```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Analyze the structure and broad properties of MMWR, PCD, EID, and PHR online
@author: chadheilig

This suite of scripts sets up to harvest the text and hypertext from the online
archives of 4 public health. The harvesting is limited to English and Spanish
publications, and it omits nontext components, such as media and items in the
Portable Document Format (though tools exist for harvesting text from those).

Each journal follows a hierarchy that includes some of the following levels:
* base: a source that points to the journal's home page
* home: the journal's canonical home page, with and overview of the archive
      MMWR  https://www.cdc.gov/mmwr/about.html
      PCD   https://www.cdc.gov/pcd/index.htm
      EID   https://wwwnc.cdc.gov/eid/
      PHR   https:////www.ncbi.nlm.nih.gov/pmc/journals/333/
* series:   the series that comprise components of the journal; MMWR has 4
* volumes:  yearly groupings of journal issues
      MMWR  volumes 31-69 (1982-2020)
      PCD   volumes 1-17 (2004-2020)
      EID   volumes 1-26 (1995-2020)
      PHR   volumes 116-133 (2001-2018)
* issues:   groupings of articles typically published simultaneously
* articles: primary unit of publication
* floats:   separate files that correspond to figures and tables (EID only)

This suite of scripts uses the level desginations above in names of
corresponding data objects.

Most levels begin with HTML sources to generate the following:
_a      list of anchor Tag ResultSets from a soup object
_a_n    number of Tags in each ResultSet in _a
_dframe pandas DataFrame
   base     URL from which href values were harvested
   href     hypertext reference (bs.a['href']); + base URL -> absolute URL
   url      absolute URL constructed from href and base URL
   path     path from absolute URL
   filename name of HTML file in hypertext reference
   mirror_path path on local mirror
   string   string from anchor element content
   level    in concatentated DataFrame, volume or article
_repeated list of indices for URLs that are duplicated
_html   full HTML text as retrieved and processed from each URL
_soup   soup objects that parse HTML text

The "analyze-structure" scripts follow a systematic approach to harvesting each
level of the journal's hierarchy, resulting in a journal-specific DataFrame (a
tabular data object) with the URL and other information on each file to be
mirrored in a local archive for further processing and analysis.
"""

#%% Import modules and set up environment
# operating system muodules to work with filenames and paths
import os
```

```
 57 from os.path import join, expanduser, normpath
 58 # built-in urllib module to work with URLs
 59 from urllib.parse import urlparse, urljoin, urlunparse
 60 # requests module to retrieve HTML files over the internet
 61 import requests
 62 # built-in regular expression module to work with patterns of text
 63 import re
 64 # UnicodeDammit module to handle character encoding gracefully
 65 # BeautifulSoup module to parse, analyze, and write HTML
 66 from bs4 import BeautifulSoup, UnicodeDammit
 67 from bs4.formatter import HTMLFormatter
 68 # pandas module to organize metadata in DataFrame structure
 69 import pandas as pd
 70 # built-in pickle module to serialize and store intermediate data structures
 71 import pickle
 72 # multiprocessing and tqdm modules for utilities to evaluate progress
 73 import multiprocessing
 74 from tqdm import tqdm, trange
 75
 76 # global, session-specific pandas option
 77 pd.set_option('display.expand_frame_repr', False) # show/wrap all DF columns
 78
 79 #%% 1. Functions for operating on URLs and paths
 80
 81 # requests.get(url).text decodes to utf-8, which could mismatch
 82 # requests.get(url).content is bytestream
 83 # UnicodeDammit(content, ["utf-8", "windows-1252"]) tries to improve match
 84 #    tries utf-8 first, but falls back to windows-1252 if there's an error
 85 #    .unicode_markup appears to be the same as .markup
 86 # reduce all non-HTML whitespace to single space character
 87 def get_html_from_url(url, print_url = False, timeout = 1):
 88    """
 89    Parameters
 90    url : str
 91       Absolute URL from which to retrieve HTML document.
 92
 93    1. Absolute URL passes to requests.get
 94    2. bytes get() result passes to UnicodeDammit
 95    3. UnicodeDammit attempts decoding to UTF-8 else to Windows-1252
 96    4. unicode_markup is (one hopes) clean UTF-8-encoded HTML
 97    5. all whitespace sequences (including \r, \n) reduce to single space (' ')
 98
 99    Returns
100    str
101       UTF-8 encoded HTML string with minimal white space.
102
103    requests.get(timeout=5) is hardcoded to deal with sluggish EID responses
104    """
105    if(print_url):
106       print(f'Retrieving URL {url}')
107    try:
108       html = re.sub(r'\s+', ' ',
109          UnicodeDammit(
110             requests.get(url, timeout=timeout).content,
111             ["utf-8", "windows-1252"]).unicode_markup)
112    except:
```

```
113         html = ''
114     return html
115 
116 # curried version that prints URL
117 def get_html_from_url_(url):
118     return get_html_from_url(url, print_url = True, timeout = 5)
119 
120 def process_aTag(aTag, base_url='https://www.cdc.gov'):
121     """
122     aTag: bs4.element.Tag anchor element from soup
123     base_url: str base URL containing (or that could contain) anchor element
124     Construct absolute URL from anchor's hypertext reference and base URL
125     returns dict: { base_url, href, url, path, file_in, file_out, string }
126     """
127     try:
128         a_href = aTag['href']
129     except:
130         print('Unable to extract href from anchor.')
131         return None
132     joined = urljoin(base_url, a_href)
133     parsed = urlparse(joined)
134     parsed = parsed._replace(scheme='https', params='', query='', fragment='')
135     a_url = parsed.geturl()
136     a_path = parsed.path
137     a_dirname = os.path.dirname(a_path)
138     a_basename = os.path.basename(a_path)
139     a_baseext = os.path.splitext(a_basename)
140     # construct filename for local mirror if internet filename isn't *.htm*
141     if a_basename == '':
142         m_filename = 'index.html'
143 #   elif re.search(r'\.html?', a_fname) is None:
144     elif a_baseext[1] == '':
145         m_filename = a_baseext[0] + '.html'
146     else:
147         m_filename = a_basename
148     m_path = os.path.join(a_dirname, m_filename)
149     a_text = aTag.get_text('|', strip = True)
150     return dict(base=base_url, href=a_href, url=a_url, path=a_path,
151         filename=a_basename, mirror_path=m_path, string=a_text)
152 
153 # UnsortedAttributes formatter adapted from BeautifulSoup documentation
154 # from bs4.formatter import HTMLFormatter
155 # renders formatted HTML as close to input as feasible, with nice indents
156 class UnsortedAttributes(HTMLFormatter):
157     def attributes(self, tag):
158         for k, v in tag.attrs.items():
159             yield k, v
160 
161 # using UnsortedAttributes in this way can yield invalid HTML
162 # because of < or > in open text not being rendered as &lt; or &gt;
163 # not sure how to prevent sorting and ensure valid code
164 # (short of hacking BeautifulSoup source)
165 
166 #%% 2. Functions for minimally processing HTML files as bytes or strings
167 
168 def calculate_mirror_dirs(paths):
```

```
169     "Given set of paths (with or without filenames), determine unique folders"
170     dirnames = [os.path.dirname(path) for path in paths]
171     dirnames = sorted(set(dirnames))
172     return dirnames
173 
174 def create_mirror_tree(base_path, dirnames):
175     """
176     Nondestructively constructs directory tree for journal archive mirror based
177     on paths in URLs to be retrieved.
178 
179     Parameters
180     base_path : str
181        base of path in which to construct journal mirror directory tree.
182     dirnames : str
183        list of paths to graft onto base_path and populate with HTML files
184 
185     Returns
186     None
187     """
188     from os import makedirs
189     from os.path import join, expanduser, normpath
190 
191     # process paths to separate head from tail
192     # get unique set of path heads
193     base_path = normpath(expanduser(base_path))
194     norm_paths = [normpath(join(base_path, dir[1:] if dir[0]=='/' else dir))
195                   for dir in dirnames]
196 
197     # clean up base_path, then try to construct tree
198     try:
199        makedirs(base_path, exist_ok=True) # the rest will work iff this does
200        for norm_path in norm_paths:
201            makedirs(norm_path, exist_ok=True)
202        print(f"Successfully made directories from {base_path}")
203        return dict(base_path = [base_path], paths = dirnames,
204                    norm_paths = norm_paths)
205     except: # could try to trap each possible error type
206        print(f"Unable to make directories from {base_path}")
207        return None
208 
209 # retrieve unprocessed HTML and immediately write it to local mirror
210 def mirror_raw_html(url, mirror_path, timeout = 1, print_url = True):
211     """Use requests.get to retrieve raw (bytes) version of HTML file
212     and write unprocessed HTML to local mirror."""
213     if(print_url):
214        # print('.', end = '')
215        print(f'Processing URL {url}', end = '')
216     try:
217        b0 = requests.get(url, timeout = timeout).content
218     except:
219        b0 = b''
220     with open(mirror_path, 'bw') as file_out:
221         file_out.write(b0)
222     if(print_url):
223         print('.')
224     return len(b0)
```

```
225
226 # Small utilities to separate sequence of operations on HTML
227 # x_b take bytes; x_u take strings (UTF-8)
228 # sub(' ?(<|>) ?', r'\1') works in well-formed HTML
229 def html_reduce_space_b(str_b):
230    str_b = re.sub(br'\s+', b' ', str_b)
231    # str_b = re.sub(b' ?(<|>) ?', br'\1', str_b)
232    return str_b
233
234 def html_insert_newlines_b(str_b):
235    # assumes result of html_reduce_space_b(); judiciously insert newlines
236    str_b = re.sub(b'>(.)', br'>\n\1', str_b)
237    str_b = re.sub(br'([^\n])<', br'\1\n<', str_b)
238    return str_b
239
240 def html_to_unicode_b(str_b):
241    # converts raw HTML (bytes) to Unicode HTML (str)
242    str_u = UnicodeDammit(str_b, ['utf-8', 'windows-1252']).unicode_markup
243    return str_u
244
245 def html_reduce_space_u(str_u):
246    str_u = re.sub(r'\s+', ' ', str_u)
247    # str_u = re.sub(' ?(<|>) ?', r'\1', str_u)
248    return str_u
249
250 def html_insert_newlines_u(str_u):
251    # assumes result of html_reduce_space_u(); judiciously insert newlines
252    str_u = re.sub('>(.)', r'>\n\1', str_u)
253    str_u = re.sub(r'([^\n])<', r'\1\n<', str_u)
254    return str_u
255
256 def html_prettify_u(str_u):
257    str_u = BeautifulSoup(str_u, 'lxml')\
258       .prettify(formatter = 'minimal')
259       # .prettify(formatter = UnsortedAttributes())
260    return str_u
261
262 def trim_leading_space_u(str_u):
263    # assumes result of basic_prettify()
264    return re.sub(r'^\s+', '', str_u, flags=re.M)
265
266 # Functions to work with local mirror
267 def mirror_raw_to_uni(b_path, u_path, counter = None):
268    "Mirror local, unprocessed HTML to local, lightly processed HTML."
269    if counter is not None:
270       print(f'{counter:05d}', end = '')
271    b0 = read_raw_html(b_path)
272    u0 = html_to_unicode_b(b0)      # convert bytes to UTF-8
273    u00 = html_reduce_space_u(u0)   # scrub u0 of excess space (esp. \r)
274    u02 = html_prettify_u(u00)      # prettify u00
275    u03 = trim_leading_space_u(u02) # scrub u02 of leading spaces
276    with open(u_path, 'w') as file_out:
277        file_out.write(u03)
278    if counter is not None:
279       print('.', end = ' ')
280    return None
```

```
281
282 def read_raw_html(path):
283     "Read raw (bytes) local copy of HTML file."
284     with open(path, 'rb') as file_in:
285         raw_html = file_in.read()
286     return raw_html
287
288 def read_uni_html(path):
289     "Read Unicode (UTF-8 string) local copy of HTML file."
290     with open(path, 'r') as file_in:
291         uni_html = file_in.read()
292     return uni_html
```