```python
#!/usr/bin/env python3

"""
This is the main script that renders any source code to pdf format with syntax
highlighting and line number and more. It has support for downloading and rendering
any online source code using requests and BeautifulSoup.
"""

from signal import SIGINT, signal
signal(SIGINT, lambda signum, frame: sys.exit(1))

import os
import re
import sys
import requests
import termcolor

from argparse import ArgumentParser
from backports.shutil_get_terminal_size import get_terminal_size
from braceexpand import braceexpand
from bs4 import BeautifulSoup
from copy import copy
from glob import glob
from natsort import natsorted, ns
from pkg_resources import DistributionNotFound, get_distribution
from pygments import highlight
from pygments.formatters import HtmlFormatter
from pygments.lexers import get_lexer_for_filename, guess_lexer
from pygments.lexers.special import TextLexer
from PyPDF2 import PdfFileReader, PdfFileWriter
from requests.exceptions import RequestException
from tempfile import mkstemp
from textwrap import fill
from traceback import print_exception
from urllib.parse import urljoin
from warnings import filterwarnings


# Require Python 3.6+
if sys.version_info < (3, 6):
    sys.exit("You have an old version of python. Install version 3.6 or higher.")

```

```
43  # Get version
44  try:
45      d = get_distribution("render50")
46  except DistributionNotFound:
47      __version__ = "UNKNOWN"
48  else:
49      __version__ = d.version
50
51
52  def main():
53
54      # Exit on ctrl-c
55      def handler(signum, frame):
56          cprint("")
57          cancel(1)
58
59      # Register handler
60      signal(SIGINT, handler)
61
62      # Parse command-line arguments
63      parser = ArgumentParser(description="A command-line tool that renders source code as a PDF.")
64      parser.add_argument("-b", "--browser", action="store_true", help="render as a browser would")
65      parser.add_argument("-f", "--force", action="store_true", default=False, help="overwrite existing files
without prompting")
66      parser.add_argument("-C", "--no-color", action="store_true", help="disable syntax highlighting")
67      parser.add_argument("-i", "--include", action="append", help="pattern to include")
68      parser.add_argument("-o", "--output", help="file to output", required=True)
69      parser.add_argument("-P", "--no-path", action="store_true", default=False, help="omit paths in headers")
70      parser.add_argument("-r", "--recursive", action="store_true", help="recurse into directories")
71      parser.add_argument("-s", "--size", help="size of page, per https://developer.mozilla.org/en-US/docs/Web/CSS
/@page/size")
72      parser.add_argument("-x", "--exclude", action="append", help="pattern to exclude")
73      parser.add_argument("-y", "--side-by-side", action="store_true", help="render inputs side by side")
74      parser.add_argument("INPUT", help="file or URL to render", nargs="*")
75      parser.add_argument("-V", "--version", action="version", version="%(prog)s {}".format(__version__))
76      args = parser.parse_args(sys.argv[1:])
77
78      # Ensure output ends in .pdf
79      output = args.output
80      if not output.lower().endswith(".pdf"):
81          output += ".pdf"
82
```

```
83      # Check for input
84      if args.INPUT:
85          inputs = args.INPUT
86      else:
87          inputs = [line.strip() for line in sys.stdin.readlines()]
88
89      # Check for includes
90      includes = []
91      if args.include:
92          for i in args.include:
93              includes.append(re.escape(i).replace("\*", ".*"))
94
95      # Check for excludes
96      excludes = []
97      if args.exclude:
98          for x in args.exclude:
99              excludes.append(re.escape(x).replace("\*", ".*"))
100
101     # Check stdin for inputs else command line
102     patterns = []
103     if len(inputs) == 1 and inputs[0] == "-":
104         patterns = sys.stdin.read().splitlines()
105     else:
106         patterns = inputs
107
108     # Glob patterns lest shell (e.g., Windows) or stdin not have done so, ignoring empty patterns
109     paths = []
110     for pattern in patterns:
111         if pattern.startswith(("http", "https")):
112             paths += [pattern]
113         if pattern:
114             for expression in list(braceexpand(pattern)):
115                 paths += natsorted(glob(expression, recursive=True), alg=ns.IGNORECASE)
116
117     # Candidates to render
118     candidates = []
119     for path in paths:
120         if os.path.isfile(path) or path.startswith(("http", "https")):
121             candidates.append(path)
122         elif os.path.isdir(path):
123             files = []
124             for dirpath, dirnames, filenames in os.walk(path):
```

```
125                 for filename in filenames:
126                     files.append(os.path.join(dirpath, filename))
127             files = natsorted(files, alg=ns.IGNORECASE)
128             candidates += files
129         else:
130             raise RuntimeError("Could not recognize {}.".format(path))
131 
132     # Filter candidates
133     queue = []
134     for candidate in candidates:
135 
136         # Skip implicit exclusions
137         if includes and not re.search(r"^" + r"|".join(includes) + "$", candidate):
138             continue
139 
140         # Skip explicit exclusions
141         if excludes and re.search(r"^" + r"|".join(excludes) + "$", candidate):
142             continue
143 
144         # Queue candidate for rendering
145         queue.append(candidate)
146 
147     # If side-by-side
148     if args.side_by_side:
149 
150         # Expect 2 or 3 inputs
151         if len(queue) < 2:
152             raise RuntimeError("Too few files to render side by side.")
153         elif len(queue) > 3:
154             raise RuntimeError("Too many files to render side by side.")
155 
156     # If rendering as browser would
157     if args.browser:
158 
159         # Expect 1 input
160         if len(queue) != 1:
161             raise RuntimeError("Can only render one input as browser would.")
162 
163 
164     # Prompt whether to overwrite output
165     if not args.force and os.path.exists(output):
166         if not args.INPUT:  # If using stdin for inputs
```

```
167             raise RuntimeError("Output exists.")
168         while True:
169             s = input("Overwrite {}? ".format(output))
170             if s.lower() in ["y", "yes"]:
171                 break
172             elif s.lower() in ["n", "no"]:
173                 cancel()
174 
175     # Create parent directory as needed
176     dirname = os.path.dirname(os.path.realpath(output))
177     if not os.path.isdir(dirname):
178         while True:
179             s = input("Create {}? ".format(dirname)).strip()
180             if s.lower() in ["n", "no"]:
181                 cancel()
182             elif s.lower() in ["y", "yes"]:
183                 try:
184                     os.makedirs(dirname)
185                 except Exception:
186                     raise RuntimeError("Could not create {}.".format(dirname))
187 
188     # Determine size
189     # https://developer.mozilla.org/en-US/docs/Web/CSS/@page/size
190     if not args.size:
191         if args.browser:
192             size = "letter portrait"
193         else:
194             size = "letter landscape"
195     else:
196         size = args.size.strip()
197         if not re.search(r"^[A-Za-z0-9\- ]+$", size):
198             raise RuntimeError("Invalid size.")
199         if size in ["A5", "A4", "A3", "B5", "B4", "JIS-B5", "JIS-B4", "letter", "legal", "ledger"]:
200             size = "{} landscape".format(size)
201 
202     # Render input as browser would, effectively screenshotting page
203     if args.browser:
204 
205         # frameset
206         try:
207             page = get(queue[0])
208             soup = BeautifulSoup(page, "html.parser")
```

```
209             framesets = soup.find_all(
210                 "frameset", {"cols": re.compile("(50%,\s*50%|33%,\s*33%,\s*33%)")})
211             assert len(framesets) == 1
212             frames = framesets[0].find_all("frame")
213             assert 2 <= len(frames) <= 3
214             if not args.size:
215                 size = "letter landscape"
216             args.side_by_side = True
217             queue = [join(queue[0], frame.attrs["src"]) for frame in frames]
218 
219         # noframes
220         except:
221 
222             # Render input
223             document = render(queue[0], browser=True, size=size)
224             if not document:
225                 cancel(1)
226 
227             # Write rendered output
228             if not write(output, [document]):
229                 cancel(1)
230             cprint("Rendered {}.".format(output), "green")
231             sys.exit(0)
232 
233     # Render inputs side by side
234     if args.side_by_side:
235 
236         # Divide width
237         document = blank(size)
238         width = int(document.pages[0].width / len(queue))
239         height = int(document.pages[0].height)
240         size = "{}px {}px".format(width, height)
241 
242         # temporary files
243         temps = []
244 
245         # Render first input
246         temps.append(mkstemp())
247         document = render(queue[0], browser=args.browser, color=not args.no_color,
248                           fontSize="8pt", margin=".5in .25in .5in .5in", path=not args.no_path, relative=False, si
ze=size)
249         if not document:
```

```
250             cancel(1)
251         write(temps[0][1], [document], False)
252 
253         # Render second input
254         temps.append(mkstemp())
255         margin = ".5in .5in .5in .25in" if len(queue) == 2 else ".5in .375in .5in .375in"
256         document = render(queue[1], browser=args.browser, color=not args.no_color,
257                           fontSize="8pt", margin=margin, path=not args.no_path, relative=False, size=size)
258         if not document:
259             cancel(1)
260         write(temps[1][1], [document], False)
261 
262         # Render third input, if any
263         if len(queue) == 3:
264             temps.append(mkstemp())
265             document = render(queue[2], browser=args.browser, color=not args.no_color,
266                               fontSize="8pt", margin=".5in .5in .5in .25in", path=not args.no_path, relative=False,
size=size)
267             if not document:
268                 cancel(1)
269             write(temps[2][1], [document], False)
270 
271         # Concatenate inputs
272         concatenate(output, list(map(lambda f: f[1], temps)))
273 
274         # Remove temporary files
275         map(lambda f: os.close(f[0]), temps)
276         map(lambda f: os.remove(f[1]), temps)
277 
278         # Rendered
279         cprint("Rendered {}.".format(output), "green")
280         sys.exit(0)
281 
282     # Render queued files
283     documents = []
284     for queued in queue:
285         document = render(queued, color=not args.no_color, path=not args.no_path, size=size)
286         if document:
287             documents.append(document)
288 
289     # Write rendered files
290     if not write(output, documents):
```

```
291         cancel(1)
292
293
294 def blank(size):
295     """Render blank page of specified size."""
296     return HTML(string="").render(stylesheets=[CSS(string="@page {{ size: {}; }}".format(size))])
297
298
299 def cancel(code=0):
300     """Report cancellation, exiting with code"""
301     cprint("Rendering cancelled.", "red")
302     sys.exit(code)
303
304
305 def concatenate(output, inputs):
306     """Concatenate (PDF) inputs side by side."""
307
308     # Read files
309     readers = list(map(PdfFileReader, inputs))
310
311     # Render blank page, inferring size from first input's first page
312     temp = mkstemp()
313     size = "{}pt {}pt".format(readers[0].getPage(0).mediaBox[2], readers[0].getPage(0).mediaBox[3])
314     write(temp[1], [blank(size)], False)
315     page = PdfFileReader(temp[1]).getPage(0)
316
317     # Concatenate files side by side
318     writer = PdfFileWriter()
319
320     # Concatenate pages
321     for i in range(max(map(lambda r: r.getNumPages(), readers))):
322
323         # Leftmost page
324         left = copy(readers[0].getPage(i)) if i < readers[0].getNumPages() else copy(page)
325
326         # Rightmost pages
327         for reader in readers[1:]:
328             right = copy(reader.getPage(i)) if i < reader.getNumPages() else copy(page)
329             left.mergeTranslatedPage(right, left.mediaBox[2], 0, expand=True)
330
331         # Add pages to output
332         writer.addPage(left)
```

```
333
334         # Ouput PDF
335         with open(output, "wb") as file:
336             writer.write(file)
337
338         # Remove temporary files
339         os.close(temp[0]), os.remove(temp[1])
340
341
342     def cprint(text="", color=None, on_color=None, attrs=None, end="\n"):
343         """Colorize text (and wraps to terminal's width)."""
344
345         # Assume 80 in case not running in a terminal
346         columns, _ = get_terminal_size()
347         if columns == 0:
348             columns = 80
349
350         # Print text, flushing output
351         termcolor.cprint(fill(text, columns, drop_whitespace=False, replace_whitespace=False),
352                          color=color, on_color=on_color, attrs=attrs, end=end)
353         sys.stdout.flush()
354
355
356     def excepthook(type, value, tb):
357         """Report an exception."""
358         excepthook.ignore = False
359         if type is RuntimeError and str(value):
360             cprint(str(value), "yellow")
361         else:
362             cprint("Sorry, something's wrong! Let sysadmins@cs50.harvard.edu know!", "yellow")
363             print_exception(type, value, tb)
364         cancel(1)
365
366
367     sys.excepthook = excepthook
368
369
370     def get(file):
371         """Gets contents of file locally or remotely."""
372
373         # Check if URL
374         if file.startswith(("http", "https")):
```

```
375
376         # Get from raw.githubusercontent.com
377         matches = re.search(r"^(https?://github.com/[^/]+/[^/]+)/blob/(.+)$", file)
378         if matches:
379             file = "{}/raw/{}".format(matches.group(1), matches.group(2))
380         matches = re.search(r"^(https?://gist.github.com/[^/]+/[^/#]+/?)(?:#file-(.+))?$", file)
381         if matches:
382             file = "{}/raw".format(matches.group(1))
383             if matches.group(2):
384                 file += "/{}".format(matches.group(2))
385
386         # Get file
387         req = requests.get(file)
388         if req.status_code == 200:
389             return req.text
390         else:
391             cprint("\033[2K", end="\r")
392             raise RuntimeError("Could not GET {}.".format(file))
393
394     # Read file
395     else:
396         try:
397             with open(file, "rb") as f:
398                 return f.read().decode("utf-8", "ignore")
399         except Exception as e:
400             cprint("\033[2K", end="\r")
401             if type(e) is FileNotFoundError:
402                 raise RuntimeError("Could not find {}.".format(file))
403             else:
404                 raise RuntimeError("Could not read {}.".format(file))
405
406
407 def join(a, b):
408     """Join a and b, where each is a URL, an absolute path, or a relative path."""
409
410     # If b is a URL, don't join
411     if b.startswith(("http://", "https://")):
412         return b
413
414     # if a is a URL (and b is not), join with b
415     if a.startswith(("http://", "https://")):
416         return urljoin(a, b)
```

```
417
418         # if a is an absolute or a relative path, join with b
419         else:
420             return os.path.normpath(os.path.join(os.path.dirname(a), b))
421
422
423     def render(filename, size, browser=False, color=True, fontSize="10pt", margin=".5in", path=True, relative=True):
424         """Render file with filename as HTML page(s) of specified size."""
425
426         # Rendering
427         cprint("Rendering {}...".format(filename), end="")
428
429         # Render as a browser would
430         if browser:
431
432             # Styles for document
433             stylesheets = [
434                 CSS(string="@page {{ margin: {}; size: {}; }}".format(margin, size)),
435                 CSS(string="html {{ font-size: {}; }}".format(fontSize))]
436
437             # Render document
438             try:
439
440                 # Parse HTML
441                 soup = BeautifulSoup(get(filename), "html.parser")
442
443                 # Remove relative links (for side-by-side outputs, for which we concatenate PDFs page-wise)
444                 if not relative:
445                     for a in soup.find_all("a"):
446                         if a["href"].startswith("#"):
447                             del a["href"]
448
449                 # Re-parse HTML
450                 document = HTML(base_url=os.path.dirname(filename),
451                                 string=str(soup)).render(stylesheets=stylesheets)
452
453             except Exception as e:
454                 cprint("\033[2K", end="\r")
455                 if type(e) in [RequestException, URLFetchingError]:
456                     raise RuntimeError("Could not GET {}.".format(filename))
457                 else:
458                     raise RuntimeError("Could not read {}.".format(filename))
```

```
459
460       # Syntax-highlight instead
461       else:
462
463           # Get code from file
464           code = get(filename)
465
466           # Check whether binary file
467           if "\x00" in code:
468               cprint("\033[2K", end="\r")
469               cprint("Could not render {} because binary.".format(filename), "yellow")
470               return None
471
472           # Highlight code unless file is empty, using inline line numbers to avoid
473           # page breaks in tables, https://github.com/Kozea/WeasyPrint/issues/36
474           if code.strip() and color:
475               try:
476                   lexer = get_lexer_for_filename(filename)
477               except:
478                   try:
479                       assert code.startswith("#!")  # else, e.g., a .gitignore file with a dotfile is mistaken by
GasLexer
480                       lexer = guess_lexer(code.splitlines()[0])
481                   except:
482                       lexer = TextLexer()
483               string = highlight(code, lexer, HtmlFormatter(linenos="inline", nobackground=True))
484           else:
485               string = highlight(code, TextLexer(), HtmlFormatter(
486                   linenos="inline", nobackground=True))
487
488           # Styles for document
489           title = filename if path else os.path.basename(filename)
490           stylesheets = [
491               CSS(string="@page {{ border-top: 1px #808080 solid; margin: {}; padding-top: 1em; size: {}; }}".form
at(margin, size)),
492               CSS(string="@page {{ @top-right {{ color: #808080; content: '{}'; padding-bottom: 1em; vertical-
align: bottom; }} }}".format(
493                   title.replace("'", "\'"))),
494               CSS(string="* {{ font-family: monospace; font-size: {}; margin: 0; overflow-wrap: break-word; white-
space: pre-wrap; }}".format(fontSize)),
495               CSS(string=HtmlFormatter().get_style_defs('.highlight')),
496               CSS(string=".highlight { background: initial; }"),
```

```
497             CSS(string="span.linenos { background-color: inherit; color: #808080; }"),
498             CSS(string="span.linenos:after { content: '  '; }")]
499 
500         # Render document
501         document = HTML(string=string).render(stylesheets=stylesheets)
502 
503     # Bookmark document
504     document.pages[0].bookmarks = [(1, title, (0, 0), "closed")]
505 
506     # Rendered
507     cprint("\033[2K", end="\r")
508     cprint("Rendered {}.".format(filename))
509     return document
510 
511 
512 def write(output, documents, echo=True):
513     """
514     Write documents to output as PDF.
515     https://github.com/Kozea/WeasyPrint/issues/212#issuecomment-52408306
516     """
517     if documents:
518         pages = [page for document in documents for page in document.pages]
519         documents[0].copy(pages).write_pdf(output)
520         if echo:
521             cprint("Rendered {}.".format(output), "green")
522         return True
523     else:
524         return False
525 
526 
527 # Check for dependencies
528 # http://weasyprint.readthedocs.io/en/latest/install.html
529 try:
530     # Ignore warnings about outdated Cairo and Pango (on Ubuntu 14.04, at least)
531     filterwarnings("ignore", category=UserWarning, module="weasyprint")
532     from weasyprint import CSS, HTML
533     from weasyprint.urls import URLFetchingError
534 except OSError as e:
535     if "pangocairo" in str(e):
536         raise RuntimeError("Missing dependency. Install Pango.")
537     elif "cairo" in str(e):
538         raise RuntimeError("Missing dependency. Install cairo.")
```

```
539         else:
540             raise RuntimeError(str(e))
541 
542 
543 if __name__ == "__main__":
544     main()
```