ['skisman', '/skism/an/', '/skism/an'], ['sudan', 'sudan', '/sud/an'], ['utopian', '/utopi/an/', '/utopi/an'], ['vilaĝan', '/vilaĝ/an/', '/vilaĝ/an'], ['arĝentan', '/arĝent/an/', '/arĝent/an']]

ON=[['duon', '/du/on/', '/du/on'], ['okon', '/ok/on/', '/ok/on'], ['nombron', '/nombr/on/', '/nombr/on'], ['patron', '/patron/', '/patr/on'], ['karbon', '/karbon/', '/karb/on'], ['ciklon', '/ciklon/', '/cikl/on'], ['aldon', '/al/don/', '/ald/on'], ['balon', '/balon/', '/bal/on'], ['baron', '/baron/', '/bar/on'], ['baston', '/baston/', '/bast/on'], ['magneton', '/magnet/on/', '/magnet/on'], ['beton', 'beton', '/bet/on'], ['bombon', '/bombon/', '/bomb/on'], ['breton', 'breton', '/bret/on'], ['burĝon', '/burĝon/', '/burĝ/on'], ['centon', '/cent/on/', '/cent/on'], ['milon', '/mil/on/', '/mil/on'], ['kanton', '/kanton/', '/kant/on'], ['citron', '/citron/', '/citr/on'], ['platon', 'platon', '/plat/on'], ['dekon', '/dek/on/', '/dek/on'], ['kvaron', '/kvar/on/', '/kvar/on'], ['kvinon', '/kvin/on/', '/kvin/on'], ['seson', '/ses/on/', '/ses/on'], ['trion', '/tri/on/', '/tri/on'], ['karton', '/karton/', '/kart/on'], ['foton', '/fot/on/', '/fot/on'], ['peron', '/peron/', '/per/on'], ['elektron', '/elektr/on/', '/elektr/on'], ['drakon', 'drakon', '/drak/on'], ['mondon', '/mon/don/', '/mond/on'], ['pension', '/pension/', '/pensi/on'], ['ordon', '/ordon/', '/ord/on'], ['eskadron', 'eskadron', '/eskadr/on'], ['senton', '/sen/ton/', '/sent/on'], ['eston', 'eston', '/est/on'], ['fanfaron', '/fanfaron/', '/fanfar/on'], ['feston', '/feston/', '/fest/on'], ['flegmon', 'flegmon', '/flegm/on'], ['fronton', '/fronton/', '/front/on'], ['galon', '/galon/', '/gal/on'], ['mason', '/mason/', '/mas/on'], ['helikon', 'helikon', '/helik/on'], ['kanon', '/kanon/', '/kan/on'], ['kapon', '/kapon/', '/kap/on'], ['kokon', '/kokon/', '/kok/on'], ['kolon', '/kolon/', '/kol/on'], ['komision', '/komision/', '/komisi/on'], ['salon', '/salon/', '/sal/on'], ['ponton', '/ponton/', '/pont/on'], ['koton', '/koton/', '/kot/on'], ['kripton', 'kripton', '/kript/on'], ['kupon', '/kupon/', '/kup/on'], ['lakon', 'lakon', '/lak/on'], ['ludon', '/lu/don/', '/lud/on'], ['melon', '/melon/', '/mel/on'], ['menton', '/menton/', '/ment/on'], ['milion', '/milion/', '/mili/on'], ['milionon', '/milion/on/', '/milion/on'], ['naŭon', '/naŭ/on/', '/naŭ/on'], ['violon', '/violon/', '/viol/on'], ['trombon', '/trombon/', '/tromb/on'], ['senson', '/sen/son/', '/sens/on'], ['sepon', '/sep/on/', '/sep/on'], ['skadron', 'skadron', '/skadr/on'], ['stadion', '/stadion/', '/stadi/on'], ['tetraon', 'tetraon', '/tetra/on'], ['timon', '/timon/', '/tim/on'], ['valon', 'valon', '/val/on']]

allowed\_values = {-1, "-1", "ー１", "ー1", "-１", "－１", "－1"}

suffix\_2char\_roots=['ad', 'ag', 'am', 'ar', 'as', 'at', 'av', 'di', 'ec', 'eg', 'ej', 'em', 'er', 'et', 'id', 'ig', 'il', 'in', 'ir', 'is', 'it', 'lu', 'nj', 'op', 'or', 'os', 'ot', 'ov', 'pi', 'te', 'uj', 'ul', 'um', 'us', 'uz','ĝu','aĵ','iĝ','aĉ','aĝ','ŝu','eĥ']

prefix\_2char\_roots=['al', 'am', 'av', 'bo', 'di', 'du', 'ek', 'el', 'en', 'fi', 'ge', 'ir', 'lu', 'ne', 'ok', 'or', 'ov', 'pi', 're', 'te', 'uz','ĝu','aĉ','aĝ','ŝu','eĥ']

standalone\_2char\_roots=['al', 'ci', 'da', 'de', 'di', 'do', 'du', 'el', 'en', 'fi', 'ha', 'he', 'ho', 'ia', 'ie', 'io', 'iu', 'ja', 'je', 'ju','ke', 'la', 'li', 'mi', 'ne', 'ni', 'nu', 'ok', 'ol', 'po', 'se', 'si', 've', 'vi','ŭa','aŭ','ĉe','ĝi','ŝi','ĉu']

#---------------------------------------------------------------------

# placeholders / 字体幅 情報などを読み込み

#---------------------------------------------------------------------

imported\_placeholders\_for\_global\_replacement = import\_placeholders(

'./Appの运行に使用する各类文件/占位符(placeholders)\_$20987$-$499999$\_全域替换用.txt'

)

imported\_placeholders\_for\_2char\_replacement = import\_placeholders(

'./Appの运行に使用する各类文件/占位符(placeholders)\_$13246$-$19834$\_二文字词根替换用.txt'

)

imported\_placeholders\_for\_local\_replacement = import\_placeholders(

'./Appの运行に使用する各类文件/占位符(placeholders)\_@20374@-@97648@\_局部文字列替换用.txt'

)

with open("./Appの运行に使用する各类文件/Unicode\_BMP全范围文字幅(宽)\_Arial16.json", "r", encoding="utf-8") as fp:

char\_widths\_dict = json.load(fp)

#=====================================================================

# 1) ページ設定 & タイトル

#=====================================================================

st.set\_page\_config(

page\_title="Strumento per generare file JSON per la sostituzione di testo (caratteri cinesi) in Esperanto",

layout="wide"

)

st.title("Generare un file JSON per la sostituzione (caratteri cinesi) in testo Esperanto.")

st.write("---")

#=====================================================================

# 2) 概要説明 (使い方)

#=====================================================================

with st.expander("Apri le istruzioni su come utilizzare", expanded=True):

st.markdown("""

#### Prima di iniziare

In questa pagina è possibile generare e scaricare

il file JSON (che potrebbe raggiungere anche 50MB)

da utilizzare nella sostituzione di testo in Esperanto

nella pagina principale.

\*\*Procedura di utilizzo\*\*:

1. Caricare (oppure utilizzare un default) il \*\*file CSV\*\*

(tabella di corrispondenze tra radici dell'Esperanto e la traduzione in italiano o i caratteri cinesi).

2. Caricare (oppure utilizzare un default) il \*\*file JSON\*\*

(regole di scomposizione delle radici e/o impostazioni personalizzate sul testo sostituito).

3. Fare clic su “Crea il file JSON per la sostituzione” per

\*\*scaricare\*\* il file JSON per la sostituzione.

Sono disponibili file di esempio nella sezione sottostante,

che possono essere utili come riferimento per la formattazione

o per eventuali modifiche personalizzate.

""")

#=====================================================================

# 3) サンプルファイル一覧 (折りたたみ)

#=====================================================================

with st.expander("Elenco di file di esempio (download)"):

st.write("#### Elenco di file di esempio")

st.markdown("""

\*\*CSV di esempio n.1 (Tabella di corrispondenza tra radici esperanto e traduzioni italiane + annotazioni ruby)\*\*

File CSV con corrispondenza riga per riga tra radici dell'Esperanto e traduzione italiana (eventualmente con annotazioni ruby).

Creando un CSV in questo formato e caricandolo, il tool genera il file JSON per la sostituzione.

""")

file\_path0 = './Appの运行に使用する各类文件/Elenco di radici esperanto con traduzioni italiane e annotazioni ruby.csv'

with open(file\_path0, "rb") as file:

st.download\_button(

label="Scarica CSV di esempio n.1 (Radici Esperanto – Traduzione Italiana + annotazioni ruby)",

data=file,

file\_name="Elenco\_di\_radici\_esperanto\_con\_traduzioni\_italiane\_e\_annotazioni\_ruby.csv",

mime="text/csv"

)

st.markdown("""

\*\*CSV di esempio n.2 (Tabella di corrispondenza tra radici esperanto e caratteri cinesi, proposta di “Mingeo”)\*\*

Qui, ogni riga del CSV associa una radice dell'Esperanto a un carattere (o più caratteri) cinesi.

""")

file\_path0 = './Appの运行に使用する各类文件/Mingeo先生版 世界语词根-汉字对应列表.csv'

with open(file\_path0, "rb") as file:

st.download\_button(

label="Scarica CSV di esempio n.2 (Radici Esperanto – Caratteri cinesi di Mingeo)",

data=file,

file\_name="RadiciEsperanto\_CaratteriCinesi\_propostaMingeo.csv",

mime="text/csv"

)

st.markdown("""

\*\*CSV di esempio n.3 (Tabella di corrispondenza tra radici esperanto e caratteri cinesi)\*\*

Ulteriore file CSV con corrispondenza radici-caratteri cinesi.

""")

file\_path0 = './Appの运行に使用する各类文件/世界语词根-汉字对应列表.csv'

with open(file\_path0, "rb") as file:

st.download\_button(

label="Scarica CSV di esempio n.3 (Radici Esperanto – Caratteri cinesi)",

data=file,

file\_name="RadiciEsperanto\_CaratteriCinesi.csv",

mime="text/csv"

)

st.markdown("""

\*\*JSON di esempio n.1 (Impostazioni utente per la scomposizione delle radici esperanto)\*\*

\*\*Utilizzo\*\*: definisce come scomporre in radici le parole esperanto, l’ordine di sostituzione,

e quando applicare i suffissi (per es. forme verbali).

(Esempio: `["am", "dflt", ["verbo\_s1"]]`)

""")

json\_file\_path = './Appの运行に使用する各类文件/世界语单词词根分解方法の使用者自定义设置.json'

with open(json\_file\_path, "rb") as file\_json:

st.download\_button(

label="Scarica JSON di esempio n.1 (Scomposizione delle radici esperanto)",

data=file\_json,

file\_name="ImpostazioniUtente\_ScomposizioneRadiciEsperanto.json",

mime="application/json"

)

st.markdown("""

\*\*JSON di esempio n.2 (Impostazioni utente per il testo sostituito)\*\*

\*\*Utilizzo\*\*: per assegnare caratteri cinesi o formati speciali a specifiche parole

in aggiunta alla definizione di scomposizione radici del JSON di cui sopra.

(In genere basta modificare il file CSV e/o il JSON di scomposizione radici, e questo file non è strettamente necessario.)

""")

json\_file\_path2 = './Appの运行に使用する各类文件/替换后文字列(汉字)の使用者自定义设置(基本上完全不推荐).json'

with open(json\_file\_path2, "rb") as file\_json:

st.download\_button(

label="Scarica JSON di esempio n.2 (Impostazioni utente per il testo sostituito)",

data=file\_json,

file\_name="ImpostazioniUtente\_TestoSostituito\_CaratteriCinesi.json",

mime="application/json"

)

st.markdown("""

\*\*Excel di esempio n.1 (4736 radici esperanto con traduzioni/annotazioni in 14 lingue)\*\*

(Giapponese, Cinese, Coreano, Inglese, Russo, Spagnolo, Italiano, Francese, Tedesco,

Arabo, Hindi, Polacco, Vietnamita, Indonesiano)

""")

with open('./Appの运行に使用する各类文件/Kreado de rubenaj komentoj en 14 lingvoj (日本語, 中文, 한국어, English, Русский, español, italiano, français, Deutsch, العربية, हिन्दी, polski, Tiếng Việt, Bahasa Indonesia) respondaj al listo de 4736 Esperant.xlsx', "rb") as file:

st.download\_button(

label="Scarica Excel di esempio n.1 (Radici esperanto con traduzioni in 14 lingue)",

data=file,

file\_name="RadiciEsperanto\_conAnnotazioni\_in14Lingue.xlsx",

mime="application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"

)

st.markdown("""

\*\*Excel di esempio n.2 (Radici esperanto – traduzioni in giapponese, con livello di apprendimento)\*\*

\*\*Utilizzo\*\*: utile per personalizzare le radici da sostituire con una traduzione o un’annotazione personalizzata.

Contiene anche un "livello di apprendimento" basato su un dizionario base Esper/Jap.

""")

with open('./Appの运行に使用する各类文件/エスペラント語根-日本語訳ルビ対応リスト(習得レベル付き).xlsx', "rb") as file:

st.download\_button(

label="Scarica Excel di esempio n.2 (Radici esperanto – trad. giapponese con livello di apprendimento)",

data=file,

file\_name="RadiciEsperanto\_Giapponese\_conLivelloApprendimento.xlsx",

mime="application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"

)

st.write("---")

#=====================================================================

# 出力形式(ルビなど)の選択肢 (GUIに表示されるキーをイタリア語に)

#=====================================================================

options = {

"HTML (con dimensione personalizzata dei caratteri Ruby)": "HTML格式\_Ruby文字\_大小调整",

"HTML (con dimensione Ruby personalizzata e sostituzione di caratteri cinesi)": "HTML格式\_Ruby文字\_大小调整\_汉字替换",

"HTML (base)": "HTML格式",

"HTML (con sostituzione di caratteri cinesi)": "HTML格式\_汉字替换",

"Formato con parentesi": "括弧(号)格式",

"Formato con parentesi (con sostituzione di caratteri cinesi)": "括弧(号)格式\_汉字替换",

"Sostituzione semplice": "替换后文字列のみ(仅)保留(简单替换)"

}

display\_options = list(options.keys())

selected\_display = st.selectbox('Seleziona il formato di output:', display\_options)

format\_type = options[selected\_display]

# 簡易サンプルの表示 (出力例)

main\_text\_list = ['Esperant','lingv', 'pac', 'amik', 'ec']

ruby\_content\_list = ['世界语', '语言', '和平', '友', '性质']

formatted\_text = ''

for i, item in enumerate(main\_text\_list):

formatted\_text += output\_format(item, ruby\_content\_list[i], format\_type, char\_widths\_dict)

st.write("---")

st.markdown("\*\*Testo formattato di esempio (anteprima):\*\*")

components.html(apply\_ruby\_html\_header\_and\_footer(formatted\_text, format\_type), height=40, scrolling=False)

st.write("---")

#=====================================================================

# ステップ１: CSVファイルを準備 (アップロード or デフォルト)

#=====================================================================

st.header("Fase 1: Preparare il file CSV")

st.markdown(

"""

### Selezionare il \*\*file CSV\*\* contenente le radici dell'Esperanto e le relative traduzioni (caratteri cinesi o italiano).

---

"""

)

csv\_choice = st.radio("Come desideri procedere con il file CSV?", ("Carica un file CSV", "Usa il file di default"))

csv\_path\_default = "./Appの运行に使用する各类文件/Elenco di radici esperanto con traduzioni italiane e annotazioni ruby.csv"

CSV\_data\_imported = None

if csv\_choice == "Carica un file CSV":

st.write("Carica un file CSV (consigliato UTF-8).")

uploaded\_file = st.file\_uploader("Seleziona il file CSV", type=['csv'])

if uploaded\_file is not None:

file\_contents = uploaded\_file.read().decode("utf-8")

converted\_text = convert\_to\_circumflex(file\_contents)

csv\_buffer = StringIO(converted\_text)

CSV\_data\_imported = pd.read\_csv(csv\_buffer, encoding="utf-8", usecols=[0, 1])

st.success("File CSV caricato correttamente.")

else:

st.warning("Nessun file CSV caricato.")

st.stop()

elif csv\_choice == "Usa il file di default":

try:

with open(csv\_path\_default, 'r', encoding="utf-8") as file:

text = file.read()

converted\_text = convert\_to\_circumflex(text)

csv\_buffer = StringIO(converted\_text)

CSV\_data\_imported = pd.read\_csv(csv\_buffer, encoding="utf-8", usecols=[0, 1])

st.info("Verrà utilizzato il file CSV di default.")

except FileNotFoundError:

st.error("Impossibile trovare il file CSV di default. Elaborazione interrotta.")

st.stop()

st.write("Caricamento del file CSV completato. Procedere oltre.")

st.write("---")

#=====================================================================

# ステップ2: JSONファイル(語根分解法など)を準備

#=====================================================================

st.header("Fase 2: Preparare il/i file JSON (regole di scomposizione, ecc.)")

st.markdown("""

È possibile caricare oppure utilizzare i file JSON di default che

contengono \*\*le regole di scomposizione delle radici esperanto\*\* e/o

eventuali \*\*testi di sostituzione personalizzati\*\*.

Si può scaricare un file JSON di esempio e modificarlo per adattarlo alle proprie esigenze.

""")

json\_choice = st.radio("1. Come procedere con il file JSON per definire la scomposizione delle radici esperanto?", ("Carica un file JSON", "Usa il file di default"))

json\_path\_default = "./Appの运行に使用する各类文件/世界语单词词根分解方法の使用者自定义设置.json"

custom\_stemming\_setting\_list = None

if json\_choice == "Carica un file JSON":

uploaded\_json = st.file\_uploader("Carica il file JSON", type=['json'])

if uploaded\_json is not None:

custom\_stemming\_setting\_list = json.load(uploaded\_json)

st.success("File JSON caricato correttamente.")

else:

st.warning("Nessun file JSON caricato.")

st.stop()

elif json\_choice == "Usa il file di default":

try:

with open(json\_path\_default, "r", encoding="utf-8") as g:

custom\_stemming\_setting\_list = json.load(g)

st.info("Verrà utilizzato il file JSON di default.")

except FileNotFoundError:

st.error("Impossibile trovare il file JSON di default.")

st.stop()

json\_choice2 = st.radio("2. Come procedere con il file JSON per definire il testo sostituito?", ("Carica un file JSON", "Usa il file di default"))

json\_path\_default2 = "./Appの运行に使用する各类文件/替换后文字列(汉字)の使用者自定义设置(基本上完全不推荐).json"

user\_replacement\_item\_setting\_list = None

if json\_choice2 == "Carica un file JSON":

uploaded\_json = st.file\_uploader("Carica il secondo file JSON", type=['json'])

if uploaded\_json is not None:

user\_replacement\_item\_setting\_list = json.load(uploaded\_json)

st.success("File JSON caricato correttamente.")

else:

st.warning("Nessun file JSON caricato.")

st.stop()

elif json\_choice2 == "Usa il file di default":

try:

with open(json\_path\_default2, "r", encoding="utf-8") as g:

user\_replacement\_item\_setting\_list = json.load(g)

st.info("Verrà utilizzato il secondo file JSON di default.")

except FileNotFoundError:

st.error("Impossibile trovare il secondo file JSON di default.")

st.stop()

st.write("---")

#=====================================================================

# ステップ3: 高度な設定 (並列処理)

#=====================================================================

st.header("Fase 3: Impostazioni avanzate (elaborazione in parallelo)")

with st.expander("Apri le impostazioni relative all'elaborazione parallela"):

st.write("""

Qui è possibile specificare il numero di processi da utilizzare

per l’elaborazione in parallelo durante la creazione del file JSON.

Se il numero di radici o il testo da sostituire è molto ampio,

si potrebbe ottenere un aumento di velocità sfruttando più core CPU.

""")

use\_parallel = st.checkbox("Usa elaborazione parallela", value=False)

num\_processes = st.number\_input("Numero di processi in parallelo", min\_value=2, max\_value=6, value=5, step=1)

st.write("### Creare il file JSON finale per la sostituzione (pulsante)")

if st.button("Crea il file JSON per la sostituzione"):

with st.spinner("Generazione del file JSON per la sostituzione in corso... Attendere qualche istante."):

#-------------------------------------------------------------

# (1) 大規模データ 'E\_stem\_with\_Part\_Of\_Speech\_list' を開く

#-------------------------------------------------------------

with open("./Appの运行に使用する各类文件/PEJVO(世界语全部单词列表)'全部'について、词尾(a,i,u,e,o,n等)をcutし、comma(,)で隔てて词性と併せて记录した列表(E\_stem\_with\_Part\_Of\_Speech\_list).json", "r", encoding="utf-8") as g:

E\_stem\_with\_Part\_Of\_Speech\_list = json.load(g)

#-------------------------------------------------------------

# (2) エスペラントの全語根(約11137個)を辞書型に格納

#-------------------------------------------------------------

temporary\_replacements\_dict = {}

with open("./Appの运行に使用する各类文件/世界语全部词根\_约11137个\_202501.txt", 'r', encoding='utf-8') as file:

E\_roots = file.readlines()

for E\_root in E\_roots:

E\_root = E\_root.strip()

if not E\_root.isdigit():

temporary\_replacements\_dict[E\_root] = [E\_root, len(E\_root)]

#-------------------------------------------------------------

# (3) CSV\_data\_imported を用いて

# 「エスペラント語根 → (漢字 or イタリア語訳ルビ等)」を反映させる

#-------------------------------------------------------------

for \_, (E\_root, hanzi\_or\_meaning) in CSV\_data\_imported.iterrows():

if pd.notna(E\_root) and pd.notna(hanzi\_or\_meaning) \

and '#' not in E\_root and (E\_root != '') and (hanzi\_or\_meaning != ''):

temporary\_replacements\_dict[E\_root] = [

output\_format(E\_root, hanzi\_or\_meaning, format\_type, char\_widths\_dict),

len(E\_root)

]

#-------------------------------------------------------------

# (4) ソート（長い単語ほど優先的に置換させる）

#-------------------------------------------------------------

temporary\_replacements\_list\_1 = []

for old, new in temporary\_replacements\_dict.items():

temporary\_replacements\_list\_1.append((old, new[0], new[1]))

temporary\_replacements\_list\_2 = sorted(temporary\_replacements\_list\_1, key=lambda x: x[2], reverse=True)

#-------------------------------------------------------------

# (5) placeholder リストを作成 (collision回避)

#-------------------------------------------------------------

temporary\_replacements\_list\_final = []

for kk in range(len(temporary\_replacements\_list\_2)):

temporary\_replacements\_list\_final.append([

temporary\_replacements\_list\_2[kk][0],

temporary\_replacements\_list\_2[kk][1],

imported\_placeholders\_for\_global\_replacement[kk]

])

#-------------------------------------------------------------

# (6) parallel\_build\_pre\_replacements\_dict の実行 (並列 or 単一)

#-------------------------------------------------------------

if use\_parallel:

pre\_replacements\_dict\_1 = parallel\_build\_pre\_replacements\_dict(

E\_stem\_with\_Part\_Of\_Speech\_list,

temporary\_replacements\_list\_final,

num\_processes

)

else:

progress\_bar = st.progress(0)

progress\_text = st.empty()

total\_items = len(E\_stem\_with\_Part\_Of\_Speech\_list)

pre\_replacements\_dict\_1 = {}

for i, j in enumerate(E\_stem\_with\_Part\_Of\_Speech\_list):

if len(j) == 2:

if len(j[0]) >= 2:

if j[0] in pre\_replacements\_dict\_1:

if j[1] not in pre\_replacements\_dict\_1[j[0]][1]:

pre\_replacements\_dict\_1[j[0]] = [

pre\_replacements\_dict\_1[j[0]][0],

pre\_replacements\_dict\_1[j[0]][1] + ',' + j[1]

]

else:

pre\_replacements\_dict\_1[j[0]] = [

safe\_replace(j[0], temporary\_replacements\_list\_final),

j[1]

]

if i % 1000 == 0:

current\_count = i + 1

progress\_value = int(current\_count / total\_items \* 100)

progress\_bar.progress(progress\_value)

progress\_text.write(f"{current\_count}/{total\_items} elementi in elaborazione...")

progress\_bar.progress(100)

progress\_text.write("L'elaborazione più lunga è completata al 100%. (Occorrono ancora 3~4 secondi.)")

# 例: 処理上、除外したいキーをここでpopする (domen, teren, posten等)

keys\_to\_remove = ['domen', 'teren','posten']

for key in keys\_to\_remove:

pre\_replacements\_dict\_1.pop(key, None)

#-------------------------------------------------------------

# (7) pre\_replacements\_dict\_1 をさらに加工(優先順位調整等)していく

# → pre\_replacements\_dict\_2 にまとめる

# 「置換しない単語の場合は優先順位を下げる」「ルビの一部を除去/再設定」など

#-------------------------------------------------------------

pre\_replacements\_dict\_2 = {}

for i,j in pre\_replacements\_dict\_1.items():

# j[0] = safe\_replace後の文字列, j[1] = 品詞

# i==j[0] の場合は「実質置換されなかった単語(変化なし)」とみなし、優先順位を低めに設定

if i==j[0]:

# 文字列末尾等に含まれる'/','</rt></ruby>'などを一部加工している

pre\_replacements\_dict\_2[i.replace('/', '')] = [

j[0].replace("</rt></ruby>","%%%").replace('/', '').replace("%%%","</rt></ruby>"),

j[1],

len(i.replace('/', ''))\*10000 - 3000

]

else:

# 置換後文字列は j[0] だが、一部'/'を取り除いて処理し、優先順位を(文字数\*10000)に設定

pre\_replacements\_dict\_2[i.replace('/', '')] = [

j[0].replace("</rt></ruby>","%%%").replace('/', '').replace("%%%","</rt></ruby>"),

j[1],

len(i.replace('/', ''))\*10000

]

#-------------------------------------------------------------

# (8) ここから先は、AN, ON, 動詞語尾などの接頭辞/接尾辞を用いた

# 優先順位調整を大量に行う。

#

# 具体的には、辞書 pre\_replacements\_dict\_2 をさらに書き換えたり、

# 新しいキー(=語尾を付けた形など)を追加して、より精度の高い置換を行えるようにしている。

#

# コード量は多いですが、やっていることは

# 「(語根 + an)を名詞/形容詞とみなすか、それとも接尾辞an(員)とみなすか」

# 「(語根 + as)で動詞現在形にする場合の優先順位をどうするか」

# などの細かいルール付けです。

#-------------------------------------------------------------

#------------------------------------------

# verb\_suffix\_2l\_2 という辞書を作る:

# verb\_suffix\_2l の各キー(例:'as')とその置換結果をsafe\_replace()で更新

# こうすることで "(語根)+(動詞接尾辞)" に対してルビなどを入れ込めるようにします。

#------------------------------------------

verb\_suffix\_2l\_2={}

for original\_verb\_suffix,replaced\_verb\_suffix in verb\_suffix\_2l.items():

# 例: 'as'→'as' のままのことが多いが、safe\_replaceで更に別ルビを当てはめる可能性あり

verb\_suffix\_2l\_2[original\_verb\_suffix] = safe\_replace(replaced\_verb\_suffix, temporary\_replacements\_list\_final)

# 一番の工夫ポイント(以下、コメントはコード内にある通り):

# 置換の優先順位をどう定めるかで、置換の精度が大きく変わる。

# 文字数の多い単語を先に置換する、動詞の場合は活用語尾を付けた形を優先度高くするetc.

#

# pre\_replacements\_dict\_1→pre\_replacements\_dict\_2→pre\_replacements\_dict\_3

# という流れで段階的に書き換え、最終的に "replacements\_final\_list" へまとめる方針。

unchangeable\_after\_creation\_list=[]

AN\_replacement = safe\_replace('an', temporary\_replacements\_list\_final)

AN\_treatment=[]

pre\_replacements\_dict\_3={}

# 辞書をコピー (2回以上繰り返す時に改変が及ばないように)

pre\_replacements\_dict\_2\_copy = pre\_replacements\_dict\_2.copy()

# (8-1) 例えば "xxxan" という語があり、それが名詞品詞("名词")なのに

# 中で "an"がルビとして置換されている...等、誤置換を防ぐための調整。

for i,j in pre\_replacements\_dict\_2\_copy.items(): # j[0]:置換後文字列, j[1]:品詞, j[2]:優先順位

if i.endswith('an') and (AN\_replacement in j[0]) and ("名词" in j[1]) and (i[:-2] in pre\_replacements\_dict\_2\_copy):

# 形容詞語尾anと接尾辞anが衝突する場合などに対応

AN\_treatment.append([i,j[0]])

pre\_replacements\_dict\_2.pop(i, None)

# そこへさらに "i+"o,"i+"a,"i+"e などの派生形を追加する処理

for k in ["o","a","e"]:

if not i+k in pre\_replacements\_dict\_2\_copy:

pre\_replacements\_dict\_3[i+k]=[j[0]+k, j[2]+len(k)\*10000-2000]

elif (j[1] == "名词") and (len(i)<=6) and not(j[2] in [60000,50000,40000,30000,20000]):

# 名詞で6文字以下、かつ特定優先順位でないものを調整

for k in ["o"]:

if not i+k in pre\_replacements\_dict\_2\_copy:

pre\_replacements\_dict\_3[i+k]=[j[0]+k,j[2]+len(k)\*10000-2000]

pre\_replacements\_dict\_2.pop(i, None)

# (8-2) 2文字語根の特別処理(例えば "am" "ar" など)

# 動詞の接尾辞(ag, ig等)を足した形の置換を優先させたいが、名詞や形容詞の場合はどうするか等

for i,j in pre\_replacements\_dict\_2.items():

# j[2]が20000の場合は2文字語根の優先度っぽい

if j[2]==20000:

# 名詞の場合

if "名词" in j[1]:

for k in ["o","on",'oj']:

if not i+k in pre\_replacements\_dict\_2:

pre\_replacements\_dict\_3[' '+i+k] = [' '+j[0]+k, j[2] + (len(k)+1)\*10000 - 5000]

# 形容詞の場合

if "形容词" in j[1]:

for k in ["a","aj",'an']:

if not i+k in pre\_replacements\_dict\_2:

pre\_replacements\_dict\_3[' '+i+k] = [' '+j[0]+k, j[2] + (len(k)+1)\*10000 - 5000]

else:

pre\_replacements\_dict\_3[i+k] = [j[0]+k, j[2] + len(k)\*10000 - 5000]

unchangeable\_after\_creation\_list.append(i+k)

# 副詞の場合

if "副词" in j[1]:

for k in ["e"]:

if not i+k in pre\_replacements\_dict\_2:

pre\_replacements\_dict\_3[' '+i+k] = [' '+j[0]+k, j[2] + (len(k)+1)\*10000 - 5000]

else:

pre\_replacements\_dict\_3[' '+i+k] = [' '+j[0]+k, j[2] + (len(k)+1)\*10000 - 5000]

# 動詞の場合(ここが複雑; 動詞活用語尾(as,is,os,etc)と組み合わせる)

if "动词" in j[1]:

for k1,k2 in verb\_suffix\_2l\_2.items():

if not i+k1 in pre\_replacements\_dict\_2:

pre\_replacements\_dict\_3[i+k1] = [j[0]+k2, j[2] + len(k1)\*10000 - 3000]

elif j[0]+k2 != pre\_replacements\_dict\_2[i+k1][0]:

pre\_replacements\_dict\_3[i+k1] = [j[0]+k2, j[2] + len(k1)\*10000 - 3000]

unchangeable\_after\_creation\_list.append(i+k1)

for k in ["u ","i ","u","i"]:

if not i+k in pre\_replacements\_dict\_2:

pre\_replacements\_dict\_3[i+k] = [j[0]+k, j[2] + len(k)\*10000 - 3000]

continue

else:

if not i in unchangeable\_after\_creation\_list:# unchangeable\_after\_creation\_list に含まれる場合は除外。(上記で新しく定めた語根分解が更新されてしまわないようにするため。)

pre\_replacements\_dict\_3[i]=[j[0],j[2]]# 品詞情報はここで用いるためにあった。以後は不要なので省いていく。

if j[2]==60000 or j[2]==50000 or j[2]==40000 or j[2]==30000:# 文字数が比較的少なく(<=5)、実際に置換するエスペラント語根(文字数×10000)のみを対象とする

if "名词" in j[1]:# 名词については形容词、副词と違い、置換しないものにもoをつける。

for k in ["o","on",'oj']:

if not i+k in pre\_replacements\_dict\_2:

pre\_replacements\_dict\_3[i+k]=[j[0]+k,j[2]+len(k)\*10000-3000]# 既存でないものは優先順位を大きく下げる→普通の品詞接尾辞が既存でないという言い方はおかしい気がしてきた。(20240612)

elif j[0]+k != pre\_replacements\_dict\_2[i+k][0]:

pre\_replacements\_dict\_3[i+k]=[j[0]+k,j[2]+len(k)\*10000-3000]# 新しく作った方の語根分解を優先する

unchangeable\_after\_creation\_list.append(i+k)

# on系[['nombron', '<ruby>nombr<rt class="ruby-X\_X\_X">数</rt></ruby><ruby>on<rt class="ruby-M\_M\_M">分数</rt></ruby>', '<ruby>nombr<rt class="ruby-X\_X\_X">数</rt></ruby>on'], ['patron', '<ruby>patron<rt class="ruby-X\_X\_X">後援者</rt></ruby>', '<ruby>patr<rt class="ruby-X\_X\_X">父</rt></ruby>on'], ['karbon', '<ruby>karbon<rt class="ruby-L\_L\_L">[化]炭素</rt></ruby>', '<ruby>karb<rt class="ruby-X\_X\_X">炭</rt></ruby>on'], ['ciklon', '<ruby>ciklon<rt class="ruby-X\_X\_X">低気圧</rt></ruby>', '<ruby>cikl<rt class="ruby-X\_X\_X">周期</rt></ruby>on'], ['aldon', '<ruby>al<rt class="ruby-S\_S\_S">~の方へ</rt></ruby><ruby>don<rt class="ruby-M\_M\_M">与える</rt></ruby>', '<ruby>ald<rt class="ruby-M\_M\_M">アルト</rt></ruby>on'], ['balon', '<ruby>balon<rt class="ruby-X\_X\_X">気球</rt></ruby>', '<ruby>bal<rt class="ruby-M\_M\_M">舞踏会</rt></ruby>on'], ['baron', '<ruby>baron<rt class="ruby-X\_X\_X">男爵</rt></ruby>', '<ruby>bar<rt class="ruby-L\_L\_L">障害</rt></ruby>on'], ['baston', '<ruby>baston<rt class="ruby-X\_X\_X">棒</rt></ruby>', '<ruby>bast<rt class="ruby-M\_M\_M">[植]じん皮</rt></ruby>on'], ['magneton', '<ruby>magnet<rt class="ruby-L\_L\_L">[理]磁石</rt></ruby><ruby>on<rt class="ruby-M\_M\_M">分数</rt></ruby>', '<ruby>magnet<rt class="ruby-L\_L\_L">[理]磁石</rt></ruby>on'], ['beton', 'beton', '<ruby>bet<rt class="ruby-M\_M\_M">ビート</rt></ruby>on'], ['bombon', '<ruby>bombon<rt class="ruby-L\_L\_L">キャンデー</rt></ruby>', '<ruby>bomb<rt class="ruby-X\_X\_X">爆弾</rt></ruby>on'], ['breton', 'breton', '<ruby>bret<rt class="ruby-X\_X\_X">棚</rt></ruby>on'], ['burgxon', '<ruby>burgxon<rt class="ruby-X\_X\_X">芽</rt></ruby>', '<ruby>burgx<rt class="ruby-M\_M\_M">ブルジョワ</rt></ruby>on'], ['centon', '<ruby>cent<rt class="ruby-X\_X\_X">百</rt></ruby><ruby>on<rt class="ruby-M\_M\_M">分数</rt></ruby>', '<ruby>cent<rt class="ruby-X\_X\_X">百</rt></ruby>on'], ['milon', '<ruby>mil<rt class="ruby-X\_X\_X">千</rt></ruby><ruby>on<rt class="ruby-M\_M\_M">分数</rt></ruby>', '<ruby>mil<rt class="ruby-X\_X\_X">千</rt></ruby>on'], ['kanton', '<ruby>kanton<rt class="ruby-M\_M\_M">(フランスの)郡</rt></ruby>', '<ruby>kant<rt class="ruby-M\_M\_M">(を)歌う</rt></ruby>on'], ['citron', '<ruby>citron<rt class="ruby-M\_M\_M">[果]シトロン</rt></ruby>', '<ruby>citr<rt class="ruby-M\_M\_M">[楽]チター</rt></ruby>on'], ['platon', 'platon', '<ruby>plat<rt class="ruby-L\_L\_L">平たい</rt></ruby>on'], ['dekon', '<ruby>dek<rt class="ruby-X\_X\_X">十</rt></ruby><ruby>on<rt class="ruby-M\_M\_M">分数</rt></ruby>', '<ruby>dek<rt class="ruby-X\_X\_X">十</rt></ruby>on'], ['kvaron', '<ruby>kvar<rt class="ruby-X\_X\_X">四</rt></ruby><ruby>on<rt class="ruby-M\_M\_M">分数</rt></ruby>', '<ruby>kvar<rt class="ruby-X\_X\_X">四</rt></ruby>on'], ['kvinon', '<ruby>kvin<rt class="ruby-X\_X\_X">五</rt></ruby><ruby>on<rt class="ruby-M\_M\_M">分数</rt></ruby>', '<ruby>kvin<rt class="ruby-X\_X\_X">五</rt></ruby>on'], ['seson', '<ruby>ses<rt class="ruby-X\_X\_X">六</rt></ruby><ruby>on<rt class="ruby-M\_M\_M">分数</rt></ruby>', '<ruby>ses<rt class="ruby-X\_X\_X">六</rt></ruby>on'], ['trion', '<ruby>tri<rt class="ruby-X\_X\_X">三</rt></ruby><ruby>on<rt class="ruby-M\_M\_M">分数</rt></ruby>', '<ruby>tri<rt class="ruby-X\_X\_X">三</rt></ruby>on'], ['karton', '<ruby>karton<rt class="ruby-X\_X\_X">厚紙</rt></ruby>', '<ruby>kart<rt class="ruby-L\_L\_L">カード</rt></ruby>on'], ['foton', '<ruby>fot<rt class="ruby-S\_S\_S">写真を撮る</rt></ruby><ruby>on<rt class="ruby-M\_M\_M">分数</rt></ruby>', '<ruby>fot<rt class="ruby-S\_S\_S">写真を撮る</rt></ruby>on'], ['peron', '<ruby>peron<rt class="ruby-X\_X\_X">階段</rt></ruby>', '<ruby>per<rt class="ruby-M\_M\_M">よって</rt></ruby>on'], ['elektron', '<ruby>elektr<rt class="ruby-X\_X\_X">電気</rt></ruby><ruby>on<rt class="ruby-M\_M\_M">分数</rt></ruby>', '<ruby>elektr<rt class="ruby-X\_X\_X">電気</rt></ruby>on'], ['drakon', 'drakon', '<ruby>drak<rt class="ruby-X\_X\_X">竜</rt></ruby>on'], ['mondon', '<ruby>mon<rt class="ruby-L\_L\_L">金銭</rt></ruby><ruby>don<rt class="ruby-M\_M\_M">与える</rt></ruby>', '<ruby>mond<rt class="ruby-X\_X\_X">世界</rt></ruby>on'], ['pension', '<ruby>pension<rt class="ruby-X\_X\_X">下宿屋</rt></ruby>', '<ruby>pensi<rt class="ruby-X\_X\_X">年金</rt></ruby>on'], ['ordon', '<ruby>ordon<rt class="ruby-M\_M\_M">(を)命令する</rt></ruby>', '<ruby>ord<rt class="ruby-L\_L\_L">順序</rt></ruby>on'], ['eskadron', 'eskadron', '<ruby>eskadr<rt class="ruby-L\_L\_L">[軍]艦隊</rt></ruby>on'], ['senton', '<ruby>sen<rt class="ruby-S\_S\_S">(~)なしで</rt></ruby><ruby>ton<rt class="ruby-M\_M\_M">[楽]楽音</rt></ruby>', '<ruby>sent<rt class="ruby-M\_M\_M">(を)感じる</rt></ruby>on'], ['eston', 'eston', '<ruby>est<rt class="ruby-S\_S\_S">(~)である</rt></ruby>on'], ['fanfaron', '<ruby>fanfaron<rt class="ruby-L\_L\_L">大言壮語する</rt></ruby>', '<ruby>fanfar<rt class="ruby-S\_S\_S">[楽]ファンファーレ</rt></ruby>on'], ['fero', 'fero', '<ruby>fer<rt class="ruby-X\_X\_X">鉄</rt></ruby>o'], ['feston', '<ruby>feston<rt class="ruby-X\_X\_X">花綱</rt></ruby>', '<ruby>fest<rt class="ruby-M\_M\_M">(を)祝う</rt></ruby>on'], ['flegmon', 'flegmon', '<ruby>flegm<rt class="ruby-X\_X\_X">冷静</rt></ruby>on'], ['fronton', '<ruby>fronton<rt class="ruby-M\_M\_M">[建]ペディメント</rt></ruby>', '<ruby>front<rt class="ruby-X\_X\_X">正面</rt></ruby>on'], ['galon', '<ruby>galon<rt class="ruby-M\_M\_M">[服]モール</rt></ruby>', '<ruby>gal<rt class="ruby-M\_M\_M">[生]胆汁</rt></ruby>on'], ['mason', '<ruby>mason<rt class="ruby-X\_X\_X">築く</rt></ruby>', '<ruby>mas<rt class="ruby-M\_M\_M">かたまり</rt></ruby>on'], ['helikon', 'helikon', '<ruby>helik<rt class="ruby-S\_S\_S">[動]カタツムリ</rt></ruby>on'], ['kanon', '<ruby>kanon<rt class="ruby-L\_L\_L">[軍]大砲</rt></ruby>', '<ruby>kan<rt class="ruby-M\_M\_M">[植]アシ</rt></ruby>on'], ['kapon', '<ruby>kapon<rt class="ruby-M\_M\_M">去勢オンドリ</rt></ruby>', '<ruby>kap<rt class="ruby-X\_X\_X">頭</rt></ruby>on'], ['kokon', '<ruby>kokon<rt class="ruby-M\_M\_M">[虫]繭(まゆ)</rt></ruby>', '<ruby>kok<rt class="ruby-M\_M\_M">ニワトリ</rt></ruby>on'], ['kolon', '<ruby>kolon<rt class="ruby-L\_L\_L">[建]円柱</rt></ruby>', '<ruby>kol<rt class="ruby-M\_M\_M">[解]首</rt></ruby>on'], ['komision', '<ruby>komision<rt class="ruby-L\_L\_L">(調査)委員会</rt></ruby>', '<ruby>komisi<rt class="ruby-M\_M\_M">(を)委託する</rt></ruby>on'], ['salon', '<ruby>salon<rt class="ruby-L\_L\_L">サロン</rt></ruby>', '<ruby>sal<rt class="ruby-X\_X\_X">塩</rt></ruby>on'], ['ponton', '<ruby>ponton<rt class="ruby-L\_L\_L">[軍]平底舟</rt></ruby>', '<ruby>pont<rt class="ruby-X\_X\_X">橋</rt></ruby>on'], ['koton', '<ruby>koton<rt class="ruby-X\_X\_X">綿</rt></ruby>', '<ruby>kot<rt class="ruby-X\_X\_X">泥</rt></ruby>on'], ['kripton', 'kripton', '<ruby>kript<rt class="ruby-M\_M\_M">[宗]地下聖堂</rt></ruby>on'], ['kupon', '<ruby>kupon<rt class="ruby-M\_M\_M">クーポン券</rt></ruby>', '<ruby>kup<rt class="ruby-M\_M\_M">吸い玉</rt></ruby>on'], ['lakon', 'lakon', '<ruby>lak<rt class="ruby-M\_M\_M">ラッカー</rt></ruby>on'], ['ludon', '<ruby>lu<rt class="ruby-S\_S\_S">賃借する</rt></ruby><ruby>don<rt class="ruby-M\_M\_M">与える</rt></ruby>', '<ruby>lud<rt class="ruby-M\_M\_M">(を)遊ぶ</rt></ruby>on'], ['melon', '<ruby>melon<rt class="ruby-M\_M\_M">[果]メロン</rt></ruby>', '<ruby>mel<rt class="ruby-M\_M\_M">アナグマ</rt></ruby>on'], ['menton', '<ruby>menton<rt class="ruby-L\_L\_L">[解]下あご</rt></ruby>', '<ruby>ment<rt class="ruby-M\_M\_M">[植]ハッカ</rt></ruby>on'], ['milion', '<ruby>milion<rt class="ruby-X\_X\_X">百万</rt></ruby>', '<ruby>mili<rt class="ruby-M\_M\_M">[植]キビ</rt></ruby>on'], ['milionon', '<ruby>milion<rt class="ruby-X\_X\_X">百万</rt></ruby><ruby>on<rt class="ruby-M\_M\_M">分数</rt></ruby>', '<ruby>milion<rt class="ruby-X\_X\_X">百万</rt></ruby>on'], ['nauxon', '<ruby>naux<rt class="ruby-X\_X\_X">九</rt></ruby><ruby>on<rt class="ruby-M\_M\_M">分数</rt></ruby>', '<ruby>naux<rt class="ruby-X\_X\_X">九</rt></ruby>on'], ['violon', '<ruby>violon<rt class="ruby-M\_M\_M">[楽]バイオリン</rt></ruby>', '<ruby>viol<rt class="ruby-M\_M\_M">[植]スミレ</rt></ruby>on'], ['refoj', '<ruby>re<rt class="ruby-M\_M\_M">再び</rt></ruby><ruby>foj<rt class="ruby-X\_X\_X">回</rt></ruby>', '<ruby>ref<rt class="ruby-M\_M\_M">リーフ</rt></ruby>oj'], ['trombon', '<ruby>trombon<rt class="ruby-M\_M\_M">[楽]トロンボーン</rt></ruby>', '<ruby>tromb<rt class="ruby-M\_M\_M">[気]たつまき</rt></ruby>on'], ['samo', 'samo', '<ruby>sam<rt class="ruby-M\_M\_M">同一の</rt></ruby>o'], ['savoj', 'savoj', '<ruby>sav<rt class="ruby-M\_M\_M">救助する</rt></ruby>oj'], ['senson', '<ruby>sen<rt class="ruby-S\_S\_S">(~)なしで</rt></ruby><ruby>son<rt class="ruby-M\_M\_M">音がする</rt></ruby>', '<ruby>sens<rt class="ruby-M\_M\_M">[生]感覚</rt></ruby>on'], ['sepon', '<ruby>sep<rt class="ruby-X\_X\_X">七</rt></ruby><ruby>on<rt class="ruby-M\_M\_M">分数</rt></ruby>', '<ruby>sep<rt class="ruby-X\_X\_X">七</rt></ruby>on'], ['skadron', 'skadron', '<ruby>skadr<rt class="ruby-M\_M\_M">[軍]騎兵中隊</rt></ruby>on'], ['stadion', '<ruby>stadion<rt class="ruby-L\_L\_L">スタジアム</rt></ruby>', '<ruby>stadi<rt class="ruby-X\_X\_X">段階</rt></ruby>on'], ['tetraon', 'tetraon', '<ruby>tetra<rt class="ruby-S\_S\_S">エゾライチョウ</rt></ruby>on'], ['timon', '<ruby>timon<rt class="ruby-L\_L\_L">かじ棒</rt></ruby>', '<ruby>tim<rt class="ruby-M\_M\_M">恐れる</rt></ruby>on'], ['valon', 'valon', '<ruby>val<rt class="ruby-M\_M\_M">[地]谷</rt></ruby>on'], ['veto', 'veto', '<ruby>vet<rt class="ruby-M\_M\_M">賭ける</rt></ruby>o']]

# on系以外は、'fero','refoj','samo','savoj','veto'

if "形容词" in j[1]:

for k in ["a","aj",'an']:

if not i+k in pre\_replacements\_dict\_2:

pre\_replacements\_dict\_3[i+k]=[j[0]+k,j[2]+len(k)\*10000-3000]

elif j[0]+k != pre\_replacements\_dict\_2[i+k][0]:

pre\_replacements\_dict\_3[i+k]=[j[0]+k,j[2]+len(k)\*10000-3000]# 新しく作った方の語根分解を優先する つまり、"an"は形容詞語尾として語根分解する。

unchangeable\_after\_creation\_list.append(i+k)

# an系 [['dietan', '<ruby>diet<rt class="ruby-M\_M\_M">[医]規定食</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>diet<rt class="ruby-M\_M\_M">[医]規定食</rt></ruby>an'], ['afrikan', '<ruby>afrik<rt class="ruby-S\_S\_S">[地名]アフリカ</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>afrik<rt class="ruby-S\_S\_S">[地名]アフリカ</rt></ruby>an'], ['movadan', '<ruby>mov<rt class="ruby-M\_M\_M">動かす</rt></ruby><ruby>ad<rt class="ruby-S\_S\_S">継続行為</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>mov<rt class="ruby-M\_M\_M">動かす</rt></ruby><ruby>ad<rt class="ruby-S\_S\_S">継続行為</rt></ruby>an'], ['akcian', '<ruby>akci<rt class="ruby-M\_M\_M">[商]株式</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>akci<rt class="ruby-M\_M\_M">[商]株式</rt></ruby>an'], ['montaran', '<ruby>mont<rt class="ruby-X\_X\_X">山</rt></ruby><ruby>ar<rt class="ruby-M\_M\_M">集団</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>mont<rt class="ruby-X\_X\_X">山</rt></ruby><ruby>ar<rt class="ruby-M\_M\_M">集団</rt></ruby>an'], ['amerikan', '<ruby>amerik<rt class="ruby-M\_M\_M">[地名]アメリカ</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>amerik<rt class="ruby-M\_M\_M">[地名]アメリカ</rt></ruby>an'], ['regnan', '<ruby>regn<rt class="ruby-M\_M\_M">[法]国家</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>regn<rt class="ruby-M\_M\_M">[法]国家</rt></ruby>an'], ['dezertan', '<ruby>dezert<rt class="ruby-X\_X\_X">砂漠</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>dezert<rt class="ruby-X\_X\_X">砂漠</rt></ruby>an'], ['asocian', '<ruby>asoci<rt class="ruby-X\_X\_X">協会</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>asoci<rt class="ruby-X\_X\_X">協会</rt></ruby>an'], ['insulan', '<ruby>insul<rt class="ruby-X\_X\_X">島</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>insul<rt class="ruby-X\_X\_X">島</rt></ruby>an'], ['azian', '<ruby>azi<rt class="ruby-M\_M\_M">アジア</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>azi<rt class="ruby-M\_M\_M">アジア</rt></ruby>an'], ['sxtatan', '<ruby>sxtat<rt class="ruby-X\_X\_X">国家</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>sxtat<rt class="ruby-X\_X\_X">国家</rt></ruby>an'], ['doman', '<ruby>dom<rt class="ruby-X\_X\_X">家</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>dom<rt class="ruby-X\_X\_X">家</rt></ruby>an'], ['montan', '<ruby>mont<rt class="ruby-X\_X\_X">山</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>mont<rt class="ruby-X\_X\_X">山</rt></ruby>an'], ['familian', '<ruby>famili<rt class="ruby-X\_X\_X">家族</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>famili<rt class="ruby-X\_X\_X">家族</rt></ruby>an'], ['urban', '<ruby>urb<rt class="ruby-X\_X\_X">市</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>urb<rt class="ruby-X\_X\_X">市</rt></ruby>an'], ['inka', 'inka', '<ruby>ink<rt class="ruby-M\_M\_M">インク</rt></ruby>a'], ['popolan', '<ruby>popol<rt class="ruby-X\_X\_X">人民</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>popol<rt class="ruby-X\_X\_X">人民</rt></ruby>an'], ['dekan', '<ruby>dekan<rt class="ruby-L\_L\_L">学部長</rt></ruby>', '<ruby>dek<rt class="ruby-X\_X\_X">十</rt></ruby>an'], ['partian', '<ruby>parti<rt class="ruby-L\_L\_L">[政]党派</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>parti<rt class="ruby-L\_L\_L">[政]党派</rt></ruby>an'], ['lokan', '<ruby>lok<rt class="ruby-L\_L\_L">場所</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>lok<rt class="ruby-L\_L\_L">場所</rt></ruby>an'], ['sxipan', '<ruby>sxip<rt class="ruby-X\_X\_X">船</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>sxip<rt class="ruby-X\_X\_X">船</rt></ruby>an'], ['eklezian', '<ruby>eklezi<rt class="ruby-L\_L\_L">[宗]教会</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>eklezi<rt class="ruby-L\_L\_L">[宗]教会</rt></ruby>an'], ['landan', '<ruby>land<rt class="ruby-X\_X\_X">国</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>land<rt class="ruby-X\_X\_X">国</rt></ruby>an'], ['orientan', '<ruby>orient<rt class="ruby-M\_M\_M">方位定める;東</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>orient<rt class="ruby-M\_M\_M">方位定める;東</rt></ruby>an'], ['lernejan', '<ruby>lern<rt class="ruby-S\_S\_S">(を)学習する</rt></ruby><ruby>ej<rt class="ruby-M\_M\_M">場所</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>lern<rt class="ruby-S\_S\_S">(を)学習する</rt></ruby><ruby>ej<rt class="ruby-M\_M\_M">場所</rt></ruby>an'], ['enlandan', '<ruby>en<rt class="ruby-M\_M\_M">中で</rt></ruby><ruby>land<rt class="ruby-X\_X\_X">国</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>en<rt class="ruby-M\_M\_M">中で</rt></ruby><ruby>land<rt class="ruby-X\_X\_X">国</rt></ruby>an'], ['kalkan', '<ruby>kalkan<rt class="ruby-X\_X\_X">[解]踵</rt></ruby>', '<ruby>kalk<rt class="ruby-M\_M\_M">[化]石灰</rt></ruby>an'], ['estraran', '<ruby>estr<rt class="ruby-M\_M\_M">[接尾辞]長</rt></ruby><ruby>ar<rt class="ruby-M\_M\_M">集団</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>estr<rt class="ruby-M\_M\_M">[接尾辞]長</rt></ruby><ruby>ar<rt class="ruby-M\_M\_M">集団</rt></ruby>an'], ['etnan', '<ruby>etn<rt class="ruby-L\_L\_L">民族</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>etn<rt class="ruby-L\_L\_L">民族</rt></ruby>an'], ['euxropan', '<ruby>euxrop<rt class="ruby-L\_L\_L">ヨーロッパ</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>euxrop<rt class="ruby-L\_L\_L">ヨーロッパ</rt></ruby>an'], ['fazan', '<ruby>fazan<rt class="ruby-L\_L\_L">[鳥]キジ</rt></ruby>', '<ruby>faz<rt class="ruby-M\_M\_M">[理]位相</rt></ruby>an'], ['polican', '<ruby>polic<rt class="ruby-X\_X\_X">警察</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>polic<rt class="ruby-X\_X\_X">警察</rt></ruby>an'], ['socian', '<ruby>soci<rt class="ruby-X\_X\_X">社会</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>soci<rt class="ruby-X\_X\_X">社会</rt></ruby>an'], ['societan', '<ruby>societ<rt class="ruby-X\_X\_X">会</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>societ<rt class="ruby-X\_X\_X">会</rt></ruby>an'], ['grupan', '<ruby>grup<rt class="ruby-M\_M\_M">グループ</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>grup<rt class="ruby-M\_M\_M">グループ</rt></ruby>an'], ['havaj', 'havaj', '<ruby>hav<rt class="ruby-S\_S\_S">持っている</rt></ruby>aj'], ['ligan', '<ruby>lig<rt class="ruby-S\_S\_S">結ぶ;連盟</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>lig<rt class="ruby-S\_S\_S">結ぶ;連盟</rt></ruby>an'], ['nacian', '<ruby>naci<rt class="ruby-X\_X\_X">国民</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>naci<rt class="ruby-X\_X\_X">国民</rt></ruby>an'], ['koran', '<ruby>koran<rt class="ruby-M\_M\_M">[宗]コーラン</rt></ruby>', '<ruby>kor<rt class="ruby-X\_X\_X">心</rt></ruby>an'], ['religian', '<ruby>religi<rt class="ruby-X\_X\_X">宗教</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>religi<rt class="ruby-X\_X\_X">宗教</rt></ruby>an'], ['kuban', '<ruby>kub<rt class="ruby-M\_M\_M">立方体</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>kub<rt class="ruby-M\_M\_M">立方体</rt></ruby>an'], ['lama', '<ruby>lama<rt class="ruby-M\_M\_M">[宗]ラマ僧</rt></ruby>', '<ruby>lam<rt class="ruby-M\_M\_M">びっこの</rt></ruby>a'], ['majoran', '<ruby>major<rt class="ruby-M\_M\_M">[軍]陸軍少佐</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>major<rt class="ruby-M\_M\_M">[軍]陸軍少佐</rt></ruby>an'], ['malaj', 'malaj', '<ruby>mal<rt class="ruby-M\_M\_M">正反対</rt></ruby>aj'], ['marian', 'marian', '<ruby>mari<rt class="ruby-L\_L\_L">マリア</rt></ruby>an'], ['nordan', '<ruby>nord<rt class="ruby-X\_X\_X">北</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>nord<rt class="ruby-X\_X\_X">北</rt></ruby>an'], ['paran', 'paran', '<ruby>par<rt class="ruby-L\_L\_L">一対</rt></ruby>an'], ['parizan', '<ruby>pariz<rt class="ruby-M\_M\_M">[地名]パリ</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>pariz<rt class="ruby-M\_M\_M">[地名]パリ</rt></ruby>an'], ['parokan', '<ruby>parok<rt class="ruby-L\_L\_L">[宗]教区</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>parok<rt class="ruby-L\_L\_L">[宗]教区</rt></ruby>an'], ['podian', '<ruby>podi<rt class="ruby-L\_L\_L">ひな壇</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>podi<rt class="ruby-L\_L\_L">ひな壇</rt></ruby>an'], ['rusian', '<ruby>rus<rt class="ruby-M\_M\_M">ロシア人</rt></ruby>i<ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>rus<rt class="ruby-M\_M\_M">ロシア人</rt></ruby>ian'], ['satan', '<ruby>satan<rt class="ruby-M\_M\_M">[宗]サタン</rt></ruby>', '<ruby>sat<rt class="ruby-M\_M\_M">満腹した</rt></ruby>an'], ['sektan', '<ruby>sekt<rt class="ruby-M\_M\_M">[宗]宗派</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>sekt<rt class="ruby-M\_M\_M">[宗]宗派</rt></ruby>an'], ['senatan', '<ruby>senat<rt class="ruby-M\_M\_M">[政]参議院</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>senat<rt class="ruby-M\_M\_M">[政]参議院</rt></ruby>an'], ['skisman', '<ruby>skism<rt class="ruby-M\_M\_M">(団体の)分裂</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>skism<rt class="ruby-M\_M\_M">(団体の)分裂</rt></ruby>an'], ['sudan', 'sudan', '<ruby>sud<rt class="ruby-X\_X\_X">南</rt></ruby>an'], ['utopian', '<ruby>utopi<rt class="ruby-M\_M\_M">ユートピア</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>utopi<rt class="ruby-M\_M\_M">ユートピア</rt></ruby>an'], ['vilagxan', '<ruby>vilagx<rt class="ruby-X\_X\_X">村</rt></ruby><ruby>an<rt class="ruby-M\_M\_M">会員</rt></ruby>', '<ruby>vilagx<rt class="ruby-X\_X\_X">村</rt></ruby>an']]

# an系以外は'inka','malaj','havaj','lama'　　'marian'については、'マリアan'で行く。

if "副词" in j[1]:

for k in ["e"]:

if not i+k in pre\_replacements\_dict\_2:

pre\_replacements\_dict\_3[i+k]=[j[0]+k,j[2]+len(k)\*10000-3000]

elif j[0]+k != pre\_replacements\_dict\_2[i+k][0]:

pre\_replacements\_dict\_3[i+k]=[j[0]+k,j[2]+len(k)\*10000-3000]# 新しく作った方の語根分解を優先する

unchangeable\_after\_creation\_list.append(i+k)

# [['alte', '<ruby>alte<rt class="ruby-M\_M\_M">タチアオイ</rt></ruby>', '<ruby>alt<rt class="ruby-L\_L\_L">高い</rt></ruby>e'], ['apoge', '<ruby>apoge<rt class="ruby-M\_M\_M">[天]遠地点</rt></ruby>', '<ruby>apog<rt class="ruby-M\_M\_M">(を)支える</rt></ruby>e'], ['kaze', '<ruby>kaze<rt class="ruby-M\_M\_M">[化]凝乳</rt></ruby>', '<ruby>kaz<rt class="ruby-M\_M\_M">[文]格</rt></ruby>e'], ['pere', '<ruby>pere<rt class="ruby-M\_M\_M">破滅する</rt></ruby>', '<ruby>per<rt class="ruby-M\_M\_M">よって</rt></ruby>e'], ['kore', 'kore', '<ruby>kor<rt class="ruby-X\_X\_X">心</rt></ruby>e'], ['male', 'male', '<ruby>mal<rt class="ruby-M\_M\_M">正反対</rt></ruby>e'], ['sole', '<ruby>sole<rt class="ruby-M\_M\_M">シタビラメ</rt></ruby>', '<ruby>sol<rt class="ruby-M\_M\_M">唯一の</rt></ruby>e']]

if "动词" in j[1]:

for k1,k2 in verb\_suffix\_2l\_2.items():

if not i+k1 in pre\_replacements\_dict\_2:

pre\_replacements\_dict\_3[i+k1]=[j[0]+k2,j[2]+len(k1)\*10000-3000]

elif j[0]+k2 != pre\_replacements\_dict\_2[i+k1][0]:

pre\_replacements\_dict\_3[i+k1]=[j[0]+k2,j[2]+len(k1)\*10000-3000]# 新しく作った方の語根分解を優先する

unchangeable\_after\_creation\_list.append(i+k1)

# [['regulus', 'regulus', '<ruby>regul<rt class="ruby-X\_X\_X">規則</rt></ruby><ruby>us<rt class="ruby-S\_S\_S">条件法</rt></ruby>'], ['akirant', 'akirant', '<ruby>akir<rt class="ruby-S\_S\_S">(を)獲得する</rt></ruby><ruby>ant<rt class="ruby-S\_S\_S">能動;継続</rt></ruby>'], ['radius', 'radius', '<ruby>radi<rt class="ruby-L\_L\_L">[理]線</rt></ruby><ruby>us<rt class="ruby-S\_S\_S">条件法</rt></ruby>'], ['premis', '<ruby>premis<rt class="ruby-X\_X\_X">前提</rt></ruby>', '<ruby>prem<rt class="ruby-M\_M\_M">(を)押える</rt></ruby><ruby>is<rt class="ruby-S\_S\_S">過去形</rt></ruby>'], ['sonat', '<ruby>sonat<rt class="ruby-M\_M\_M">[楽]ソナタ</rt></ruby>', '<ruby>son<rt class="ruby-M\_M\_M">音がする</rt></ruby><ruby>at<rt class="ruby-S\_S\_S">受動継続</rt></ruby>'], ['format', '<ruby>format<rt class="ruby-X\_X\_X">[印]判</rt></ruby>', '<ruby>form<rt class="ruby-X\_X\_X">形</rt></ruby><ruby>at<rt class="ruby-S\_S\_S">受動継続</rt></ruby>'], ['markot', '<ruby>markot<rt class="ruby-L\_L\_L">[園]取木</rt></ruby>', '<ruby>mark<rt class="ruby-L\_L\_L">しるし</rt></ruby><ruby>ot<rt class="ruby-S\_S\_S">受動将然</rt></ruby>'], ['nomad', '<ruby>nomad<rt class="ruby-L\_L\_L">遊牧民</rt></ruby>', '<ruby>nom<rt class="ruby-L\_L\_L">名前</rt></ruby><ruby>ad<rt class="ruby-S\_S\_S">継続行為</rt></ruby>'], ['kantat', '<ruby>kantat<rt class="ruby-M\_M\_M">[楽]カンタータ</rt></ruby>', '<ruby>kant<rt class="ruby-M\_M\_M">(を)歌う</rt></ruby><ruby>at<rt class="ruby-S\_S\_S">受動継続</rt></ruby>'], ['kolorad', 'kolorad', '<ruby>kolor<rt class="ruby-X\_X\_X">色</rt></ruby><ruby>ad<rt class="ruby-S\_S\_S">継続行為</rt></ruby>'], ['diplomat', '<ruby>diplomat<rt class="ruby-X\_X\_X">外交官</rt></ruby>', '<ruby>diplom<rt class="ruby-X\_X\_X">免状</rt></ruby><ruby>at<rt class="ruby-S\_S\_S">受動継続</rt></ruby>'], ['diskont', '<ruby>diskont<rt class="ruby-M\_M\_M">[商]手形割引する</rt></ruby>', '<ruby>disk<rt class="ruby-X\_X\_X">円盤</rt></ruby><ruby>ont<rt class="ruby-S\_S\_S">能動;将然</rt></ruby>'], ['endos', 'endos', '<ruby>end<rt class="ruby-L\_L\_L">必要</rt></ruby><ruby>os<rt class="ruby-S\_S\_S">未来形</rt></ruby>'], ['esperant', '<ruby>esperant<rt class="ruby-L\_L\_L">エスペラント</rt></ruby>', '<ruby>esper<rt class="ruby-M\_M\_M">(を)希望する</rt></ruby><ruby>ant<rt class="ruby-S\_S\_S">能動;継続</rt></ruby>'], ['forkant', '<ruby>for<rt class="ruby-M\_M\_M">離れて</rt></ruby><ruby>kant<rt class="ruby-M\_M\_M">(を)歌う</rt></ruby>', '<ruby>fork<rt class="ruby-S\_S\_S">[料]フォーク</rt></ruby><ruby>ant<rt class="ruby-S\_S\_S">能動;継続</rt></ruby>'], ['gravit', 'gravit', '<ruby>grav<rt class="ruby-L\_L\_L">重要な</rt></ruby><ruby>it<rt class="ruby-S\_S\_S">受動完了</rt></ruby>'], ['konus', '<ruby>konus<rt class="ruby-L\_L\_L">[数]円錐</rt></ruby>', '<ruby>kon<rt class="ruby-S\_S\_S">知っている</rt></ruby><ruby>us<rt class="ruby-S\_S\_S">条件法</rt></ruby>'], ['salat', '<ruby>salat<rt class="ruby-M\_M\_M">[料]サラダ</rt></ruby>', '<ruby>sal<rt class="ruby-X\_X\_X">塩</rt></ruby><ruby>at<rt class="ruby-S\_S\_S">受動継続</rt></ruby>'], ['legat', '<ruby>legat<rt class="ruby-M\_M\_M">[宗]教皇特使</rt></ruby>', '<ruby>leg<rt class="ruby-M\_M\_M">(を)読む</rt></ruby><ruby>at<rt class="ruby-S\_S\_S">受動継続</rt></ruby>'], ['lekant', '<ruby>lekant<rt class="ruby-M\_M\_M">[植]マーガレット</rt></ruby>', '<ruby>lek<rt class="ruby-M\_M\_M">なめる</rt></ruby><ruby>ant<rt class="ruby-S\_S\_S">能動;継続</rt></ruby>'], ['lotus', '<ruby>lotus<rt class="ruby-L\_L\_L">[植]ハス</rt></ruby>', '<ruby>lot<rt class="ruby-L\_L\_L">くじ</rt></ruby><ruby>us<rt class="ruby-S\_S\_S">条件法</rt></ruby>'], ['malvolont', '<ruby>mal<rt class="ruby-M\_M\_M">正反対</rt></ruby><ruby>volont<rt class="ruby-L\_L\_L">自ら進んで</rt></ruby>', '<ruby>mal<rt class="ruby-M\_M\_M">正反対</rt></ruby><ruby>vol<rt class="ruby-S\_S\_S">意志がある</rt></ruby><ruby>ont<rt class="ruby-S\_S\_S">能動;将然</rt></ruby>'], ['mankis', '<ruby>man<rt class="ruby-X\_X\_X">手</rt></ruby><ruby>kis<rt class="ruby-M\_M\_M">キスする</rt></ruby>', '<ruby>mank<rt class="ruby-M\_M\_M">欠けている</rt></ruby><ruby>is<rt class="ruby-S\_S\_S">過去形</rt></ruby>'], ['minus', '<ruby>minus<rt class="ruby-L\_L\_L">マイナス</rt></ruby>', '<ruby>min<rt class="ruby-L\_L\_L">鉱山</rt></ruby><ruby>us<rt class="ruby-S\_S\_S">条件法</rt></ruby>'], ['patos', '<ruby>patos<rt class="ruby-M\_M\_M">[芸]パトス</rt></ruby>', '<ruby>pat<rt class="ruby-S\_S\_S">フライパン</rt></ruby><ruby>os<rt class="ruby-S\_S\_S">未来形</rt></ruby>'], ['predikat', '<ruby>predikat<rt class="ruby-X\_X\_X">[文]述部</rt></ruby>', '<ruby>predik<rt class="ruby-M\_M\_M">(を)説教する</rt></ruby><ruby>at<rt class="ruby-S\_S\_S">受動継続</rt></ruby>'], ['rabat', '<ruby>rabat<rt class="ruby-L\_L\_L">[商]割引</rt></ruby>', '<ruby>rab<rt class="ruby-M\_M\_M">強奪する</rt></ruby><ruby>at<rt class="ruby-S\_S\_S">受動継続</rt></ruby>'], ['rabot', '<ruby>rabot<rt class="ruby-S\_S\_S">かんなをかける</rt></ruby>', '<ruby>rab<rt class="ruby-M\_M\_M">強奪する</rt></ruby><ruby>ot<rt class="ruby-S\_S\_S">受動将然</rt></ruby>'], ['remont', 'remont', '<ruby>rem<rt class="ruby-L\_L\_L">漕ぐ</rt></ruby><ruby>ont<rt class="ruby-S\_S\_S">能動;将然</rt></ruby>'], ['satirus', 'satirus', '<ruby>satir<rt class="ruby-M\_M\_M">諷刺(詩;文)</rt></ruby><ruby>us<rt class="ruby-S\_S\_S">条件法</rt></ruby>'], ['sendat', '<ruby>sen<rt class="ruby-S\_S\_S">(~)なしで</rt></ruby><ruby>dat<rt class="ruby-L\_L\_L">日付</rt></ruby>', '<ruby>send<rt class="ruby-M\_M\_M">(を)送る</rt></ruby><ruby>at<rt class="ruby-S\_S\_S">受動継続</rt></ruby>'], ['sendot', '<ruby>sen<rt class="ruby-S\_S\_S">(~)なしで</rt></ruby><ruby>dot<rt class="ruby-M\_M\_M">持参金</rt></ruby>', '<ruby>send<rt class="ruby-M\_M\_M">(を)送る</rt></ruby><ruby>ot<rt class="ruby-S\_S\_S">受動将然</rt></ruby>'], ['spirit', '<ruby>spirit<rt class="ruby-X\_X\_X">精神</rt></ruby>', '<ruby>spir<rt class="ruby-M\_M\_M">呼吸する</rt></ruby><ruby>it<rt class="ruby-S\_S\_S">受動完了</rt></ruby>'], ['spirant', 'spirant', '<ruby>spir<rt class="ruby-M\_M\_M">呼吸する</rt></ruby><ruby>ant<rt class="ruby-S\_S\_S">能動;継続</rt></ruby>'], ['taksus', '<ruby>taksus<rt class="ruby-L\_L\_L">[植]イチイ</rt></ruby>', '<ruby>taks<rt class="ruby-S\_S\_S">(を)評価する</rt></ruby><ruby>us<rt class="ruby-S\_S\_S">条件法</rt></ruby>'], ['tenis', 'tenis', '<ruby>ten<rt class="ruby-M\_M\_M">支え持つ</rt></ruby><ruby>is<rt class="ruby-S\_S\_S">過去形</rt></ruby>'], ['traktat', '<ruby>traktat<rt class="ruby-X\_X\_X">[政]条約</rt></ruby>', '<ruby>trakt<rt class="ruby-M\_M\_M">(を)取り扱う</rt></ruby><ruby>at<rt class="ruby-S\_S\_S">受動継続</rt></ruby>'], ['trikot', '<ruby>trikot<rt class="ruby-M\_M\_M">[織]トリコット</rt></ruby>', '<ruby>trik<rt class="ruby-S\_S\_S">編み物をする</rt></ruby><ruby>ot<rt class="ruby-S\_S\_S">受動将然</rt></ruby>'], ['trilit', '<ruby>tri<rt class="ruby-X\_X\_X">三</rt></ruby><ruby>lit<rt class="ruby-M\_M\_M">ベッド</rt></ruby>', '<ruby>tril<rt class="ruby-M\_M\_M">[楽]トリル</rt></ruby><ruby>it<rt class="ruby-S\_S\_S">受動完了</rt></ruby>'], ['vizit', '<ruby>vizit<rt class="ruby-M\_M\_M">(を)訪問する</rt></ruby>', '<ruby>viz<rt class="ruby-L\_L\_L">ビザ</rt></ruby><ruby>it<rt class="ruby-S\_S\_S">受動完了</rt></ruby>'], ['volont', '<ruby>volont<rt class="ruby-L\_L\_L">自ら進んで</rt></ruby>', '<ruby>vol<rt class="ruby-S\_S\_S">意志がある</rt></ruby><ruby>ont<rt class="ruby-S\_S\_S">能動;将然</rt></ruby>']]

for k in ["u ","i ","u","i"]:# 动词の"u","i"単体の接尾辞は後ろが空白と決まっているので、2文字分増やすことができる。

if not i+k in pre\_replacements\_dict\_2:

pre\_replacements\_dict\_3[i+k]=[j[0]+k,j[2]+len(k)\*10000-3000]

elif j[0]+k != pre\_replacements\_dict\_2[i+k][0]:

pre\_replacements\_dict\_3[i+k]=[j[0]+k,j[2]+len(k)\*10000-3000]# 新しく作った方の語根分解を優先する

unchangeable\_after\_creation\_list.append(i+k)

# [['agxi', '<ruby>agxi<rt class="ruby-L\_L\_L">打ち歩</rt></ruby>', '<ruby>agx<rt class="ruby-L\_L\_L">年齢</rt></ruby>i'], ['premi', '<ruby>premi<rt class="ruby-X\_X\_X">賞品</rt></ruby>', '<ruby>prem<rt class="ruby-M\_M\_M">(を)押える</rt></ruby>i'], ['bari', 'bari', '<ruby>bar<rt class="ruby-L\_L\_L">障害</rt></ruby>i'], ['tempi', '<ruby>tempi<rt class="ruby-L\_L\_L">こめかみ</rt></ruby>', '<ruby>temp<rt class="ruby-X\_X\_X">時間</rt></ruby>i'], ['noktu', '<ruby>noktu<rt class="ruby-S\_S\_S">[鳥]コフクロウ</rt></ruby>', '<ruby>nokt<rt class="ruby-X\_X\_X">夜</rt></ruby>u'], ['vakcini', 'vakcini', '<ruby>vakcin<rt class="ruby-M\_M\_M">[薬]ワクチン</rt></ruby>i'], ['procesi', '<ruby>procesi<rt class="ruby-X\_X\_X">[宗]行列</rt></ruby>', '<ruby>proces<rt class="ruby-L\_L\_L">[法]訴訟</rt></ruby>i'], ['statu', '<ruby>statu<rt class="ruby-X\_X\_X">立像</rt></ruby>', '<ruby>stat<rt class="ruby-X\_X\_X">状態</rt></ruby>u'], ['devi', 'devi', '<ruby>dev<rt class="ruby-L\_L\_L">must</rt></ruby>i'], ['feri', '<ruby>feri<rt class="ruby-X\_X\_X">休日</rt></ruby>', '<ruby>fer<rt class="ruby-X\_X\_X">鉄</rt></ruby>i'], ['fleksi', '<ruby>fleksi<rt class="ruby-M\_M\_M">[文]語尾変化</rt></ruby>', '<ruby>fleks<rt class="ruby-M\_M\_M">(を)曲げる</rt></ruby>i'], ['pensi', '<ruby>pensi<rt class="ruby-X\_X\_X">年金</rt></ruby>', '<ruby>pens<rt class="ruby-X\_X\_X">思う</rt></ruby>i'], ['jesu', '<ruby>jesu<rt class="ruby-M\_M\_M">[宗]イエス</rt></ruby>', '<ruby>jes<rt class="ruby-L\_L\_L">はい</rt></ruby>u'], ['jxaluzi', 'jxaluzi', '<ruby>jxaluz<rt class="ruby-L\_L\_L">嫉妬深い</rt></ruby>i'], ['konfesi', 'konfesi', '<ruby>konfes<rt class="ruby-M\_M\_M">(を)告白する</rt></ruby>i'], ['konsili', 'konsili', '<ruby>konsil<rt class="ruby-M\_M\_M">(を)助言する</rt></ruby>i'], ['legi', '<ruby>legi<rt class="ruby-M\_M\_M">[史]軍団</rt></ruby>', '<ruby>leg<rt class="ruby-M\_M\_M">(を)読む</rt></ruby>i'], ['licenci', 'licenci', '<ruby>licenc<rt class="ruby-L\_L\_L">[商]認可</rt></ruby>i'], ['logxi', '<ruby>logxi<rt class="ruby-L\_L\_L">[劇]桟敷</rt></ruby>', '<ruby>logx<rt class="ruby-M\_M\_M">(に)住む</rt></ruby>i'], ['meti', '<ruby>meti<rt class="ruby-L\_L\_L">手仕事</rt></ruby>', '<ruby>met<rt class="ruby-M\_M\_M">(を)置く</rt></ruby>i'], ['pasi', '<ruby>pasi<rt class="ruby-X\_X\_X">情熱</rt></ruby>', '<ruby>pas<rt class="ruby-M\_M\_M">通過する</rt></ruby>i'], ['revu', '<ruby>revu<rt class="ruby-M\_M\_M">専門雑誌</rt></ruby>', '<ruby>rev<rt class="ruby-M\_M\_M">空想する</rt></ruby>u'], ['rabi', '<ruby>rabi<rt class="ruby-M\_M\_M">[病]狂犬病</rt></ruby>', '<ruby>rab<rt class="ruby-M\_M\_M">強奪する</rt></ruby>i'], ['religi', '<ruby>religi<rt class="ruby-X\_X\_X">宗教</rt></ruby>', '<ruby>re<rt class="ruby-M\_M\_M">再び</rt></ruby><ruby>lig<rt class="ruby-S\_S\_S">結ぶ;連盟</rt></ruby>i'], ['sagu', '<ruby>sagu<rt class="ruby-M\_M\_M">[料]サゴ粉</rt></ruby>', '<ruby>sag<rt class="ruby-X\_X\_X">矢</rt></ruby>u'], ['sekci', '<ruby>sekci<rt class="ruby-X\_X\_X">部</rt></ruby>', '<ruby>sekc<rt class="ruby-S\_S\_S">[医]切断する</rt></ruby>i'], ['sendi', '<ruby>sen<rt class="ruby-S\_S\_S">(~)なしで</rt></ruby><ruby>di<rt class="ruby-X\_X\_X">神</rt></ruby>', '<ruby>send<rt class="ruby-M\_M\_M">(を)送る</rt></ruby>i'], ['teni', '<ruby>teni<rt class="ruby-M\_M\_M">サナダムシ</rt></ruby>', '<ruby>ten<rt class="ruby-M\_M\_M">支え持つ</rt></ruby>i'], ['vaku', 'vaku', '<ruby>vak<rt class="ruby-S\_S\_S">あいている</rt></ruby>u'], ['vizi', '<ruby>vizi<rt class="ruby-X\_X\_X">幻影</rt></ruby>', '<ruby>viz<rt class="ruby-L\_L\_L">ビザ</rt></ruby>i']]

elif len(i)>=3 and len(i)<=6:# 3文字から6文字の語根で置換しないもの　　結局2文字の語根で置換しないものについては、完全に除外している。

if "名词" in j[1]:# 名词については形容词、副词と違い、置換しないものにもoをつける。

for k in ["o"]:

if not i+k in pre\_replacements\_dict\_2:

pre\_replacements\_dict\_3[i+k]=[j[0]+k,j[2]+len(k)\*10000-5000]# 実質3000# 存でないものは優先順位を大きく下げる→普通の品詞接尾辞が既存でないという言い方はおかしい気がしてきた。(20240612)

elif j[0]+k != pre\_replacements\_dict\_2[i+k][0]:

pass

if "形容词" in j[1]:

for k in ["a"]:

if not i+k in pre\_replacements\_dict\_2:

pre\_replacements\_dict\_3[i+k]=[j[0]+k,j[2]+len(k)\*10000-5000]

elif j[0]+k != pre\_replacements\_dict\_2[i+k][0]:

pass

if "副词" in j[1]:

for k in ["e"]:

if not i+k in pre\_replacements\_dict\_2:

pre\_replacements\_dict\_3[i+k]=[j[0]+k,j[2]+len(k)\*10000-5000]

elif j[0]+k != pre\_replacements\_dict\_2[i+k][0]:

pass

# (8-3) AN, ONリストを用いて更に新しい形を派生(XXXan/o, XXXon/aなど)

for an in AN:

if an[1].endswith("/an/"):

i2 = an[1]

i3 = re.sub(r"/an/$", "", i2)

i4 = i3+"/an/o"

i5 = i3+"/an/a"

i6 = i3+"/an/e"

i7 = i3+"/a/n/"

pre\_replacements\_dict\_3[i4.replace('/', '')] = [safe\_replace(i4,temporary\_replacements\_list\_final).replace("</rt></ruby>","%%%").replace('/', '').replace("%%%","</rt></ruby>"), (len(i4.replace('/', ''))-1)\*10000+3000]

pre\_replacements\_dict\_3[i5.replace('/', '')] = [safe\_replace(i5,temporary\_replacements\_list\_final).replace("</rt></ruby>","%%%").replace('/', '').replace("%%%","</rt></ruby>"), (len(i5.replace('/', ''))-1)\*10000+3000]

pre\_replacements\_dict\_3[i6.replace('/', '')] = [safe\_replace(i6,temporary\_replacements\_list\_final).replace("</rt></ruby>","%%%").replace('/', '').replace("%%%","</rt></ruby>"), (len(i6.replace('/', ''))-1)\*10000+3000]

pre\_replacements\_dict\_3[i7.replace('/', '')] = [safe\_replace(i7,temporary\_replacements\_list\_final).replace("</rt></ruby>","%%%").replace('/', '').replace("%%%","</rt></ruby>"), (len(i7.replace('/', ''))-1)\*10000+3000]

else:

# 末尾に"an"がつくパターンに準じた置換処理

i2 = an[1]

i2\_2 = re.sub(r"an$", "", i2)

i3 = re.sub(r"an/$", "", i2\_2)

i4 = i3+"an/o"

i5 = i3+"an/a"

i6 = i3+"an/e"

i7 = i3+"/a/n/"

pre\_replacements\_dict\_3[i4.replace('/', '')] = [safe\_replace(i4,temporary\_replacements\_list\_final).replace("</rt></ruby>","%%%").replace('/', '').replace("%%%","</rt></ruby>"), (len(i4.replace('/', ''))-1)\*10000+3000]

pre\_replacements\_dict\_3[i5.replace('/', '')] = [safe\_replace(i5,temporary\_replacements\_list\_final).replace("</rt></ruby>","%%%").replace('/', '').replace("%%%","</rt></ruby>"), (len(i5.replace('/', ''))-1)\*10000+3000]

pre\_replacements\_dict\_3[i6.replace('/', '')] = [safe\_replace(i6,temporary\_replacements\_list\_final).replace("</rt></ruby>","%%%").replace('/', '').replace("%%%","</rt></ruby>"), (len(i6.replace('/', ''))-1)\*10000+3000]

pre\_replacements\_dict\_3[i7.replace('/', '')] = [safe\_replace(i7,temporary\_replacements\_list\_final).replace("</rt></ruby>","%%%").replace('/', '').replace("%%%","</rt></ruby>"), (len(i7.replace('/', ''))-1)\*10000+3000]

for on in ON:

if on[1].endswith("/on/"):

i2 = on[1]

i3 = re.sub(r"/on/$", "", i2)

i4 = i3+"/on/o"

i5 = i3+"/on/a"

i6 = i3+"/on/e"

i7 = i3+"/o/n/"

pre\_replacements\_dict\_3[i4.replace('/', '')] = [safe\_replace(i4,temporary\_replacements\_list\_final).replace("</rt></ruby>","%%%").replace('/', '').replace("%%%","</rt></ruby>"), (len(i4.replace('/', ''))-1)\*10000+3000]

pre\_replacements\_dict\_3[i5.replace('/', '')] = [safe\_replace(i5,temporary\_replacements\_list\_final).replace("</rt></ruby>","%%%").replace('/', '').replace("%%%","</rt></ruby>"), (len(i5.replace('/', ''))-1)\*10000+3000]

pre\_replacements\_dict\_3[i6.replace('/', '')] = [safe\_replace(i6,temporary\_replacements\_list\_final).replace("</rt></ruby>","%%%").replace('/', '').replace("%%%","</rt></ruby>"), (len(i6.replace('/', ''))-1)\*10000+3000]

pre\_replacements\_dict\_3[i7.replace('/', '')] = [safe\_replace(i7,temporary\_replacements\_list\_final).replace("</rt></ruby>","%%%").replace('/', '').replace("%%%","</rt></ruby>"), (len(i7.replace('/', ''))-1)\*10000+3000]

else:

i2 = on[1]

i2\_2 = re.sub(r"on$", "", i2)

i3 = re.sub(r"on/$", "", i2\_2)

i4 = i3+"on/o"

i5 = i3+"on/a"

i6 = i3+"on/e"

i7 = i3+"/o/n/"

pre\_replacements\_dict\_3[i4.replace('/', '')] = [safe\_replace(i4,temporary\_replacements\_list\_final).replace("</rt></ruby>","%%%").replace('/', '').replace("%%%","</rt></ruby>"), (len(i4.replace('/', ''))-1)\*10000+3000]

pre\_replacements\_dict\_3[i5.replace('/', '')] = [safe\_replace(i5,temporary\_replacements\_list\_final).replace("</rt></ruby>","%%%").replace('/', '').replace("%%%","</rt></ruby>"), (len(i5.replace('/', ''))-1)\*10000+3000]

pre\_replacements\_dict\_3[i6.replace('/', '')] = [safe\_replace(i6,temporary\_replacements\_list\_final).replace("</rt></ruby>","%%%").replace('/', '').replace("%%%","</rt></ruby>"), (len(i6.replace('/', ''))-1)\*10000+3000]

pre\_replacements\_dict\_3[i7.replace('/', '')] = [safe\_replace(i7,temporary\_replacements\_list\_final).replace("</rt></ruby>","%%%").replace('/', '').replace("%%%","</rt></ruby>"), (len(i7.replace('/', ''))-1)\*10000+3000]

#-------------------------------------------------------------

# (9) custom\_stemming\_setting\_list (ユーザーが定義した語根分解法) を適用

# - 例: ["am", "dflt", ["verbo\_s1"]] → "am"に動詞活用語尾を付けた形を挿入

# - もし優先順位を-1等にしていたら、置換対象から除外(pop)する

#-------------------------------------------------------------

if len(custom\_stemming\_setting\_list) > 0:

# 最初の要素が3つでなければ「説明用の行」とみなし削除

if len(custom\_stemming\_setting\_list[0]) != 3:

custom\_stemming\_setting\_list.pop(0)

for i in custom\_stemming\_setting\_list:

if len(i)==3:

try:

esperanto\_Word\_before\_replacement = i[0].replace('/', '')

if i[1] == "dflt":

replacement\_priority\_by\_length = len(esperanto\_Word\_before\_replacement)\*10000

elif i[1] in allowed\_values:

# 置換優先順位(i[1])が-1等の場合、該当単語を除去

pre\_replacements\_dict\_3.pop(esperanto\_Word\_before\_replacement, None)

if "ne" in i[2]:

pre\_replacements\_dict\_3.pop(esperanto\_Word\_before\_replacement, None)

i[2].remove("ne")

if "verbo\_s1" in i[2]:

for k1 in verb\_suffix\_2l\_2.keys():

removed\_E\_word = esperanto\_Word\_before\_replacement + k1

pre\_replacements\_dict\_3.pop(removed\_E\_word, None)

i[2].remove("verbo\_s1")

if "verbo\_s2" in i[2]:

for k in ["u ", "i ", "u", "i"]:

removed\_E\_word = esperanto\_Word\_before\_replacement + k

pre\_replacements\_dict\_3.pop(removed\_E\_word, None)

i[2].remove("verbo\_s2")

if len(i[2]) >= 1:

for j2\_item in i[2]:

j2\_item2 = j2\_item.replace('/', '')

removed\_E\_word = esperanto\_Word\_before\_replacement + j2\_item2

pre\_replacements\_dict\_3.pop(removed\_E\_word, None)

continue

elif isinstance(i[1], int) or (isinstance(i[1], str) and i[1].isdigit()):

# 整数(もしくは整数文字列)であれば優先順位として使用

replacement\_priority\_by\_length = int(i[1])

# ここで "i[0]"をsafe\_replaceしてルビ等を入れる

Replaced\_String = safe\_replace(i[0], temporary\_replacements\_list\_final)\

.replace("</rt></ruby>","%%%").replace('/', '').replace("%%%","</rt></ruby>")

if "ne" in i[2]:

pre\_replacements\_dict\_3[esperanto\_Word\_before\_replacement] = [Replaced\_String, replacement\_priority\_by\_length]

i[2].remove("ne")

if "verbo\_s1" in i[2]:

for k1,k2 in verb\_suffix\_2l\_2.items():

pre\_replacements\_dict\_3[esperanto\_Word\_before\_replacement + k1] = [Replaced\_String + k2, replacement\_priority\_by\_length+len(k1)\*10000]

i[2].remove("verbo\_s1")

if "verbo\_s2" in i[2]:

for k in ["u ","i ","u","i"]:

pre\_replacements\_dict\_3[esperanto\_Word\_before\_replacement + k] = [Replaced\_String + k, replacement\_priority\_by\_length+len(k)\*10000]

i[2].remove("verbo\_s2")

if len(i[2])>=1:

for j\_item in i[2]:

j2 = j\_item.replace('/', '')

j3 = safe\_replace(j\_item,temporary\_replacements\_list\_final)\

.replace("</rt></ruby>","%%%").replace('/', '').replace("%%%","</rt></ruby>")

pre\_replacements\_dict\_3[esperanto\_Word\_before\_replacement + j2] = [Replaced\_String + j3, replacement\_priority\_by\_length+len(j2)\*10000]

else:

pre\_replacements\_dict\_3[esperanto\_Word\_before\_replacement] = [Replaced\_String, replacement\_priority\_by\_length]

except:

continue

#-------------------------------------------------------------

# (10) user\_replacement\_item\_setting\_list を適用

# こちらはさらに細かい「特定の単語→独自の漢字表記」の設定など

#-------------------------------------------------------------

if len(user\_replacement\_item\_setting\_list) > 0:

# 同様に最初が4つでなければ説明行とみなし削除

if len(user\_replacement\_item\_setting\_list[0]) != 4:

user\_replacement\_item\_setting\_list.pop(0)

for i in user\_replacement\_item\_setting\_list:

if len(i)==4:

try:

# i[0]のように "xxx/yyy" といった複数rootが書かれている場合、"/"でsplit

esperanto\_Roots\_before\_replacement = i[0].strip('/').split('/')

replaced\_roots = i[3].strip('/').split('/')

if len(esperanto\_Roots\_before\_replacement) == len(replaced\_roots):

# 同じ数だけsplitされているならOK

Replaced\_String = ""

for kk in range(len(esperanto\_Roots\_before\_replacement)):

Replaced\_String += output\_format(

esperanto\_Roots\_before\_replacement[kk],

replaced\_roots[kk],

format\_type,

char\_widths\_dict

)

esperanto\_Word\_before\_replacement = i[0].replace('/', '')

if i[1]=="dflt":

replacement\_priority\_by\_length = len(esperanto\_Word\_before\_replacement)\*10000

elif isinstance(i[1], int) or (isinstance(i[1], str) and i[1].isdigit()):

replacement\_priority\_by\_length = int(i[1])

if "ne" in i[2]:

pre\_replacements\_dict\_3[esperanto\_Word\_before\_replacement] = [Replaced\_String, replacement\_priority\_by\_length]

i[2].remove("ne")

if "verbo\_s1" in i[2]:

for k1,k2 in verb\_suffix\_2l\_2.items():

pre\_replacements\_dict\_3[esperanto\_Word\_before\_replacement + k1] = [Replaced\_String + k2, replacement\_priority\_by\_length+len(k1)\*10000]

i[2].remove("verbo\_s1")

if "verbo\_s2" in i[2]:

for k in ["u ","i ","u","i"]:

pre\_replacements\_dict\_3[esperanto\_Word\_before\_replacement + k] = [Replaced\_String + k, replacement\_priority\_by\_length+len(k)\*10000]

i[2].remove("verbo\_s2")

if len(i[2])>=1:

for j\_item in i[2]:

j2 = j\_item.replace('/', '')

j3 = safe\_replace(j\_item, temporary\_replacements\_list\_final)\

.replace("</rt></ruby>","%%%").replace('/', '').replace("%%%","</rt></ruby>")

pre\_replacements\_dict\_3[esperanto\_Word\_before\_replacement + j2] = [Replaced\_String + j3, replacement\_priority\_by\_length+len(j2)\*10000]

else:

pre\_replacements\_dict\_3[esperanto\_Word\_before\_replacement] = [Replaced\_String, replacement\_priority\_by\_length]

except:

continue

#-------------------------------------------------------------

# (11) pre\_replacements\_dict\_3 をリスト化して、優先順位の大きい順にソート

# →「最終的に大域置換に使う置換リスト(replacements\_final\_list)」の元を作る

#-------------------------------------------------------------

pre\_replacements\_list\_1 = []

for old,new in pre\_replacements\_dict\_3.items():

# new[0] = 実際の置換後文字列, new[1] = 優先順位(int)

if isinstance(new[1], int):

pre\_replacements\_list\_1.append((old,new[0],new[1]))

pre\_replacements\_list\_2 = sorted(pre\_replacements\_list\_1, key=lambda x: x[2], reverse=True)

# remove\_redundant\_ruby\_if\_identical() で親文字とルビ文字が同じときの二重ルビを除去

pre\_replacements\_list\_3 = []

for kk in range(len(pre\_replacements\_list\_2)):

if len(pre\_replacements\_list\_2[kk][0])>=3: # 3文字以上のみを対象

# remove\_redundant\_ruby\_if\_identical: "<ruby>xxx<rt>xxx</rt></ruby>" をただの "xxx" にする

processed\_new = remove\_redundant\_ruby\_if\_identical(pre\_replacements\_list\_2[kk][1])

pre\_replacements\_list\_3.append([

pre\_replacements\_list\_2[kk][0],

processed\_new,

imported\_placeholders\_for\_global\_replacement[kk]

])

# (12) 大文字・小文字・文頭だけ大文字(capitalize) の3パターンをそれぞれ生成

# → エスペラント文中は先頭大文字などのケースもあるため

pre\_replacements\_list\_4 = []

if format\_type in ('HTML格式\_Ruby文字\_大小调整','HTML格式\_Ruby文字\_大小调整\_汉字替换','HTML格式','HTML格式\_汉字替换'):

# ルビ(HTML)系の場合、大文字化すると <ruby>や<rt>部分があるため、

# capitalize\_ruby\_and\_rt() を呼んで親文字だけ大文字化するなどの処理を行う。

for old,new,place\_holder in pre\_replacements\_list\_3:

pre\_replacements\_list\_4.append((old,new,place\_holder))

pre\_replacements\_list\_4.append((old.upper(), new.upper(), place\_holder[:-1]+'up$'))

if old.startswith(' '):

pre\_replacements\_list\_4.append((old[0] + old[1:].capitalize(), new[0] + capitalize\_ruby\_and\_rt(new[1:]), place\_holder[:-1]+'cap$'))

else:

pre\_replacements\_list\_4.append((old.capitalize(), capitalize\_ruby\_and\_rt(new), place\_holder[:-1]+'cap$'))

elif format\_type in ('括弧(号)格式', '括弧(号)格式\_汉字替换'):

# 括弧形式の場合はrubyタグではなく単なる文字列なので

# capitalize() で単純に先頭大文字化

for old,new,place\_holder in pre\_replacements\_list\_3:

pre\_replacements\_list\_4.append((old,new,place\_holder))

pre\_replacements\_list\_4.append((old.upper(), new.upper(), place\_holder[:-1]+'up$'))

if old[0]==' ':

pre\_replacements\_list\_4.append((old[0] + old[1:].capitalize(), new[0] + new[1:].capitalize(), place\_holder[:-1]+'cap$'))

else:

pre\_replacements\_list\_4.append((old.capitalize(), new.capitalize(), place\_holder[:-1]+'cap$'))

elif format\_type in ('替换后文字列のみ(仅)保留(简单替换)'):

# 単純置換の場合

for old,new,place\_holder in pre\_replacements\_list\_3:

pre\_replacements\_list\_4.append((old,new,place\_holder))

pre\_replacements\_list\_4.append((old.upper(), new.upper(), place\_holder[:-1]+'up$'))

if old[0]==' ':

pre\_replacements\_list\_4.append((old[0] + old[1:].capitalize(), new[0] + new[1:].capitalize(), place\_holder[:-1]+'cap$'))

else:

pre\_replacements\_list\_4.append((old.capitalize(), new.capitalize(), place\_holder[:-1]+'cap$'))

# (13) ここでいよいよ "replacements\_final\_list" を構築

# (old, new, placeholder) のタプルをまとめる。

replacements\_final\_list = []

for old, new, place\_holder in pre\_replacements\_list\_4:

modified\_placeholder = place\_holder

# 置換対象が空白で始/終している場合、その空白をplaceholderに反映

if old.startswith(' '):

modified\_placeholder = ' ' + modified\_placeholder

if not new.startswith(' '):

new = ' ' + new

if old.endswith(' '):

modified\_placeholder = modified\_placeholder + ' '

if not new.endswith(' '):

new = new + ' '

replacements\_final\_list.append((old, new, modified\_placeholder))

#-------------------------------------------------------------

# (14) 二文字词根替换用のリスト(全域とは別)を生成

# suffix\_2char\_roots / prefix\_2char\_roots / standalone\_2char\_roots など

#-------------------------------------------------------------

replacements\_list\_for\_suffix\_2char\_roots = []

for i in range(len(suffix\_2char\_roots)):

replaced\_suffix = remove\_redundant\_ruby\_if\_identical(safe\_replace(suffix\_2char\_roots[i], temporary\_replacements\_list\_final))

replacements\_list\_for\_suffix\_2char\_roots.append([

"$"+suffix\_2char\_roots[i],

"$"+replaced\_suffix,

"$"+imported\_placeholders\_for\_2char\_replacement[i]

])

replacements\_list\_for\_suffix\_2char\_roots.append([

"$"+suffix\_2char\_roots[i].upper(),

"$"+replaced\_suffix.upper(),

"$"+imported\_placeholders\_for\_2char\_replacement[i][:-1]+'up$'

])

replacements\_list\_for\_suffix\_2char\_roots.append([

"$"+suffix\_2char\_roots[i].capitalize(),

"$"+capitalize\_ruby\_and\_rt(replaced\_suffix),

"$"+imported\_placeholders\_for\_2char\_replacement[i][:-1]+'cap$'

])

replacements\_list\_for\_prefix\_2char\_roots = []

for i in range(len(prefix\_2char\_roots)):

replaced\_prefix = remove\_redundant\_ruby\_if\_identical(safe\_replace(prefix\_2char\_roots[i], temporary\_replacements\_list\_final))

replacements\_list\_for\_prefix\_2char\_roots.append([

prefix\_2char\_roots[i]+"$",

replaced\_prefix+"$",

imported\_placeholders\_for\_2char\_replacement[i+1000]+"$"

])

replacements\_list\_for\_prefix\_2char\_roots.append([

prefix\_2char\_roots[i].upper()+"$",

replaced\_prefix.upper()+"$",

imported\_placeholders\_for\_2char\_replacement[i+1000][:-1]+'up$'+"$"

])

replacements\_list\_for\_prefix\_2char\_roots.append([

prefix\_2char\_roots[i].capitalize()+"$",

capitalize\_ruby\_and\_rt(replaced\_prefix)+"$",

imported\_placeholders\_for\_2char\_replacement[i+1000][:-1]+'cap$'+"$"

])

replacements\_list\_for\_standalone\_2char\_roots = []

for i in range(len(standalone\_2char\_roots)):

replaced\_standalone = remove\_redundant\_ruby\_if\_identical(safe\_replace(standalone\_2char\_roots[i], temporary\_replacements\_list\_final))

replacements\_list\_for\_standalone\_2char\_roots.append([

" "+standalone\_2char\_roots[i]+" ",

" "+replaced\_standalone+" ",

" "+imported\_placeholders\_for\_2char\_replacement[i+2000]+" "

])

replacements\_list\_for\_standalone\_2char\_roots.append([

" "+standalone\_2char\_roots[i].upper()+" ",

" "+replaced\_standalone.upper()+" ",

" "+imported\_placeholders\_for\_2char\_replacement[i+2000][:-1]+'up$'+" "

])

replacements\_list\_for\_standalone\_2char\_roots.append([

" "+standalone\_2char\_roots[i].capitalize()+" ",

" "+capitalize\_ruby\_and\_rt(replaced\_standalone)+" ",

" "+imported\_placeholders\_for\_2char\_replacement[i+2000][:-1]+'cap$'+" "

])

replacements\_list\_for\_2char = (

replacements\_list\_for\_standalone\_2char\_roots

+ replacements\_list\_for\_suffix\_2char\_roots

+ replacements\_list\_for\_prefix\_2char\_roots

)

#-------------------------------------------------------------

# (15) 局所的な文字列(漢字)置換用のリストを作成

# これは "%"や"@"で囲まれた部分だけ置換したいときに使う想定。

# CSV\_data\_imported にある(語根,訳)だけを対象とする。

#-------------------------------------------------------------

pre\_replacements\_list\_for\_localized\_string\_1 = []

for \_, (E\_root, hanzi\_or\_meaning) in CSV\_data\_imported.iterrows():

if pd.notna(E\_root) and pd.notna(hanzi\_or\_meaning) and '#' not in E\_root and (E\_root != '') and (hanzi\_or\_meaning != ''):

if E\_root == hanzi\_or\_meaning:

# E\_rootと翻訳が同じ場合(稀だが)でも、一応3パターン(大文字/先頭大文字含む)追加

pre\_replacements\_list\_for\_localized\_string\_1.append([E\_root, hanzi\_or\_meaning, len(E\_root)])

pre\_replacements\_list\_for\_localized\_string\_1.append([E\_root.upper(), hanzi\_or\_meaning.upper(), len(E\_root)])

pre\_replacements\_list\_for\_localized\_string\_1.append([E\_root.capitalize(), hanzi\_or\_meaning.capitalize(), len(E\_root)])

else:

# それ以外は output\_format() を通す

pre\_replacements\_list\_for\_localized\_string\_1.append([

E\_root,

output\_format(E\_root, hanzi\_or\_meaning, format\_type, char\_widths\_dict),

len(E\_root)

])

pre\_replacements\_list\_for\_localized\_string\_1.append([

E\_root.upper(),

output\_format(E\_root.upper(), hanzi\_or\_meaning.upper(), format\_type, char\_widths\_dict),

len(E\_root)

])

pre\_replacements\_list\_for\_localized\_string\_1.append([

E\_root.capitalize(),

output\_format(E\_root.capitalize(), hanzi\_or\_meaning.capitalize(), format\_type, char\_widths\_dict),

len(E\_root)

])

# 長い語根を先に置換できるようソート(文字数多い順)

pre\_replacements\_list\_for\_localized\_string\_2 = sorted(pre\_replacements\_list\_for\_localized\_string\_1, key=lambda x: x[2], reverse=True)

replacements\_list\_for\_localized\_string = []

for kk in range(len(pre\_replacements\_list\_for\_localized\_string\_2)):

replacements\_list\_for\_localized\_string.append([

pre\_replacements\_list\_for\_localized\_string\_2[kk][0],

pre\_replacements\_list\_for\_localized\_string\_2[kk][1],

imported\_placeholders\_for\_local\_replacement[kk]

])

#=============================================================

# (16) 最後に3種類のリストを JSON 化してダウンロードできる形にする

# - 全域替换用のリスト(列表)型配列 → replacements\_final\_list

# - 二文字词根替换用のリスト(列表)型配列 → replacements\_list\_for\_2char

# - 局部文字替换用のリスト(列表)型配列 → replacements\_list\_for\_localized\_string

#=============================================================

combined\_data = {}

combined\_data["全域替换用のリスト(列表)型配列(replacements\_final\_list)"] = replacements\_final\_list

combined\_data["二文字词根替换用のリスト(列表)型配列(replacements\_list\_for\_2char)"] = replacements\_list\_for\_2char

combined\_data["局部文字替换用のリスト(列表)型配列(replacements\_list\_for\_localized\_string)"] = replacements\_list\_for\_localized\_string

# JSON文字列にダンプし、ダウンロードボタンを生成

download\_data = json.dumps(combined\_data, ensure\_ascii=False, indent=2)

st.success("Elenco di sostituzione generato con successo!")

st.download\_button(

label="Scarica la lista finale di sostituzione (unione di 3 file JSON)",

data=download\_data,

file\_name="ListaDefinitivaDiSostituzione\_UnioneDi3FileJSON.json",

mime='application/json'

)

## esp\_text\_replacement\_module.py(3つ目)

"""

このモジュールは「エスペラント文章の文字列(漢字)置換」を包括的に扱うツール集です。

主な機能：

1. エスペラント独自の文字形式（ĉ, ĝなど）への変換 → convert\_to\_circumflex

2. 特殊な半角スペースの統一（ASCIIスペースに） → unify\_halfwidth\_spaces

3. (現在不要になった) HTMLルビ付与関数 → wrap\_text\_with\_ruby (コメントのみ)

4. %や@で囲まれたテキストのスキップ・局所変換 → (create\_replacements\_list\_for\_...)

5. 大域的なプレースホルダー置換 → safe\_replace

6. それらをまとめて実行する複合置換関数 → orchestrate\_comprehensive\_esperanto\_text\_replacement

7. multiprocessing を用いた行単位の並列実行 → parallel\_process / process\_segment

"""

import re

import json

from typing import List, Tuple, Dict

import multiprocessing

# ================================

# 1) エスペラント文字変換用の辞書

# ================================

# それぞれ (x表記 → ĉ) や (ĉ → c^)など、様々なマッピングを辞書にしている

x\_to\_circumflex = {

'cx': 'ĉ', 'gx': 'ĝ', 'hx': 'ĥ', 'jx': 'ĵ', 'sx': 'ŝ', 'ux': 'ŭ',

'Cx': 'Ĉ', 'Gx': 'Ĝ', 'Hx': 'Ĥ', 'Jx': 'Ĵ', 'Sx': 'Ŝ', 'Ux': 'Ŭ'

}

circumflex\_to\_x = {

'ĉ': 'cx', 'ĝ': 'gx', 'ĥ': 'hx', 'ĵ': 'jx', 'ŝ': 'sx', 'ŭ': 'ux',

'Ĉ': 'Cx', 'Ĝ': 'Gx', 'Ĥ': 'Hx', 'Ĵ': 'Jx', 'Ŝ': 'Sx', 'Ŭ': 'Ux'

}

x\_to\_hat = {

'cx': 'c^', 'gx': 'g^', 'hx': 'h^', 'jx': 'j^', 'sx': 's^', 'ux': 'u^',

'Cx': 'C^', 'Gx': 'G^', 'Hx': 'H^', 'Jx': 'J^', 'Sx': 'S^', 'Ux': 'U^'

}

hat\_to\_x = {

'c^': 'cx', 'g^': 'gx', 'h^': 'hx', 'j^': 'jx', 's^': 'sx', 'u^': 'ux',

'C^': 'Cx', 'G^': 'Gx', 'H^': 'Hx', 'J^': 'Jx', 'S^': 'Sx', 'U^': 'Ux'

}

hat\_to\_circumflex = {

'c^': 'ĉ', 'g^': 'ĝ', 'h^': 'ĥ', 'j^': 'ĵ', 's^': 'ŝ', 'u^': 'ŭ',

'C^': 'Ĉ', 'G^': 'Ĝ', 'H^': 'Ĥ', 'J^': 'Ĵ', 'S^': 'Ŝ', 'U^': 'Ŭ'

}

circumflex\_to\_hat = {

'ĉ': 'c^', 'ĝ': 'g^', 'ĥ': 'h^', 'ĵ': 'j^', 'ŝ': 's^', 'ŭ': 'u^',

'Ĉ': 'C^', 'Ĝ': 'G^', 'Ĥ': 'H^', 'Ĵ': 'J^', 'Ŝ': 'S^', 'Ŭ': 'U^'

}

# ================================

# 2) 基本の文字形式変換関数

# ================================

def replace\_esperanto\_chars(text, char\_dict: Dict[str, str]) -> str:

# char\_dict に含まれるペア (original\_char, converted\_char) ごとに

# text.replace() していく

for original\_char, converted\_char in char\_dict.items():

text = text.replace(original\_char, converted\_char)

return text

def convert\_to\_circumflex(text: str) -> str:

"""

テキストを字上符形式（ĉ, ĝ, ĥ, ĵ, ŝ, ŭなど）に統一します。

1. hat\_to\_circumflex: c^ → ĉ

2. x\_to\_circumflex: cx → ĉ

"""

text = replace\_esperanto\_chars(text, hat\_to\_circumflex)

text = replace\_esperanto\_chars(text, x\_to\_circumflex)

return text

def unify\_halfwidth\_spaces(text: str) -> str:

"""

全角スペース(U+3000)は変更せず、半角スペースと視覚的に区別がつきにくい空白文字を

ASCII半角スペース(U+0020)に統一する。

"""

pattern = r"[\u00A0\u2002\u2003\u2004\u2005\u2006\u2007\u2008\u2009\u200A]"

return re.sub(pattern, " ", text)

# ================================

# 3) (HTMLルビタグの補助関数)

# (現状不要とされている)

# ================================

# ================================

# 4) 占位符(placeholder)関連

# ================================

def safe\_replace(text: str, replacements: List[Tuple[str, str, str]]) -> str:

"""

(old, new, placeholder) のリストを受け取り、

text中の old → placeholder → new の段階置換を行う。

"""

valid\_replacements = {}

# まず old→placeholder

for old, new, placeholder in replacements:

if old in text:

text = text.replace(old, placeholder)

valid\_replacements[placeholder] = new

# 次に placeholder→new

for placeholder, new in valid\_replacements.items():

text = text.replace(placeholder, new)

return text

def import\_placeholders(filename: str) -> List[str]:

"""

プレースホルダを行単位で読み込むだけの関数

"""

with open(filename, 'r') as file:

placeholders = [line.strip() for line in file if line.strip()]

return placeholders

# '%' で囲まれた箇所をスキップするための正規表現

PERCENT\_PATTERN = re.compile(r'%(.{1,50}?)%')

def find\_percent\_enclosed\_strings\_for\_skipping\_replacement(text: str) -> List[str]:

"""'%foo%' の形を全て抽出。50文字以内に限定。"""

matches = []

used\_indices = set()

for match in PERCENT\_PATTERN.finditer(text):

start, end = match.span()

if start not in used\_indices and end-2 not in used\_indices:

matches.append(match.group(1))

used\_indices.update(range(start, end))

return matches

def create\_replacements\_list\_for\_intact\_parts(text: str, placeholders: List[str]) -> List[Tuple[str, str]]:

"""

'%xxx%' で囲まれた箇所を検出し、

( '%xxx%', placeholder ) という形で対応させるリストを作る

"""

matches = find\_percent\_enclosed\_strings\_for\_skipping\_replacement(text)

replacements\_list\_for\_intact\_parts = []

for i, match in enumerate(matches):

if i < len(placeholders):

replacements\_list\_for\_intact\_parts.append([f"%{match}%", placeholders[i]])

else:

break

return replacements\_list\_for\_intact\_parts

# '@' で囲まれた箇所を局所置換するための正規表現

AT\_PATTERN = re.compile(r'@(.{1,18}?)@')

def find\_at\_enclosed\_strings\_for\_localized\_replacement(text: str) -> List[str]:

"""'@foo@' の形を全て抽出。18文字以内に限定。"""

matches = []

used\_indices = set()

for match in AT\_PATTERN.finditer(text):

start, end = match.span()

if start not in used\_indices and end-2 not in used\_indices:

matches.append(match.group(1))

used\_indices.update(range(start, end))

return matches

def create\_replacements\_list\_for\_localized\_replacement(text, placeholders: List[str],

replacements\_list\_for\_localized\_string: List[Tuple[str, str, str]]

) -> List[List[str]]:

"""

'@xxx@' で囲まれた箇所を検出し、

その内部文字列 'xxx' を replacements\_list\_for\_localized\_string で置換した結果を

placeholder に置き換える。

"""

matches = find\_at\_enclosed\_strings\_for\_localized\_replacement(text)

tmp\_list = []

for i, match in enumerate(matches):

if i < len(placeholders):

replaced\_match = safe\_replace(match, replacements\_list\_for\_localized\_string)

tmp\_list.append([f"@{match}@", placeholders[i], replaced\_match])

else:

break

return tmp\_list

# ================================

# 5) メインの複合文字列(漢字)置換関数

# ================================

def orchestrate\_comprehensive\_esperanto\_text\_replacement(

text,

placeholders\_for\_skipping\_replacements: List[str],

replacements\_list\_for\_localized\_string: List[Tuple[str, str, str]],

placeholders\_for\_localized\_replacement: List[str],

replacements\_final\_list: List[Tuple[str, str, str]],

replacements\_list\_for\_2char: List[Tuple[str, str, str]],

format\_type: str

) -> str:

"""

複数の変換ルールに従ってエスペラント文を文字列(漢字)置換するメイン関数。

1) 空白の正規化 → 2) エスペラント文字(ĉ等)の字上符形式統一

3) %で囲まれた部分をスキップ

4) @で囲まれた部分を局所置換

5) 大域置換

6) 2文字語根の置換を2回

7) プレースホルダ復元

8) HTML形式が指定なら追加整形

"""

# 1, 2) 空白の正規化 + エスペラント字上符への変換

text = unify\_halfwidth\_spaces(text)

text = convert\_to\_circumflex(text)

# 3) %...% スキップ部の一時置換

replacements\_list\_for\_intact\_parts = create\_replacements\_list\_for\_intact\_parts(text, placeholders\_for\_skipping\_replacements)

# 文字数長い順にsort (衝突を避けるため)

sorted\_replacements\_list\_for\_intact\_parts = sorted(replacements\_list\_for\_intact\_parts, key=lambda x: len(x[0]), reverse=True)

for original, place\_holder\_ in sorted\_replacements\_list\_for\_intact\_parts:

text = text.replace(original, place\_holder\_)

# 4) @...@ 局所置換

tmp\_replacements\_list\_for\_localized\_string\_2 = create\_replacements\_list\_for\_localized\_replacement(

text, placeholders\_for\_localized\_replacement, replacements\_list\_for\_localized\_string

)

sorted\_replacements\_list\_for\_localized\_string = sorted(tmp\_replacements\_list\_for\_localized\_string\_2, key=lambda x: len(x[0]), reverse=True)

for original, place\_holder\_, replaced\_original in sorted\_replacements\_list\_for\_localized\_string:

text = text.replace(original, place\_holder\_)

# 5) 大域置換 (old, new, placeholder)

valid\_replacements = {}

for old, new, placeholder in replacements\_final\_list:

if old in text:

text = text.replace(old, placeholder)

valid\_replacements[placeholder] = new

# 6) 2文字語根置換(2回)

valid\_replacements\_for\_2char\_roots = {}

for old, new, placeholder in replacements\_list\_for\_2char:

if old in text:

text = text.replace(old, placeholder)

valid\_replacements\_for\_2char\_roots[placeholder] = new

valid\_replacements\_for\_2char\_roots\_2 = {}

for old, new, placeholder in replacements\_list\_for\_2char:

if old in text:

place\_holder\_second = "!" + placeholder + "!"

text = text.replace(old, place\_holder\_second)

valid\_replacements\_for\_2char\_roots\_2[place\_holder\_second] = new

# 7) placeholderを最終的な文字列に戻す

for place\_holder\_second, new in reversed(valid\_replacements\_for\_2char\_roots\_2.items()):

text = text.replace(place\_holder\_second, new)

for placeholder, new in reversed(valid\_replacements\_for\_2char\_roots.items()):

text = text.replace(placeholder, new)

for placeholder, new in valid\_replacements.items():

text = text.replace(placeholder, new)

# 局所(@)・スキップ(%) の復元

for original, place\_holder\_, replaced\_original in sorted\_replacements\_list\_for\_localized\_string:

text = text.replace(place\_holder\_, replaced\_original.replace("@",""))

for original, place\_holder\_ in sorted\_replacements\_list\_for\_intact\_parts:

text = text.replace(place\_holder\_, original.replace("%",""))

# 8) HTML形式であれば、改行を <br> に変換 + スペースを &nbsp; に置換

if "HTML" in format\_type:

text = text.replace("\n", "<br>\n")

# text = wrap\_text\_with\_ruby(text, chunk\_size=10) # (過去の関数/不要)

text = re.sub(r" ", "&nbsp;&nbsp;&nbsp;", text) # 3つ以上の空白を変換

text = re.sub(r" ", "&nbsp;&nbsp;", text) # 2つ以上の空白を変換

return text

# ================================

# 6) multiprocessing 関連

# ================================

def process\_segment(

lines: List[str],

placeholders\_for\_skipping\_replacements: List[str],

replacements\_list\_for\_localized\_string: List[Tuple[str, str, str]],

placeholders\_for\_localized\_replacement: List[str],

replacements\_final\_list: List[Tuple[str, str, str]],

replacements\_list\_for\_2char: List[Tuple[str, str, str]],

format\_type: str

) -> str:

"""

multiprocessing用の下請け関数。

lines (文字列リスト) を結合してから orchestrate\_comprehensive\_esperanto\_text\_replacement を実行。

"""

segment = ''.join(lines)

result = orchestrate\_comprehensive\_esperanto\_text\_replacement(

segment,

placeholders\_for\_skipping\_replacements,

replacements\_list\_for\_localized\_string,

placeholders\_for\_localized\_replacement,

replacements\_final\_list,

replacements\_list\_for\_2char,

format\_type

)

return result

def parallel\_process(

text: str,

num\_processes: int,

placeholders\_for\_skipping\_replacements: List[str],

replacements\_list\_for\_localized\_string: List[Tuple[str, str, str]],

placeholders\_for\_localized\_replacement: List[str],

replacements\_final\_list: List[Tuple[str, str, str]],

replacements\_list\_for\_2char: List[Tuple[str, str, str]],

format\_type: str

) -> str:

"""

与えられた text を行単位で分割し、process\_segment を

マルチプロセスで並列実行した結果を結合する。

"""

if num\_processes <= 1:

# シングルコアで直接orchestrate\_comprehensive\_esperanto\_text\_replacementを呼ぶ

return orchestrate\_comprehensive\_esperanto\_text\_replacement(

text,

placeholders\_for\_skipping\_replacements,

replacements\_list\_for\_localized\_string,

placeholders\_for\_localized\_replacement,

replacements\_final\_list,

replacements\_list\_for\_2char,

format\_type

)

# 行ごとに分割 (改行込み)

lines = re.findall(r'.\*?\n|.+$', text)

num\_lines = len(lines)

if num\_lines <= 1:

# 行数が1以下なら並列化しても意味ないのでシングルで

return orchestrate\_comprehensive\_esperanto\_text\_replacement(

text,

placeholders\_for\_skipping\_replacements,

replacements\_list\_for\_localized\_string,

placeholders\_for\_localized\_replacement,

replacements\_final\_list,

replacements\_list\_for\_2char,

format\_type

)

lines\_per\_process = max(num\_lines // num\_processes, 1)

ranges = [(i \* lines\_per\_process, (i + 1) \* lines\_per\_process) for i in range(num\_processes)]

# 最後のプロセスに残りを全部割り当てる

ranges[-1] = (ranges[-1][0], num\_lines)

with multiprocessing.Pool(processes=num\_processes) as pool:

results = pool.starmap(

process\_segment,

[

(

lines[start:end],

placeholders\_for\_skipping\_replacements,

replacements\_list\_for\_localized\_string,

placeholders\_for\_localized\_replacement,

replacements\_final\_list,

replacements\_list\_for\_2char,

format\_type

)

for (start, end) in ranges

]

)

return ''.join(results)

def apply\_ruby\_html\_header\_and\_footer(processed\_text: str, format\_type: str) -> str:

"""

指定された出力形式に応じて、processed\_text に対するHTMLヘッダーとフッターを適用する。

例: ルビサイズ調整用の<style> を挿入するなど。

"""

if format\_type in ('HTML格式\_Ruby文字\_大小调整','HTML格式\_Ruby文字\_大小调整\_汉字替换'):

# html形式におけるルビサイズの変更形式

ruby\_style\_head="""<!DOCTYPE html>

<html lang="ja">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>大多数の环境中で正常に运行するRuby显示功能</title>

<style>

html, body {

-webkit-text-size-adjust: 100%;

-moz-text-size-adjust: 100%;

-ms-text-size-adjust: 100%;

text-size-adjust: 100%;

}

:root {

--ruby-color: blue;

--ruby-font-size: 0.5em;

}

html {

font-size: 100%; /\* 多くのブラウザは16px相当が標準 \*/

}

.text-M\_M {

font-size: 1rem!important;

font-family: Arial, sans-serif;

line-height: 2.0 !important; /\* text-M\_Mのline-heightとrubyのline-heightは一致させる必要がある。 \*/

display: block; /\* ブロック要素として扱う \*/

position: relative;

}

/\* ▼ ルビ（フレックスでルビを上に表示） \*/

ruby {

display: inline-flex;

flex-direction: column;

align-items: center;

vertical-align: top !important;

line-height: 2.0 !important;

margin: 0 !important;

padding: 0 !important;

font-size: 1rem !important;

}

/\* ▼ 追加マイナス余白（ルビサイズ別に上書き） \*/

rt {

display: block !important;

font-size: var(--ruby-font-size);

color: var(--ruby-color);

line-height: 1.05;/\*ルビを改行するケースにおけるルビの行間\*/

text-align: center;

/\* margin-top: 0.2em !important;

transform: translateY(0.4em) !important; \*/

}

rt.XXXS\_S {

--ruby-font-size: 0.3em;

margin-top: -8.3em !important;/\* ルビの高さ位置はここで調節する。 \*/

transform: translateY(-0em) !important;

}

rt.XXS\_S {

--ruby-font-size: 0.3em;

margin-top: -7.2em !important;/\* ルビの高さ位置はここで調節する。 \*/

transform: translateY(-0em) !important;

}

rt.XS\_S {

--ruby-font-size: 0.3em;

margin-top: -6.1em !important;

transform: translateY(-0em) !important;

}

rt.S\_S {

--ruby-font-size: 0.4em;

margin-top: -4.85em !important;

transform: translateY(-0em) !important;

}

rt.M\_M {

--ruby-font-size: 0.5em;

margin-top: -4.00em !important;

transform: translateY(-0.0em) !important;

}

rt.L\_L {

--ruby-font-size: 0.6em;

margin-top: -3.55em !important;

transform: translateY(-0.0em) !important;

}

rt.XL\_L {

--ruby-font-size: 0.7em;

margin-top: -3.20em !important;

transform: translateY(-0.0em) !important;

}

rt.XXL\_L {

--ruby-font-size: 0.8em;

margin-top: -2.80em !important;

transform: translateY(-0.0em) !important;

}

</style>

</head>

<body>

<p class="text-M\_M">

"""

ruby\_style\_tail = "</p></body></html>"

elif format\_type in ('HTML格式','HTML格式\_汉字替换'):

ruby\_style\_head = """<style>

ruby rt {

color: blue;

}

</style>

"""

ruby\_style\_tail = "<br>"

else:

ruby\_style\_head = ""

ruby\_style\_tail = ""

return ruby\_style\_head + processed\_text + ruby\_style\_tail

## esp\_replacement\_json\_make\_module.py(4つ目)

"""

エスペラント文字の変換や、ルビサイズ調整、置換処理用の関数などをまとめたモジュール。

【構成】

1) 文字変換用の辞書定義 (字上符形式への変換など)

2) 基本の文字形式変換関数 (replace\_esperanto\_chars, convert\_to\_circumflex, など)

3) 文字幅計測＆<br>挿入関数 (measure\_text\_width\_Arial16, insert\_br\_at\_half\_width, insert\_br\_at\_third\_width)

4) 出力フォーマット (output\_format) 関連

5) 文字列判定・placeholder インポートなどの補助関数

6) multiprocessing 関連の並列置換用関数 (process\_chunk\_for\_pre\_replacements, parallel\_build\_pre\_replacements\_dict)

"""

import re

import json

import multiprocessing

import pandas as pd

import os

from typing import List, Dict, Tuple, Optional

#=================================================================

# 1) エスペラント文字変換用の辞書 (同様のものが他のファイルにもある)

#=================================================================

x\_to\_circumflex = {'cx': 'ĉ', 'gx': 'ĝ', 'hx': 'ĥ', 'jx': 'ĵ', 'sx': 'ŝ', 'ux': 'ŭ',

'Cx': 'Ĉ', 'Gx': 'Ĝ', 'Hx': 'Ĥ', 'Jx': 'Ĵ', 'Sx': 'Ŝ', 'Ux': 'Ŭ'}

circumflex\_to\_x = {'ĉ': 'cx', 'ĝ': 'gx', 'ĥ': 'hx', 'ĵ': 'jx', 'ŝ': 'sx', 'ŭ': 'ux',

'Ĉ': 'Cx', 'Ĝ': 'Gx', 'Ĥ': 'Hx', 'Ĵ': 'Jx', 'Ŝ': 'Sx', 'Ŭ': 'Ux'}

x\_to\_hat = {'cx': 'c^', 'gx': 'g^', 'hx': 'h^', 'jx': 'j^', 'sx': 's^', 'ux': 'u^',

'Cx': 'C^', 'Gx': 'G^', 'Hx': 'H^', 'Jx': 'J^', 'Sx': 'S^', 'Ux': 'U^'}

hat\_to\_x = {'c^': 'cx', 'g^': 'gx', 'h^': 'hx', 'j^': 'jx', 's^': 'sx', 'u^': 'ux',

'C^': 'Cx', 'G^': 'Gx', 'H^': 'Hx', 'J^': 'Jx', 'S^': 'Sx', 'U^': 'Ux'}

hat\_to\_circumflex = {'c^': 'ĉ', 'g^': 'ĝ', 'h^': 'ĥ', 'j^': 'ĵ', 's^': 'ŝ', 'u^': 'ŭ',

'C^': 'Ĉ', 'G^': 'Ĝ', 'H^': 'Ĥ', 'J^': 'Ĵ', 'S^': 'Ŝ', 'U^': 'Ŭ'}

circumflex\_to\_hat = {'ĉ': 'c^', 'ĝ': 'g^', 'ĥ': 'h^', 'ĵ': 'j^', 'ŝ': 's^', 'ŭ': 'u^',

'Ĉ': 'C^', 'Ĝ': 'G^', 'Ĥ': 'H^', 'Ĵ': 'J^', 'Ŝ': 'S^', 'Ŭ': 'U^'}

#=================================================================

# 2) 基本の文字形式変換関数

#=================================================================

def replace\_esperanto\_chars(text, char\_dict: Dict[str, str]) -> str:

for original\_char, converted\_char in char\_dict.items():

text = text.replace(original\_char, converted\_char)

return text

def convert\_to\_circumflex(text: str) -> str:

# c^, g^... → ĉ, ĝ... および cx, gx... → ĉ, ĝ... に変換

text = replace\_esperanto\_chars(text, hat\_to\_circumflex)

text = replace\_esperanto\_chars(text, x\_to\_circumflex)

return text

#=================================================================

# 3) 文字幅計測 & <br> 挿入関数

#=================================================================

def measure\_text\_width\_Arial16(text, char\_widths\_dict: Dict[str, int]) -> int:

"""

JSONで読み込んだ {文字: 幅(px)} の辞書を使い、

text の合計幅を算出する

"""

total\_width = 0

for ch in text:

char\_width = char\_widths\_dict.get(ch, 8)

total\_width += char\_width

return total\_width

def insert\_br\_at\_half\_width(text, char\_widths\_dict: Dict[str, int]) -> str:

"""

文字列幅が半分を超えたら <br> を入れる

"""

total\_width = measure\_text\_width\_Arial16(text, char\_widths\_dict)

half\_width = total\_width / 2

current\_width = 0

insert\_index = None

for i, ch in enumerate(text):

char\_width = char\_widths\_dict.get(ch, 8)

current\_width += char\_width

if current\_width >= half\_width:

insert\_index = i + 1

break

if insert\_index is not None:

result = text[:insert\_index] + "<br>" + text[insert\_index:]

else:

result = text

return result

def insert\_br\_at\_third\_width(text, char\_widths\_dict: Dict[str, int]) -> str:

"""

文字列幅を三等分し、1/3 と 2/3 の位置に <br> を挿入する

"""

total\_width = measure\_text\_width\_Arial16(text, char\_widths\_dict)

third\_width = total\_width / 3

thresholds = [third\_width, third\_width\*2]

current\_width = 0

insert\_indices = []

found\_first = False

for i, ch in enumerate(text):

char\_width = char\_widths\_dict.get(ch, 8)

current\_width += char\_width

if not found\_first and current\_width >= thresholds[0]:

insert\_indices.append(i+1)

found\_first = True

elif found\_first and current\_width >= thresholds[1]:

insert\_indices.append(i+1)

break

result = text

for idx in reversed(insert\_indices):

result = result[:idx] + "<br>" + result[idx:]

return result

#=================================================================

# 4) 出力フォーマット (HTML/括弧形式等)

#=================================================================

def output\_format(main\_text, ruby\_content, format\_type, char\_widths\_dict):

"""

エスペラント語根(main\_text) と それに対応する訳/漢字(ruby\_content) を

指定の format\_type で繋ぎ合わせる

"""

if format\_type == 'HTML格式\_Ruby文字\_大小调整':

width\_ruby = measure\_text\_width\_Arial16(ruby\_content, char\_widths\_dict)

width\_main = measure\_text\_width\_Arial16(main\_text, char\_widths\_dict)

ratio\_1 = width\_ruby / width\_main

if ratio\_1 > 6:

return f'<ruby>{main\_text}<rt class="XXXS\_S">{insert\_br\_at\_third\_width(ruby\_content, char\_widths\_dict)}</rt></ruby>'

elif ratio\_1 > (9/3):

return f'<ruby>{main\_text}<rt class="XXS\_S">{insert\_br\_at\_half\_width(ruby\_content, char\_widths\_dict)}</rt></ruby>'

elif ratio\_1 > (9/4):

return f'<ruby>{main\_text}<rt class="XS\_S">{ruby\_content}</rt></ruby>'

elif ratio\_1 > (9/5):

return f'<ruby>{main\_text}<rt class="S\_S">{ruby\_content}</rt></ruby>'

elif ratio\_1 > (9/6):

return f'<ruby>{main\_text}<rt class="M\_M">{ruby\_content}</rt></ruby>'

elif ratio\_1 > (9/7):

return f'<ruby>{main\_text}<rt class="L\_L">{ruby\_content}</rt></ruby>'

elif ratio\_1 > (9/8):

return f'<ruby>{main\_text}<rt class="XL\_L">{ruby\_content}</rt></ruby>'

else:

return f'<ruby>{main\_text}<rt class="XXL\_L">{ruby\_content}</rt></ruby>'

elif format\_type == 'HTML格式\_Ruby文字\_大小调整\_汉字替换':

# main と ruby の立場を逆転したような形式

width\_ruby = measure\_text\_width\_Arial16(ruby\_content, char\_widths\_dict)

width\_main = measure\_text\_width\_Arial16(main\_text, char\_widths\_dict)

ratio\_2 = width\_main / width\_ruby

if ratio\_2 > 6:

return f'<ruby>{ruby\_content}<rt class="XXXS\_S">{insert\_br\_at\_third\_width(main\_text, char\_widths\_dict)}</rt></ruby>'

elif ratio\_2 > (9/3):

return f'<ruby>{ruby\_content}<rt class="XXS\_S">{insert\_br\_at\_half\_width(main\_text, char\_widths\_dict)}</rt></ruby>'

elif ratio\_2 > (9/4):

return f'<ruby>{ruby\_content}<rt class="XS\_S">{main\_text}</rt></ruby>'

elif ratio\_2 > (9/5):

return f'<ruby>{ruby\_content}<rt class="S\_S">{main\_text}</rt></ruby>'

elif ratio\_2 > (9/6):

return f'<ruby>{ruby\_content}<rt class="M\_M">{main\_text}</rt></ruby>'

elif ratio\_2 > (9/7):

return f'<ruby>{ruby\_content}<rt class="L\_L">{main\_text}</rt></ruby>'

elif ratio\_2 > (9/8):

return f'<ruby>{ruby\_content}<rt class="XL\_L">{main\_text}</rt></ruby>'

else:

return f'<ruby>{ruby\_content}<rt class="XXL\_L">{main\_text}</rt></ruby>'

elif format\_type == 'HTML格式':

return f'<ruby>{main\_text}<rt>{ruby\_content}</rt></ruby>'

elif format\_type == 'HTML格式\_汉字替换':

return f'<ruby>{ruby\_content}<rt>{main\_text}</rt></ruby>'

elif format\_type == '括弧(号)格式':

return f'{main\_text}({ruby\_content})'

elif format\_type == '括弧(号)格式\_汉字替换':

return f'{ruby\_content}({main\_text})'

elif format\_type == '替换后文字列のみ(仅)保留(简单替换)':

return f'{ruby\_content}'

#=================================================================

# 5) 文字列判定・placeholder インポート等の補助関数

#=================================================================

def contains\_digit(s: str) -> bool:

return any(char.isdigit() for char in s)

def import\_placeholders(filename: str) -> List[str]:

with open(filename, 'r') as file:

placeholders = [line.strip() for line in file if line.strip()]

return placeholders

# 以下のパターンはHTMLルビを大文字化するためのもの(一部の拡張)

RUBY\_PATTERN = re.compile(

r'^'

r'(.\*?)'

r'(<ruby>)'

r'([^<]+)'

r'(<rt[^>]\*>)'

r'([^<]\*?(?:<br>[^<]\*?){0,2})'

r'(</rt>)'

r'(</ruby>)?'

r'(.\*)'

r'$'

)

def capitalize\_ruby\_and\_rt(text: str) -> str:

"""

<ruby>〜</ruby> の親文字列 / ルビ文字列を大文字化する例。

"""

def replacer(match):

g1 = match.group(1)

g2 = match.group(2)

g3 = match.group(3)

g4 = match.group(4)

g5 = match.group(5)

g6 = match.group(6)

g7 = match.group(7)

g8 = match.group(8)

if g1.strip():

return g1.capitalize() + g2 + g3 + g4 + g5 + g6 + (g7 if g7 else '') + g8

else:

parent\_text = g3.capitalize()

rt\_text = g5.capitalize()

return g1 + g2 + parent\_text + g4 + rt\_text + g6 + (g7 if g7 else '') + g8

replaced\_text = RUBY\_PATTERN.sub(replacer, text)

if replaced\_text == text:

replaced\_text = text.capitalize()

return replaced\_text

#=================================================================

# 6) multiprocessing 関連

#=================================================================

def safe\_replace(text: str, replacements: List[Tuple[str, str, str]]) -> str:

"""

こちらにも safe\_replace が定義されている (同名関数)

(mainページ用のesp\_text\_replacement\_module.pyと重複しているが別ファイル)

"""

valid\_replacements = {}

for old, new, placeholder in replacements:

if old in text:

text = text.replace(old, placeholder)

valid\_replacements[placeholder] = new

for placeholder, new in valid\_replacements.items():

text = text.replace(placeholder, new)

return text

def process\_chunk\_for\_pre\_replacements(

chunk: List[List[str]],

replacements: List[Tuple[str, str, str]]

) -> Dict[str, List[str]]:

"""

chunk: [[E\_root, pos], ...] の部分リスト

safe\_replace による置換結果を { E\_root: [replaced\_stem, pos], ... } の形で返す

"""

local\_dict = {}

for item in chunk:

if len(item) != 2:

continue

E\_root, pos\_info = item

if len(E\_root) < 2:

continue

if E\_root in local\_dict:

replaced\_stem, existing\_pos\_str = local\_dict[E\_root]

existing\_pos\_list = existing\_pos\_str.split(',')

if pos\_info not in existing\_pos\_list:

existing\_pos\_list.append(pos\_info)

merged\_pos\_str = ",".join(existing\_pos\_list)

local\_dict[E\_root] = [replaced\_stem, merged\_pos\_str]

else:

replaced = safe\_replace(E\_root, replacements)

local\_dict[E\_root] = [replaced, pos\_info]

return local\_dict

def parallel\_build\_pre\_replacements\_dict(

E\_stem\_with\_Part\_Of\_Speech\_list: List[List[str]],

replacements: List[Tuple[str, str, str]],

num\_processes: int = 4

) -> Dict[str, List[str]]:

"""

データを num\_processes 個に分割し、process\_chunk\_for\_pre\_replacements を並列実行

最終的に辞書をマージして返す。

"""

total\_len = len(E\_stem\_with\_Part\_Of\_Speech\_list)

if total\_len == 0:

return {}

chunk\_size = -(-total\_len // num\_processes)

chunks = []

start\_index = 0

for \_ in range(num\_processes):

end\_index = min(start\_index + chunk\_size, total\_len)

chunk = E\_stem\_with\_Part\_Of\_Speech\_list[start\_index:end\_index]

chunks.append(chunk)

start\_index = end\_index

if start\_index >= total\_len:

break

with multiprocessing.Pool(num\_processes) as pool:

partial\_dicts = pool.starmap(

process\_chunk\_for\_pre\_replacements,

[(chunk, replacements) for chunk in chunks]

)

merged\_dict = {}

for partial\_d in partial\_dicts:

for E\_root, val in partial\_d.items():

replaced\_stem, pos\_str = val

if E\_root not in merged\_dict:

merged\_dict[E\_root] = [replaced\_stem, pos\_str]

else:

existing\_replaced\_stem, existing\_pos\_str = merged\_dict[E\_root]

existing\_pos\_list = existing\_pos\_str.split(',')

new\_pos\_list = pos\_str.split(',')

pos\_merged = list(set(existing\_pos\_list) | set(new\_pos\_list))

pos\_merged\_str = ",".join(sorted(pos\_merged))

merged\_dict[E\_root] = [existing\_replaced\_stem, pos\_merged\_str]

return merged\_dict

#=================================================================

# 追加(202502):

# 同一ルビが (ルビ付けした結果) 重複している場合に削除する関数

#=================================================================

IDENTICAL\_RUBY\_PATTERN = re.compile(r'<ruby>([^<]+)<rt class="XXL\_L">([^<]+)</rt></ruby>')

def remove\_redundant\_ruby\_if\_identical(text: str) -> str:

"""

<ruby>xxx<rt class="XXL\_L">xxx</rt></ruby> のように、

親文字列とルビ文字列が完全に同一の場合に <ruby> を取り除く

"""

def replacer(match: re.Match) -> str:

group1 = match.group(1)

group2 = match.group(2)

if group1 == group2:

return group1

else:

return match.group(0)

replaced\_text = IDENTICAL\_RUBY\_PATTERN.sub(replacer, text)

return replaced\_text