以下は中国版の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”用于生成世界语文本(含汉字)替换的 JSON 文件工具.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、このアプリの中国の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中国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中国の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)

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

# 这里是 Streamlit 应用程序的主文件 (扩展功能版 202502)

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

import streamlit as st

import re

import io

import json

import pandas as pd # 如果需要的话使用

from typing import List, Dict, Tuple, Optional

import streamlit.components.v1 as components

import multiprocessing

# 在使用 multiprocessing 时，为避免 PicklingError，必须在 streamlit 中显式指定 "spawn"：

# 如果已经设置过 start method，则会抛出 RuntimeError，这里直接忽略

try:

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

except RuntimeError:

pass

# 从 esp\_text\_replacement\_module.py 中导入必要函数

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

)

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

# 通过对函数的结果进行 cache\_data，可以加快读取默认的替换用 JSON 文件(可能有 50MB左右)

# (约 1.0 秒 -> 0.5 秒 的加速)

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

@st.cache\_data

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

"""

从 JSON 文件中读取以下三种列表并以元组形式返回:

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,

)

# 设置页面基本信息

st.set\_page\_config(page\_title="（汉字替换）世界语文本转换工具", layout="wide")

# 页面主标题

st.title("将世界语文本转换为汉字形式，或生成带有汉字注释的 HTML (扩展版)")

st.write("---")

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

# 第1步：读取 JSON 文件（替换规则）。

# 用户可选择“使用默认”或“上传自定义 JSON”

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

selected\_option = st.radio(

"请选择替换规则 JSON 文件的读取方式：",

("使用默认 JSON", "上传 JSON 文件")

)

# 在折叠框中提供一个示例 JSON 文件可下载

with st.expander("【示例 JSON 文件（替换用）】"):

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="下载示例 JSON（替换用）",

data=file\_json,

file\_name="示例\_替换用JSON文件.json",

mime="application/json"

)

# 准备好这三个列表，以便之后进行替换

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]] = []

if selected\_option == "使用默认 JSON":

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("成功读取默认 JSON 文件。")

except Exception as e:

st.error(f"读取默认 JSON 文件时出错: {e}")

st.stop()

else:

uploaded\_file = st.file\_uploader("请上传 JSON 文件 (合并3个JSON文件).json 格式", 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("已成功读取上传的 JSON 文件。")

except Exception as e:

st.error(f"读取上传 JSON 文件时出错: {e}")

st.stop()

else:

st.warning("尚未上传 JSON 文件，无法继续执行。")

st.stop()

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

# 2) 读取一些与替换相关的 placeholder(占位符) 文件

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

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("---")

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

# 给用户提供一个并行处理（multiprocessing）的配置选项

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

st.header("高级设置：并行处理选项")

with st.expander("点击此处配置并行处理"):

st.write("""

如果文本很大，可以通过多进程并行处理加快转换速度。

请在此处勾选“使用并行处理”，并指定进程数。

""")

use\_parallel = st.checkbox("使用并行处理", value=False)

num\_processes = st.number\_input("并行进程数量", min\_value=2, max\_value=4, value=4, step=1)

st.write("---")

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

# 选择“输出形式”（HTML 或者带括号等等）

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

format\_type = st.selectbox(

"请选择输出格式（请与生成替换用JSON时的设定保持一致）：",

[

"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",

"括弧(号)格式",

"括弧(号)格式\_汉字替换",

"替换后文字列のみ(仅)保留(简单替换)"

]

)

# 准备一个全局字符串 processed\_text 来保存处理后的文本

processed\_text = ""

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

# 4) 选择如何输入待转换的文本（手动输入或上传文件）

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

st.subheader("输入文本来源")

source\_option = st.radio("请选择输入文本的方式：", ("手动输入", "上传文件"))

uploaded\_text = ""

if source\_option == "上传文件":

text\_file = st.file\_uploader("上传文本文件 (UTF-8 编码)", type=["txt", "csv", "md"])

if text\_file is not None:

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

st.info("文件已读取。")

else:

st.warning("尚未上传文本文件，请重新上传或切换为手动输入。")

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

# 使用表单的方式，让用户输入文本并提交

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

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

# 如果上传了文本，则在 text\_area 中默认填充

if uploaded\_text:

initial\_text = uploaded\_text

else:

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

text0 = st.text\_area(

"请输入世界语文章",

height=150,

value=initial\_text

)

st.markdown("""如果您使用“%”包裹文本（例如“%这段文本%”），则这部分内容将\*\*跳过替换\*\*。""")

st.markdown("""另外，如果您用“@”包裹文本（例如“@这段文本@”），则这部分内容将\*\*只做局部替换\*\*（即使用特定的局部替换列表）。""")

# 让用户选择“输出字符形式”（上标形式、x 形式、^ 形式）

letter\_type = st.radio('选择世界语字母形式', ('上标形式', 'x 形式', '^形式'))

# 提交、取消按钮

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

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

# 如果用户点击“取消”，则停止执行

if cancel\_btn:

st.warning("操作已取消。")

st.stop()

# 如果点击了“提交”

if submit\_btn:

# 将本次输入保存到会话状态

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

# 根据是否勾选并行处理，调用不同函数

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

)

# 将上标形式等应用到结果中

if letter\_type == '上标形式':

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)

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

# 表单外：若已经生成 processed\_text，则展示结果

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

if processed\_text:

# 如果文本过大，仅显示部分行：例如先显示前 47 行 + "..." + 后 3 行

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"文本行数较多（共 {len(lines)} 行），只显示部分内容（前 247 行 + 后 3 行）。"

)

else:

preview\_text = processed\_text

# 如果输出中带有 HTML，则分两个 Tab：一个用于渲染，一个用于查看源码

if "HTML" in format\_type:

tab1, tab2 = st.tabs(["HTML 预览", "HTML 源码"])

with tab1:

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

with tab2:

st.text\_area("HTML源码", preview\_text, height=300)

else:

tab3\_list = st.tabs(["转换结果"])

with tab3\_list[0]:

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

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

st.download\_button(

label="下载转换结果 (HTML)",

data=download\_data,

file\_name="转换结果.html",

mime="text/html"

)

st.write("---")

st.title("GitHub 仓库链接")

st.markdown("https://github.com/Takatakatake/Esperanto-Kanji-Converter-and-Ruby-Annotation-Tool\_Chinese\_beta")

st.markdown("<https://github.com/Takatakatake/Esperanto-Hanzi-Converter-and-Ruby-Annotation-Tool-Chinese>")

## 用于生成世界语文本(含汉字)替换的 JSON 文件工具.py(2つ目)

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

# 说明：

# 这是一个放置在 streamlit pages 文件夹中的脚本，用于生成“替换用 JSON 文件”

# main.py 会调用该 JSON 文件进行世界语文本中插入汉字/注释等操作。

# 本工具允许用户上传 CSV 和自定义 JSON，合并生成大规模的替换规则 JSON。

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

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

# 从 esp\_text\_replacement\_module.py 和 esp\_replacement\_json\_make\_module.py

# 导入必要的函数，用来进行世界语字符转换、并行处理构建替换字典等。

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

from esp\_text\_replacement\_module import (

convert\_to\_circumflex,

safe\_replace,

import\_placeholders,

apply\_ruby\_html\_header\_and\_footer

)

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

convert\_to\_circumflex,

output\_format,

import\_placeholders,

capitalize\_ruby\_and\_rt,

process\_chunk\_for\_pre\_replacements,

parallel\_build\_pre\_replacements\_dict,

remove\_redundant\_ruby\_if\_identical

)

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

# 以下是关于动词后缀(活用)以及若干特殊后缀(例如 an, on) 的变量定义。

# 这些数据会在生成 JSON 的流程中被使用，用来给特定单词自动添加后缀。

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

# 动词活用词尾 (as, is, os, us 等)

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'

}

# 例如 an, on 这两个列表，用于在处理世界语词根 + 后缀“an”/“on”时，自动拆分或区分不同含义

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 用于处理 -1 等表示不进行替换的自定义设定

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

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

# 以下三组列表用于处理 2 字母词根 （前缀/后缀/独立词根）

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

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'

]

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

# 下面从文件中读取相应的 placeholder，用于全域替换、二字词根替换、局部替换

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

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'

)

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

# 从 JSON 读取“Unicode\_BMP全范围字符宽度(Arial16).json”

# 用于在生成 HTML 形式时，针对文字宽度做精细化处理

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

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)

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

# 设置当前 Streamlit 页面的基本属性（标题、布局等）

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

st.set\_page\_config(

page\_title="生成用于世界语文本(汉字)替换的 JSON 文件",

layout="wide"

)

st.title("生成世界语文本(含汉字)替换所需的 JSON 文件")

st.write("---")

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

# 给出本页面功能的简单说明

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

with st.expander("使用说明（点击展开）", expanded=True):

st.markdown("""

\*\*本页面简介：\*\*

您可以在这里生成一份“世界语文本替换用的 JSON 文件”（可能体积较大，约 50MB），

main.py 中会使用该 JSON 文件来执行自动替换、加注汉字或 Ruby。

使用步骤如下：

1. 上传或使用默认的 CSV 文件（其中包含“世界语词根 → 中文（或汉字）”的对应关系）。

2. （可选）上传或使用默认的 JSON 文件（可自定义词根解析方式、特殊替换等）。

3. 点击按钮，即可生成“合并后的替换 JSON 文件”并下载。

如果您需要参考格式，可在下方折叠处下载示例文件。

""")

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

# 展示一些可下载的示例文件（CSV/JSON/Excel）

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

with st.expander("示例文件列表（点击下载）"):

st.write("\*\*示例文件：\*\*")

# 示例 CSV 1

st.markdown("""

\*\*示例 CSV 1：世界语词根 - 中文含义对应表\*\*

（如果您有自定义的 CSV，该 CSV 应有两列，第一列是世界语词根，第二列是对应的翻译/汉字/含义）

""")

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

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

btn = st.download\_button(

label="下载示例 CSV 1（词根 → 中文/汉字）",

data=file,

file\_name="示例\_世界语词根\_中文对应.csv",

mime="text/csv"

)

# 示例 CSV 2

st.markdown("""

\*\*示例 CSV 2：世界语词根 - 汉字 对应表（Mingeo 方案）\*\*

可作为参考，用 Mingeo 先生的汉字化思路。

""")

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:

btn = st.download\_button(

label="下载示例 CSV 2（Mingeo 先生版本）",

data=file,

file\_name="世界语词根\_汉字对应\_Mingeo.csv",

mime="text/csv"

)

# 示例 CSV 3

st.markdown("""

\*\*示例 CSV 3：世界语词根 - 汉字 对应表\*\*

""")

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:

btn = st.download\_button(

label="下载示例 CSV 3（词根 - 汉字）",

data=file,

file\_name="世界语词根\_汉字对应.csv",

mime="text/csv"

)

# 示例 JSON 1

st.markdown("""

\*\*示例 JSON 1：世界语单词词根解析的用户自定义设置\*\*

例如决定某些词是否算作动词，并自动添加(as,is,os)等后缀等。

文件内有详细注释，请结合实际需要进行编辑。

""")

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:

btn\_json = st.download\_button(

label="下载示例 JSON 1（自定义词根分解）",

data=file\_json,

file\_name="示例\_词根分解自定义.json",

mime="application/json"

)

# 示例 JSON 2

st.markdown("""

\*\*示例 JSON 2：自定义替换后文字（不太推荐）\*\*

除了 CSV + 词根分解 JSON，大多数情况不需要此额外文件。

仅在极少数场合，用于指定某些特殊词的特殊汉字。

""")

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:

btn\_json = st.download\_button(

label="下载示例 JSON 2（自定义替换后文字）",

data=file\_json,

file\_name="示例\_自定义替换后文字.json",

mime="application/json"

)

# 示例 Excel

st.markdown("""

\*\*示例 Excel：世界语词根 - 日语 词义+掌握等级

（仅作参考，帮助您确定常用/非常用词根，非必需文件）\*\*

""")

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="下载示例 Excel（带日语词义、掌握等级）",

data=file,

file\_name="示例\_世界语词根\_日语等级.xlsx",

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

)

st.write("---")

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

# 提供一个下拉菜单，让用户选择输出格式（例如 HTML+Ruby 等）

# （与 main.py 中的 format\_type 对应）

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

# 供用户选择的输出格式（原先是韩语/日语等），改用中文标签

options = {

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

'HTML形式＿调整Ruby文字大小（含汉字替换）': 'HTML格式\_Ruby文字\_大小调整\_汉字替换',

'HTML形式（不带汉字替换）': 'HTML格式',

'HTML形式（带汉字替换）': 'HTML格式\_汉字替换',

'括号形式（不带汉字替换）': '括弧(号)格式',

'括号形式（带汉字替换）': '括弧(号)格式\_汉字替换',

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

}

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

selected\_display = st.selectbox('请选择输出格式：', display\_options)

format\_type = options[selected\_display]

# 做一个小测试：展示几段文本的转换效果

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("---")

# 以 HTML + Ruby 的方式显示示例

st.markdown("\*\*格式化示例文本：\*\*")

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

st.write("---")

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

# 第 1 步：选择 CSV 文件（上传 或 使用默认）

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

st.header("第一步：准备 CSV 文件")

st.markdown("""

此 CSV 文件包含“世界语词根 - 翻译（汉字/中文）”信息。

请在下方选择使用默认示例文件，或上传自己的 CSV 文件。

""")

csv\_choice = st.radio("CSV 文件来源：", ("上传 CSV", "使用默认 CSV"))

csv\_path\_default = "./Appの运行に使用する各类文件/世界语词根-中文注释对应列表.csv"

CSV\_data\_imported = None

if csv\_choice == "上传 CSV":

st.write("请上传任意 CSV 文件（UTF-8 编码），包含两列：[词根, 翻译]")

uploaded\_file = st.file\_uploader("请选择 CSV 文件", type=['csv'])

if uploaded\_file is not None:

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

# 将内部所有世界语字母转换成带抑扬符（ĉ, ĝ 等）

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("CSV 文件已上传并读取。")

else:

st.warning("尚未上传任何 CSV 文件。")

st.stop()

elif csv\_choice == "使用默认 CSV":

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("已使用默认 CSV 文件。")

except FileNotFoundError:

st.error("找不到默认 CSV 文件，无法继续。")

st.stop()

st.write("CSV 文件加载完成。继续下一步。")

st.write("---")

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

# 第二步：准备 JSON 文件（包括词根分解规则、替换后文字等）

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

st.header("第二步：准备词根分解法/自定义替换 JSON 文件")

st.markdown("""

如果您有自定义的“词根解析/拆分”或“特殊替换后文字”，

可在这里上传 JSON 文件，也可使用示例 JSON 文件。

""")

# 1) 词根分解法 JSON

json\_choice = st.radio("1) 词根分解法 JSON 文件：", ("上传 JSON", "使用默认 JSON"))

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 == "上传 JSON":

uploaded\_json = st.file\_uploader("请上传词根分解自定义 JSON：", type=['json'])

if uploaded\_json is not None:

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

st.success("词根分解 JSON 文件已上传。")

else:

st.warning("尚未上传此 JSON 文件。")

st.stop()

elif json\_choice == "使用默认 JSON":

try:

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

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

st.info("已使用默认 JSON（词根分解）。")

except FileNotFoundError:

st.error("未找到默认词根分解 JSON 文件。")

st.stop()

# 2) 自定义替换后文字 JSON

json\_choice2 = st.radio("2) 替换后文字自定义 JSON 文件：", ("上传 JSON", "使用默认 JSON"))

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 == "上传 JSON":

uploaded\_json = st.file\_uploader("请上传替换后文字自定义 JSON：", type=['json'])

if uploaded\_json is not None:

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

st.success("自定义替换后文字 JSON 文件已上传。")

else:

st.warning("尚未上传此 JSON 文件。")

st.stop()

elif json\_choice2 == "使用默认 JSON":

try:

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

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

st.info("已使用默认 JSON（替换后文字）。")

except FileNotFoundError:

st.error("找不到默认的“替换后文字” JSON 文件。")

st.stop()

st.write("---")

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

# 第三步：可选的高级设置：并行处理

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

st.header("第三步：并行处理（高级设置）")

with st.expander("点击展开并行处理设置"):

st.write("""

若要生成大规模替换 JSON，处理速度可能较慢。

这里可以勾选“使用并行处理”，并设置要启动的 CPU 进程数（2~6）。

""")

use\_parallel = st.checkbox("使用并行处理", value=False)

num\_processes = st.number\_input("并行进程数量", min\_value=2, max\_value=6, value=5, step=1)

st.write("### 最后，生成替换用 JSON 文件")

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

# 在页面上放置一个按钮，点击后开始构建替换用 JSON

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

if st.button("生成并下载替换用 JSON 文件"):

with st.spinner("正在生成替换用 JSON 文件，请稍候……"):

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

# 1) 读取“大规模世界语词典/列表” (E\_stem\_with\_Part\_Of\_Speech\_list)

# 内含约数万条世界语单词(含词性信息)

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

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)

# 先读取“世界语全部词根(约11137个)”到一个临时字典

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():

# 临时记录：[替换后文字, 替换优先级]

# 先全部设成 E\_root 本身，优先级设为 len(E\_root)

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

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

# 2) 用 CSV 文件中的词根->(汉字/翻译) 覆盖这个临时字典

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

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)

]

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

# 3) 把临时字典转为列表，并根据优先级(字符串长度)从大到小排序

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

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)

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

# 4) 用 placeholder(占位符) 来构建 “(old->placeholder->new)” 的安全替换列表

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

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]

])

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

# 5) 对 E\_stem\_with\_Part\_Of\_Speech\_list 执行批量替换（可并行）

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

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:

# 如果用户不使用并行，则手动循环，顺便显示进度

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"处理进度：{current\_count}/{total\_items} ...")

progress\_bar.progress(100)

progress\_text.write("批量处理已完成 (100%)，可能还有少量收尾操作……")

# 例如可在这里 pop 掉某些键

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

for key in keys\_to\_remove:

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

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

# 6) 将 pre\_replacements\_dict\_1 进一步做各种优先级、后缀添加等处理：

# 生成 “replacements\_final\_list” 的最终形态

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

pre\_replacements\_dict\_2 = {}

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

if i==j[0]:

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

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

j[1],

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

]

else:

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={}

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

# 针对 AN、ON 等也进行对应处理

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:

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]

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

# 7) 应用自定义词根分解 JSON

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

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

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:

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.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 j\_ in i[2]:

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

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

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

continue

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

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

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.items():

replaced\_k2 = safe\_replace(k2, temporary\_replacements\_list\_final)

pre\_replacements\_dict\_3[esperanto\_Word\_before\_replacement + k1]=[Replaced\_String+replaced\_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\_ in i[2]:

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

j3 = safe\_replace(j\_,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

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

# 8) 应用“自定义替换后文字” JSON

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

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

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:

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

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

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

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.items():

replaced\_k2 = safe\_replace(k2,temporary\_replacements\_list\_final)

pre\_replacements\_dict\_3[esperanto\_Word\_before\_replacement + k1]=[Replaced\_String+replaced\_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\_ in i[2]:

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

j3 = safe\_replace(j\_,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

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

# 9) 将生成的 pre\_replacements\_dict\_3 转为列表，并按优先级排序

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

pre\_replacements\_list\_1=[]

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

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)

# 利用 remove\_redundant\_ruby\_if\_identical 删除可能出现的 parent=child 的情况

pre\_replacements\_list\_3=[]

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

if len(pre\_replacements\_list\_2[kk][0])>=3:

pre\_replacements\_list\_3.append([pre\_replacements\_list\_2[kk][0],remove\_redundant\_ruby\_if\_identical(pre\_replacements\_list\_2[kk][1]),imported\_placeholders\_for\_global\_replacement[kk]])

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

# 10) 针对大写/首字母大写等情况，再生成两条替换记录

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

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'):

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] + 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'):

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)'):

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$'))

replacements\_final\_list=[]

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

modified\_placeholder = place\_holder

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))

# 生成 2字词根的替换列表

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

# 用 CSV（仅包含词根→翻译) 构建“局部替换用列表”

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:

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:

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)])

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]

])

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

# 最终把这三种列表写进一个 JSON 并提供下载

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

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

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

st.success("替换用 JSON 列表构建完成！")

st.download\_button(

label="下载生成的（合并三份）替换 JSON 文件",

data=download\_data,

file\_name="世界语文本替换用\_合并3个JSON文件.json",

mime='application/json'

)

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

"""

本模块是“针对世界语文本进行字符串（汉字等）替换”的一系列工具函数。

主要功能：

1. 将各种世界语标记形式（带 x 的 cx, gx...、或带 ^ 的 c^, g^...）转换到字上符形式（ĉ, ĝ, ĥ 等）

2. 实现 %...%（跳过替换） 和 @...@（局部替换）的逻辑

3. safe\_replace()：使用 placeholder（占位符）进行安全替换

4. orchestrate\_comprehensive\_esperanto\_text\_replacement()：综合替换流程的核心函数

5. parallel\_process()：使用多进程来并行处理长文本

代码大体结构：

- 定义若干世界语字符转换的字典（如 x\_to\_circumflex 等）

- 提供若干辅助函数（unify\_halfwidth\_spaces, convert\_to\_circumflex...）

- 提供对 %...%、@...@ 的专门处理

- 提供 orchestrate\_comprehensive\_esperanto\_text\_replacement()，将多种替换整合起来

- parallel\_process() / process\_segment() 用于多进程并行处理长文本时的替换

"""

import re

import json

from typing import List, Tuple, Dict

import multiprocessing

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

# 1) 世界语字符转换相关的字典

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

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) 基本的字符转换函数

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

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

"""

将文本中的若干 key 替换为对应的 value。

例如，{'cx': 'ĉ', ...} 可以把“cx”全部替换成“ĉ”。

"""

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:

"""

将给定文本中的世界语特殊字母统一转换为字上符形式（ĉ, ĝ, ĥ, ĵ, ŝ, ŭ等）。

实际包括两步：hat\_to\_circumflex（将 c^转为ĉ 等），x\_to\_circumflex（将 cx转为ĉ 等）。

"""

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:

"""

将文本中的各种半角空白（如 \u00A0, \u2002 等）统一为 ASCII 标准半角空格 (U+0020)。

不处理全角空格 (U+3000)。

"""

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

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

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

# 3) 占位符（placeholder）相关

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

def safe\_replace(text: str, replacements: List[Tuple[str, str, str]]) -> str:

"""

执行安全替换：replacements 列表中每个元素是 (old, new, placeholder)。

先把 text 中的 old 全部替换为 placeholder，再把 placeholder 替换为 new。

这样可避免重复替换或交叉覆盖的问题。

"""

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 import\_placeholders(filename: str) -> List[str]:

"""

从指定文件读取 placeholder 列表。文件中每行一个 placeholder，返回一个列表。

"""

with open(filename, 'r') as file:

placeholders = [line.strip() for line in file if line.strip()]

return placeholders

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

# 用于 %...% (跳过替换) 的逻辑

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

PERCENT\_PATTERN = re.compile(r'%(.{1,50}?)%')

def find\_percent\_enclosed\_strings\_for\_skipping\_replacement(text: str) -> List[str]:

"""

在文本中查找形如 %foo% 的片段（1~50 字符），返回匹配部分（不含 %）。

"""

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]]:

"""

分析文本中的 %xxx% 段落，把它们映射到 placeholders。

返回类似 [("%xxx%", placeholder), ...]

"""

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

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

# 用于 @...@ (局部替换) 的逻辑

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

AT\_PATTERN = re.compile(r'@(.{1,18}?)@')

def find\_at\_enclosed\_strings\_for\_localized\_replacement(text: str) -> List[str]:

"""

查找 @foo@ 的片段（1~18 字符），返回提取的 foo。

"""

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]]:

"""

针对文本中出现的 @xxx@，用 replacements\_list\_for\_localized\_string 对其中的内容执行 safe\_replace。

最终返回 [("@xxx@", placeholder, replaced\_xxx), ...] 形式。

"""

matches = find\_at\_enclosed\_strings\_for\_localized\_replacement(text)

tmp\_replacements\_list\_for\_localized\_string = []

for i, match in enumerate(matches):

if i < len(placeholders):

replaced\_match = safe\_replace(match, replacements\_list\_for\_localized\_string)

tmp\_replacements\_list\_for\_localized\_string.append([f"@{match}@", placeholders[i], replaced\_match])

else:

break

return tmp\_replacements\_list\_for\_localized\_string

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

# 4) 综合替换主函数

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

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:

"""

进行一系列替换操作：

1) 统一半角空格

2) 将 c^, cx 等转为 ĉ, ĝ 等

3) 把 %...% 段落替换为占位符（跳过后续替换）

4) 把 @...@ 段落提取、执行局部替换后再替换成占位符

5) 对其余文本进行大范围替换（replacements\_final\_list）

6) 针对 2字词根（replacements\_list\_for\_2char）进行多次替换

7) 恢复 placeholder

8) 若是 HTML 形式，替换换行符为 <br>，空白处理等

"""

text = unify\_halfwidth\_spaces(text)

text = convert\_to\_circumflex(text)

# 处理 %...% 跳过替换

replacements\_list\_for\_intact\_parts = create\_replacements\_list\_for\_intact\_parts(text, placeholders\_for\_skipping\_replacements)

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\_)

# 处理 @...@ 局部替换

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\_)

# 大域替换

valid\_replacements = {}

for old, new, placeholder in replacements\_final\_list:

if old in text:

text = text.replace(old, placeholder)

valid\_replacements[placeholder] = new

# 2 字母词根，两次替换

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

# 恢复 placeholder

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)

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("%",""))

# 如果是 HTML 形式，可替换换行符为 <br> 等

if "HTML" in format\_type:

text = text.replace("\n", "<br>\n")

text = re.sub(r" ", "&nbsp;&nbsp;&nbsp;", text)

text = re.sub(r" ", "&nbsp;&nbsp;", text)

return text

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

# 5) 多进程处理长文本

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

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:

"""

用于并行处理的子函数：把若干行拼成一段，然后调用 orchestrate\_comprehensive\_esperanto\_text\_replacement。

"""

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:

"""

把文本按行拆分，分配给多个子进程并行处理（process\_segment），然后再拼接结果。

"""

if num\_processes <= 1:

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 = re.findall(r'.\*?\n|.+$', text)

num\_lines = len(lines)

if num\_lines <= 1:

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)]

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:

"""

根据所选 format\_type，为文本加上一段 HTML 头尾（主要是 <style> 设定等），

用于在浏览器渲染时控制 Ruby 字体大小或样式。

（如果不是 HTML 相关类型，返回原文即可）

"""

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'):

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>Ruby 显示</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%;

}

.text-M\_M {

font-size: 1rem!important;

font-family: Arial, sans-serif;

line-height: 2.0 !important;

display: block;

position: relative;

}

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;

}

rt {

display: block !important;

font-size: var(--ruby-font-size);

color: var(--ruby-color);

line-height: 1.05;

text-align: center;

}

rt.XXXS\_S {

--ruby-font-size: 0.3em;

margin-top: -8.3em !important;

transform: translateY(-0em) !important;

}

rt.XXS\_S {

--ruby-font-size: 0.3em;

margin-top: -7.2em !important;

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)

"""

此模块与 “esp\_text\_replacement\_module.py” 类似，主要用于 JSON 构建时的一些辅助函数，

包括：

- 字符转换函数（convert\_to\_circumflex）

- output\_format(...)：根据用户选择的输出类型，构建 <ruby> 结构 或 括号结构

- capitalize\_ruby\_and\_rt(...)：在 HTML ruby 中将首字母大写

- 并行替换相关函数（process\_chunk\_for\_pre\_replacements, parallel\_build\_pre\_replacements\_dict）

- remove\_redundant\_ruby\_if\_identical(...)：如果 <ruby>文本 与 <rt>文本 完全相同，则去除重复

它与 esp\_text\_replacement\_module.py 有所重叠/交叉，一部分函数实现思路类似，但为保持独立性可能重复定义。

"""

import re

import json

import multiprocessing

import pandas as pd

import os

from typing import List, Dict, Tuple, Optional

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

# 1) 世界语字符转换用的字典

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

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) 字符转换函数

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

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

"""

在 text 中，用 char\_dict 做普通字符串替换。

"""

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:

"""

将文本中的 c^, cx 等统一替换为 ĉ, ĝ 等。

"""

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

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

return text

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

# 3) 文字宽度测量 & <br> 插入

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

def measure\_text\_width\_Arial16(text, char\_widths\_dict: Dict[str, int]) -> int:

"""

利用从 JSON 中加载的 {char: width(px)}，计算 text 的总宽度像素值。

如果 char 不在字典中，默认宽度 8。

"""

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:

"""

测量 text 的宽度，找到中点位置附近，插入一个 <br>。

"""

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:

"""

把 total\_width / 3, 2/3 的位置各插一个 <br>，即插两处。

"""

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) output\_format(...) 函数

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

def output\_format(main\_text, ruby\_content, format\_type, char\_widths\_dict):

"""

根据用户选择的 format\_type，不同方式组合 main\_text 和 ruby\_content。

可能是 <ruby>main<rt>ruby</rt></ruby>，也可能是“main(ruby)”等。

并对过长的 ruby 或 main\_text 做 <br> 插入。

"""

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':

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) 其他辅助函数

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

def contains\_digit(s: str) -> bool:

"""

判断字符串 s 中是否含有数字字符。

"""

return any(char.isdigit() for char in s)

def import\_placeholders(filename: str) -> List[str]:

"""

从文件导入占位符。

"""

with open(filename, 'r') as file:

placeholders = [line.strip() for line in file if line.strip()]

return placeholders

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

# capitalize\_ruby\_and\_rt(...)

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

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:

"""

当需要把 <ruby>xxx<rt>yyy</rt></ruby> 中的 xxx 或 yyy 首字母大写时使用。

实际逻辑是先匹配，然后尝试做大写化。若没匹配到，就把整段做 text.capitalize()。

"""

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) 并行处理：用在创建 JSON 过程

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

def safe\_replace(text: str, replacements: List[Tuple[str, str, str]]) -> str:

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]]:

"""

针对 chunk（类似 [ [词根, 词性], ... ]）中的每个词根，执行 safe\_replace。

返回 { 词根: [ 替换后字符串, 合并词性 ], ... }。

"""

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]]:

"""

把 E\_stem\_with\_Part\_Of\_Speech\_list 切成若干块并行处理，再合并。

返回 { 词根: [ 替换后, 合并词性 ] }。

"""

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

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

# remove\_redundant\_ruby\_if\_identical

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

IDENTICAL\_RUBY\_PATTERN = re.compile(r'<ruby>([^<]+)<rt class="XXL\_L">([^<]+)</rt></ruby>')

def remove\_redundant\_ruby\_if\_identical(text: str) -> str:

"""

如果出现 <ruby>foo<rt class="XXL\_L">foo</rt></ruby>，则去掉外层 <ruby>，只保留 foo。

"""

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