Forever Journal

2026 – 2035

```
Start Year:  2026
Num Years:  10
Lines/Day:  5
Sundays Red:  True
Paper:  A4
Test Mode:  True
Spread:  4up (2 day/page)
Align:  left
Generated:  2025-12-23 13:37:47
```

Table of Contents

February Summary

	2026	2027	2028	2029	2030
1	Su	Mo	Tu	Th	Fr
2	Mo	Tu	We	Fr	Sa
3	Tu	We	Th	Sa	Su
4	We	Th	Fr	Su	Mo
5	Th	Fr	Sa	Mo	Tu
6	Fr	Sa	Su	Tu	We
7	Sa	Su	Mo	We	Th
8	Su	Mo	Tu	Th	Fr
9	Mo	Tu	We	Fr	Sa
10	Tu	We	Th	Sa	Su
11	We	Th	Fr	Su	Mo
12	Th	Fr	Sa	Mo	Tu
13	Fr	Sa	Su	Tu	We
14	Sa	Su	Mo	We	Th
15	Su	Mo	Tu	Th	Fr
16	Mo	Tu	We	Fr	Sa
17	Tu	We	Th	Sa	Su
18	We	Th	Fr	Su	Mo
19	Th	Fr	Sa	Mo	Tu
20	Fr	Sa	Su	Tu	We
21	Sa	Su	Mo	We	Th
22	Su	Mo	Tu	Th	Fr
23	Mo	Tu	We	Fr	Sa
24	Tu	We	Th	Sa	Su
25	We	Th	Fr	Su	Mo
26	Th	Fr	Sa	Mo	Tu
27	Fr	Sa	Su	Tu	We
28	Sa	Su	Mo	We	Th
29			Tu		

February Summary

	2031	2032	2033	2034	2035
1	Sa	Su	Tu	We	Th
2	Su	Mo	We	Th	Fr
3	Mo	Tu	Th	Fr	Sa
4	Tu	We	Fr	Sa	Su
5	We	Th	Sa	Su	Mo
6	Th	Fr	Su	Mo	Tu
7	Fr	Sa	Mo	Tu	We
8	Sa	Su	Tu	We	Th
9	Su	Mo	We	Th	Fr
10	Mo	Tu	Th	Fr	Sa
11	Tu	We	Fr	Sa	Su
12	We	Th	Sa	Su	Mo
13	Th	Fr	Su	Mo	Tu
14	Fr	Sa	Mo	Tu	We
15	Sa	Su	Tu	We	Th
16	Su	Mo	We	Th	Fr
17	Mo	Tu	Th	Fr	Sa
18	Tu	We	Fr	Sa	Su
19	We	Th	Sa	Su	Mo
20	Th	Fr	Su	Mo	Tu
21	Fr	Sa	Mo	Tu	We
22	Sa	Su	Tu	We	Th
23	Su	Mo	We	Th	Fr
24	Mo	Tu	Th	Fr	Sa
25	Tu	We	Fr	Sa	Su
26	We	Th	Sa	Su	Mo
27	Th	Fr	Su	Mo	Tu
28	Fr	Sa	Mo	Tu	We
29		Su			

1 FEBRUARY	2
2026 Sun $\vec{p}$	**2026** Mon $\vec{p}$
2027 Mon $\vec{p}$	**2027** Tue $\vec{p}$
2028 Tue $\vec{p}$	**2028** Wed $\vec{p}$
2029 Thu $\vec{p}$	**2029** Fri $\vec{p}$
2030 Fri $\vec{p}$	**2030** Sat $\vec{p}$
2031 Sat $\vec{p}$	**2031** Sun $\vec{p}$
2032 Sun $\vec{p}$	**2032** Mon $\vec{p}$
2033 Tue $\vec{p}$	**2033** Wed $\vec{p}$
2034 Wed $\vec{p}$	**2034** Thu $\vec{p}$
2035 Thu $\vec{p}$	**2035** Fri $\vec{p}$

3

2026
Tue

2027
Wed

2028
Thu

2029
Sat

2030
Sun

2031
Mon

2032
Tue

2033
Thu

2034
Fri

2035
Sat

4 FEBRUARY

2026
Wed

2027
Thu

2028
Fri

2029
Sun

2030
Mon

2031
Tue

2032
Wed

2033
Fri

2034
Sat

2035
Sun

29 FEBRUARY

2028
Tue

○
○

$\vec{p}$

2032
Sun

○
○

$\vec{p}$

Year / Month Summary

	2026	2027	2028	2029	2030	2031	2032	2033	2034	2035
January										
February										
March										
April										
May										
June										
July										
August										
September										
October										
November										
December										

29 DECEMBER

2026
Tue

$\vec{p}$

2027
Wed

$\vec{p}$

2028
Fri

$\vec{p}$

2029
Sat

$\vec{p}$

2030
Sun

$\vec{p}$

2031
Mon

$\vec{p}$

2032
Wed

$\vec{p}$

2033
Thu

$\vec{p}$

2034
Fri

$\vec{p}$

2035
Sat

$\vec{p}$

30

2026
Wed

$\vec{p}$

2027
Thu

$\vec{p}$

2028
Sat

$\vec{p}$

2029
Sun

$\vec{p}$

2030
Mon

$\vec{p}$

2031
Tue

$\vec{p}$

2032
Thu

$\vec{p}$

2033
Fri

$\vec{p}$

2034
Sat

$\vec{p}$

2035
Sun

$\vec{p}$

31 DECEMBER

2026
Thu
○
○
$\vec{p}$

2027
Fri
○
○
$\vec{p}$

2028
Sun
○
○
$\vec{p}$

2029
Mon
○
○
$\vec{p}$

2030
Tue
○
○
$\vec{p}$

2031
Wed
○
○
$\vec{p}$

2032
Fri
○
○
$\vec{p}$

2033
Sat
○
○
$\vec{p}$

2034
Sun
○
○
$\vec{p}$

2035
Mon
○
○
$\vec{p}$

Source Code: forever_journal.py

```
"""
Forever Journal Generator
-------------------------
Generates a 10-year journal layout in LaTeX format.
Designed for A4 paper with specific margin requirements for hole punching.

Usage:
    python forever_journal.py [--test] [--spread 4up] [--align mirrored]
"""

import datetime
import calendar
import argparse
import os
import shutil
import subprocess

# --- CONFIGURATION: JOURNAL SETTINGS ---
START_YEAR = 2026
NUM_YEARS = 10
NUM_WRITING_LINES = 5
SUNDAYS_RED = True
OUTPUT_DIR = "output"

# --- CONFIGURATION: PAPER & MARGINS ---
# Paper Sizes (mm)
PAPER_SIZES = {
    "US_LETTER": {"w": 215.9, "h": 279.4},
    "JIS_B5":    {"w": 182.0, "h": 257.0},
    "A4":        {"w": 210.0, "h": 297.0}
}

CURRENT_PAPER_KEY = "A4"
PAPER = PAPER_SIZES[CURRENT_PAPER_KEY]

# Physical Margins (mm)
# Bottom margin set to 10mm to prevent printer cutoff
TARGET_MARGIN_INNER = 13
TARGET_MARGIN_OUTER = 5
TARGET_MARGIN_TOP = 5
TARGET_MARGIN_BOTTOM = 10

PAGE_W = PAPER["w"]
PAGE_H = PAPER["h"]

# --- CONFIGURATION: LAYOUT DIMENSIONS ---
# Text Width = Page Width - Inner - Outer
CALC_TEXT_WIDTH = PAGE_W - TARGET_MARGIN_INNER - TARGET_MARGIN_OUTER

# Header height reserved for Day/Month display
HEADER_H = 6

# Width reserved for the Year/Day label column
YEAR_LABEL_WIDTH = 10

# Vertical spacing adjustment for labels to avoid touching the line above
LABEL_Y_SHIFT = -0.8

# Calculate Block Height
# We estimate usable height based on margins to keep layout consistent
ESTIMATED_TEXT_HEIGHT = PAGE_H - TARGET_MARGIN_TOP - TARGET_MARGIN_BOTTOM
USABLE_H = ESTIMATED_TEXT_HEIGHT - HEADER_H - 2
BLOCK_H = USABLE_H / NUM_YEARS


def get_day_of_week(year, month, day):
    """Returns the abbreviated day of the week (e.g., 'Mon') for a given date.""
           "
    try:
        dt = datetime.date(year, month, day)
        return dt.strftime("%a")
    except ValueError:
        return ""


def generate_tex(test_mode=False, spread_mode="2up", align_mode="mirrored",
      no_compile=False, include_source=False, toc_enabled=False):
    """
    Generates the LaTeX source file for the journal.

    Args:
    test_mode (bool): If True, generates a small subset of pages for testing
          .
    spread_mode (str): "2up" (1 day/page) or "4up" (2 days/page).
    align_mode (str): "mirrored" (outer alignment) or "left" (standard
          alignment).
    no_compile (bool): If True, skips automatic PDF compilation.
    include_source (bool): If True, appends the script source code to the
          PDF.
    toc_enabled (bool): If True, includes a Table of Contents.
"""
end_year = START_YEAR + NUM_YEARS - 1
output_base = f"forever_journal_{START_YEAR}_{end_year}"
if test_mode:
    output_base = f"test_{output_base}"

# Ensure output directory exists
os.makedirs(OUTPUT_DIR, exist_ok=True)
output_tex = os.path.join(OUTPUT_DIR, f"{output_base}.tex")

# Determine Days Per Page
DAYS_PER_PAGE = 2 if spread_mode == "4up" else 1

# Test Mode Logic
# We define a helper to check if content should be generated based on
  context.
# We also track physical pages to ensure parity alignment.

# Global counter for physical pages written to the PDF
# Initialized to 0. Writing Title Page (Page 1) makes it 1.
physical_page_count = 0

def ensure_parity(logical_page_num):
    """
    Inserts a blank filler page if the next physical page in the PDF
    does not match the even/odd parity of the target logical page number.
    """
    nonlocal physical_page_count

    # Parity: 1 = Odd, 0 = Even
    target_parity = logical_page_num % 2
    next_physical_parity = (physical_page_count + 1) % 2

    if target_parity != next_physical_parity:
        f.write(r"\mbox{} \newpage" + "\n")
        physical_page_count += 1

def is_test_content(section, month=None, day=None, page_idx=None):
    if not test_mode:
        return True

    if section == "TITLE":
        return True

    if section == "MONTH_SUMMARY":
        # Only Feb Summary
        return month == 2

    if section == "DAILY":
        if month == 2:
            # Feb 1-4
            if day in [1, 2, 3, 4]:
                return True
            # Feb 29 (Leap check)
            if day == 29:
                return True
        if month == 12:
            # Dec 29-31
            if day in [29, 30, 31]:
                return True
        return False

    if section == "YEAR_MONTH_SUMMARY":
        # Only the one after Feb (YM1)
        return month == 2

    if section == "CONTINUATION":
        # First spread (0, 1) and Last page (19 or 20)
        if page_idx in [0, 1, 19, 20]:
            return True
        return False
        if section == "SOURCE":
            return True

        return False

    def should_write_page(page_num):
        # Deprecated in favor of is_test_content, but kept for compatibility
        # with existing calls that haven't been migrated if any.
        # In this refactor, we will replace calls to this function.
        return True

    # Column Layout
    COLUMN_GUTTER = 5  # mm
    if DAYS_PER_PAGE == 2:
        COL_WIDTH = (CALC_TEXT_WIDTH - COLUMN_GUTTER) / 2
    else:
        COL_WIDTH = CALC_TEXT_WIDTH

    with open(output_tex, "w") as f:
        # --- PREAMBLE ---
        f.write(r"""
\documentclass[10pt,twoside]{article}
""")
        # Geometry setup:
        # footskip=1mm pulls footer up; with bottom=10mm, footer sits safely
              from edge.
        f.write(rf"\usepackage[paperwidth={PAGE_W}mm, paperheight={PAGE_H}mm,
              inner={TARGET_MARGIN_INNER}mm, outer={TARGET_MARGIN_OUTER}mm, top
              ={TARGET_MARGIN_TOP}mm, bottom={TARGET_MARGIN_BOTTOM}mm, footskip
              =1mm]{{geometry}}" + "\n")

        f.write(r"""
\usepackage{helvet}
\renewcommand{\familydefault}{\sfdefault}
\usepackage{xcolor}
\usepackage{tikz}
\usepackage{fancyhdr}
\usepackage{listings} % For source code listing
\usepackage{pdflscape} % For landscape pages
\usepackage{multicol} % For multi-column layout

\pagestyle{fancy}
\fancyhf{} % clear all headers and footers
\renewcommand{\headrulewidth}{0pt}
\fancyfoot[C]{\itshape \small \thepage} % Italic page number in center footer

\setlength{\parindent}{0pt}
\setlength{\parskip}{0pt}
\raggedbottom % Prevent underfull vbox warnings and forced vertical stretching

% Color Definitions
\definecolor{guidegray}{gray}{0.6} % Darker guide lines
\definecolor{bordergray}{gray}{0.3} % Darker border lines
\definecolor{textgray}{gray}{0.4}   % Date labels
\definecolor{sundayred}{rgb}{0.8, 0.3, 0.3} % Light red for Sundays

% Code Listing Colors
\definecolor{codegreen}{rgb}{0,0.6,0}
\definecolor{codegray}{rgb}{0.5,0.5,0.5}
\definecolor{codepurple}{rgb}{0.58,0,0.82}
\definecolor{backcolour}{rgb}{0.95,0.95,0.92}
\definecolor{framegray}{gray}{0.9}

\begin{document}
""")

        # --- COVER PAGE ---
        if is_test_content("TITLE"):
            ensure_parity(1)
            f.write(r"\begin{titlepage}" + "\n")
            f.write(r"\label{sec:title}" + "\n")
            f.write(r"\centering" + "\n")
            f.write(r"\vspace*{5cm}" + "\n")
            f.write(r"{\Huge \textbf{Forever Journal} \par}" + "\n")
            f.write(r"\vspace{2cm}" + "\n")
            f.write(rf"{{\Large {START_YEAR} -- {START_YEAR + NUM_YEARS - 1} \
                  par}}" + "\n")

            # ToC Box
            if toc_enabled:
                f.write(r"\begin{tikzpicture}[remember picture, overlay]" + "\n"
              )
        f.write(rf"  \node[anchor=south east, xshift=-{
              TARGET_MARGIN_OUTER}mm, yshift={TARGET_MARGIN_BOTTOM}mm]
              at (current page.south east) {{" + "\n")
        f.write(r"    \begin{minipage}{7cm}" + "\n")
        f.write(r"      \textbf{Table of Contents} \par \vspace{2mm}" +
              "\n")
        f.write(r"      Title Page \dotfill \pageref{sec:title} \\" + "\
              n")
        for m in range(1, 13):
            m_name = calendar.month_name[m]
            # In test mode, only show months that are generated
            if is_test_content("MONTH_SUMMARY", month=m):
                f.write(rf"      {m_name} \dotfill \pageref{{sec:month_{
                      m}}} \\" + "\n")
            else:
                f.write(rf"      {m_name} \dotfill (Skipped) \\" + "\n")

        # Continuation pages are not generated in test mode
        if not test_mode:
            f.write(r"      Continuation Pages \dotfill \pageref{sec:
                  continuation} \\" + "\n")
        else:
            f.write(r"      Continuation Pages \dotfill (Skipped) \\" +
                  "\n")

        if include_source:
            f.write(r"      Source Code \dotfill \pageref{sec:source} \\
                  " + "\n")
        f.write(r"    \end{minipage}" + "\n")
        f.write(r"  };" + "\n")
        f.write(r"\end{tikzpicture}" + "\n")

    f.write(r"\vfill" + "\n")

    # Info Box at Bottom Left
    now_str = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    f.write(r"\begin{tikzpicture}[remember picture, overlay]" + "\n")
    f.write(rf"  \node[anchor=south west, xshift={TARGET_MARGIN_INNER}mm
          , yshift=1cm] at (current page.south west) {{" + "\n")
    f.write(r"    \begin{minipage}{10cm}" + "\n")
    f.write(r"      \small \ttfamily" + "\n")
    f.write(rf"      Start Year: {START_YEAR} \\" + "\n")
    f.write(rf"      Num Years: {NUM_YEARS} \\" + "\n")
    f.write(rf"      Lines/Day: {NUM_WRITING_LINES} \\" + "\n")
    f.write(rf"      Sundays Red: {SUNDAYS_RED} \\" + "\n")
    f.write(rf"      Paper: {CURRENT_PAPER_KEY.replace('_', r'\_')} \\"
          + "\n")
    f.write(rf"      Test Mode: {test_mode} \\" + "\n")
    f.write(rf"      Spread: {spread_mode} ({DAYS_PER_PAGE} day/page) \\
          " + "\n")
    f.write(rf"      Align: {align_mode} \\" + "\n")
    f.write(rf"      Generated: {now_str}" + "\n")
    f.write(r"    \end{minipage}" + "\n")
    f.write(r"  };" + "\n")
    f.write(r"\end{tikzpicture}" + "\n")

    f.write(r"\end{titlepage}" + "\n")
    physical_page_count += 1

# We need a reference leap year to ensure we iterate through Feb 29.
ref_year = START_YEAR
while not calendar.isleap(ref_year):
    ref_year += 1

page_num = 2  # Start on page 2 (Left) after title page

def generate_month_summary(month, page_num):
    """Generates a 2-page summary spread for the month."""
    month_name = calendar.month_name[month]
    days_in_month = calendar.monthrange(ref_year, month)[1]

    # Layout Constants
    ROW_H = 8 # mm
    HEADER_H = 15 # mm

    # Calculate column widths
    # Left page: Day Num + 5 Years
    # Right page: 5 Years
    # We use the full text width

    # Day Number Column Width
    DAY_NUM_W = 10

    # Year Column Width
    # Left Page: (TextWidth - DayNumW) / 5
    # Right Page: TextWidth / 5 ? Or keep consistent?
    # Let's keep year columns consistent width across both pages.
    # So we base it on the Left Page constraint.
    YEAR_COL_W = (CALC_TEXT_WIDTH - DAY_NUM_W) / 5

    # Loop for 2 pages (Left/Right)
    for page_idx in range(2):
        if is_test_content("MONTH_SUMMARY", month=month):
            ensure_parity(page_num)
            f.write(rf"\setcounter{{page}}{{{page_num}}}" + "\n")

            # Add Label for ToC (Only on first page of summary)
            if page_idx == 0:
                f.write(rf"\label{{sec:month_{month}}}" + "\n")

            # Determine year range for this page
            if page_idx == 0: # Left Page
                year_range = range(START_YEAR, START_YEAR + 5)
                is_left_page = True
            else: # Right Page
                year_range = range(START_YEAR + 5, START_YEAR + 10)
                is_left_page = False

            f.write(r"\begin{center}" + "\n")
            f.write(rf"{{\Large \textbf{{{month_name} Summary}}}}" + "\n
                  ")
            f.write(r"\end{center}" + "\n")

            f.write(r"\vspace{5mm}" + "\n")

            # TikZ Grid
            # Height = (days_in_month + 1 header) * ROW_H
            grid_h = (days_in_month + 1) * ROW_H

            f.write(rf"\begin{{tikzpicture}}[x=1mm, y=1mm]" + "\n")

            # Draw Horizontal Lines
            # We need lines from index 0 (top) to days_in_month + 1 (
                  bottom)
            # Total rows = days_in_month + 1 (header)
            # Total lines = days_in_month + 2
            w = DAY_NUM_W + 5 * YEAR_COL_W

            for d in range(days_in_month + 2):
                y = grid_h - (d * ROW_H)
                f.write(rf"\draw[bordergray] (0, {y}) -- ({w}, {y});" +
                      "\n")

            # Draw Vertical Lines
            # Left Border
            f.write(rf"\draw[bordergray] (0, 0) -- (0, {grid_h});" + "\n
                  ")
            # Day Num Separator
            f.write(rf"\draw[bordergray] ({DAY_NUM_W}, 0) -- ({DAY_NUM_W
                  }, {grid_h});" + "\n")
            # Year Columns
            for i in range(5):
                x = DAY_NUM_W + (i + 1) * YEAR_COL_W
                f.write(rf"\draw[bordergray] ({x}, 0) -- ({x}, {grid_h})
                      ;" + "\n")

            # --- CONTENT ---

            # 1. Day Numbers (Column 0)
            # Rows 1 to days_in_month
            for day in range(1, days_in_month + 1):
                # Row 0 is Header. Row 1 is Day 1.
                # y_top of Row 1 is grid_h - ROW_H
                # y_center of Row 1 is grid_h - 1.5 * ROW_H
                y_center = grid_h - (day * ROW_H) - (ROW_H / 2)
                f.write(rf"\node[anchor=center] at ({DAY_NUM_W/2}, {
                      y_center}) {{\small \textbf{{{day}}}}};" + "\n")

            # 2. Year Headers (Row 0)
            header_y = grid_h - (ROW_H / 2)
            for i in range(5):
                curr_year = year_range[i]
                header_x = DAY_NUM_W + (i * YEAR_COL_W) + (YEAR_COL_W /
                      2)
                f.write(rf"\node[anchor=center] at ({header_x}, {
                      header_y}) {{\textbf{{{curr_year}}}}};" + "\n")

            # 3. Day Cells (Rows 1 to days_in_month)
            for day in range(1, days_in_month + 1):
                row_top_y = grid_h - (day * ROW_H)

                for i in range(5):
                    curr_year = year_range[i]
                    col_left_x = DAY_NUM_W + (i * YEAR_COL_W)
                    dow = get_day_of_week(curr_year, month, day)[:2]
                    color_cmd = r"\color{sundayred}" if dow == "Su" and
                          SUNDAYS_RED else ""

                    # Top Left Corner
                    f.write(rf"\node[anchor=north west, inner sep=1pt]
                          at ({col_left_x + 1}, {row_top_y - 1}) {{\tiny
                           {color_cmd} {dow}}};" + "\n")

            f.write(r"\end{tikzpicture}" + "\n")
            f.write(r"\newpage" + "\n")
            nonlocal physical_page_count
            physical_page_count += 1

        page_num += 1

    return page_num

def generate_year_month_summary(month, page_num):
    """
    Generates a Year/Month summary grid in landscape orientation.
    Rows: Months (Jan-Dec)
    Cols: Years (Start-End)
    """
    if is_test_content("YEAR_MONTH_SUMMARY", month=month):
        ensure_parity(page_num)
        f.write(rf"\setcounter{{page}}{{{page_num}}}" + "\n")
        f.write(r"\begin{landscape}" + "\n")

        # Title
        f.write(r"\begin{center}" + "\n")
        f.write(r"{\Large \textbf{Year / Month Summary}} \par" + "\n")
        f.write(r"\end{center}" + "\n")
        f.write(r"\vspace{2mm}" + "\n")

        f.write(r"\begin{tikzpicture}[x=1mm, y=1mm]" + "\n")

        # Dimensions
        # Landscape A4: Width ~270mm (Long edge), Height ~190mm (Short
              edge)
        # We have 10 Years + 1 Label Column
        # We have 12 Months + 1 Header Row

        SUMMARY_MONTH_COL_W = 20
        SUMMARY_YEAR_COL_W = 24
        SUMMARY_ROW_H = 14
        HEADER_ROW_H = 6

        GRID_W = SUMMARY_MONTH_COL_W + (NUM_YEARS * SUMMARY_YEAR_COL_W)
        GRID_H = (12 * SUMMARY_ROW_H) + HEADER_ROW_H

        # Draw Grid
        # Horizontal Lines
        # Top
        f.write(rf"\draw[bordergray] (0, {GRID_H}) -- ({GRID_W}, {GRID_H
              });" + "\n")
        # Header Line
        f.write(rf"\draw[bordergray] (0, {GRID_H - HEADER_ROW_H}) -- ({
              GRID_W}, {GRID_H - HEADER_ROW_H});" + "\n")
        # Rows
        for m in range(1, 13):
            y = GRID_H - HEADER_ROW_H - (m * SUMMARY_ROW_H)
            f.write(rf"\draw[bordergray] (0, {y}) -- ({GRID_W}, {y});" +
                  "\n")

        # Vertical Lines
        # Left Border
        f.write(rf"\draw[bordergray] (0, 0) -- (0, {GRID_H});" + "\n")
        # Month Col Separator
        f.write(rf"\draw[bordergray] ({SUMMARY_MONTH_COL_W}, 0) -- ({
              SUMMARY_MONTH_COL_W}, {GRID_H});" + "\n")
        # Year Columns
        for i in range(NUM_YEARS):
            x = SUMMARY_MONTH_COL_W + ((i + 1) * SUMMARY_YEAR_COL_W)
            f.write(rf"\draw[bordergray] ({x}, 0) -- ({x}, {GRID_H});" +
                  "\n")

        # --- CONTENT ---

        # 1. Year Headers (Row 0)
        header_y = GRID_H - (HEADER_ROW_H / 2)
        for i in range(NUM_YEARS):
            curr_year = START_YEAR + i
            header_x = SUMMARY_MONTH_COL_W + (i * SUMMARY_YEAR_COL_W) +
                  (SUMMARY_YEAR_COL_W / 2)
            f.write(rf"\node[anchor=center] at ({header_x}, {header_y})
                  {{\textbf{{{curr_year}}}}};" + "\n")
        # 2. Month Labels (Column 0)
        for m in range(1, 13):
            m_name = calendar.month_name[m]
            y_center = GRID_H - HEADER_ROW_H - ((m - 1) * SUMMARY_ROW_H)
                    - (SUMMARY_ROW_H / 2)
            f.write(rf"\node[anchor=center] at ({SUMMARY_MONTH_COL_W/2},
                    {y_center}) {{\textbf{{{m_name}}}}};" + "\n")

        # 3. Guide Lines in Cells
        # 3 lines per cell
        line_spacing = SUMMARY_ROW_H / 4
        for m in range(1, 13):
            row_top_y = GRID_H - HEADER_ROW_H - ((m - 1) * SUMMARY_ROW_H
                   )
            for l in range(1, 4):
                y_line = row_top_y - (l * line_spacing)
                f.write(rf"\draw[guidegray, dash pattern=on 0.5pt off 1
                       pt] ({SUMMARY_MONTH_COL_W}, {y_line}) -- ({GRID_W
                       }, {y_line});" + "\n")

        f.write(r"\end{tikzpicture}" + "\n")
        f.write(r"\end{landscape}" + "\n")
        f.write(r"\newpage" + "\n")
        nonlocal physical_page_count
        physical_page_count += 1

    return page_num + 1

# Iterate through months to ensure proper pagination (Start Month on
      Left Page)
for month in range(1, 13):
    # Collect days for this month
    month_days = []
    days_in_month = calendar.monthrange(ref_year, month)[1]
    for day in range(1, days_in_month + 1):
        month_days.append((month, day))

    if not month_days:
        continue

    # Ensure we start on an Even (Left) page for the new month
    if page_num % 2 != 0:
        if is_test_content("MONTH_SUMMARY", month=month):
            ensure_parity(page_num)
            f.write(rf"\setcounter{{page}}{{{page_num}}}" + "\n")
            f.write(r"\mbox{} \newpage" + "\n")
            physical_page_count += 1
        page_num += 1

    # --- MONTH SUMMARY SPREAD ---
    # Insert the 2-page summary before the daily pages
    page_num = generate_month_summary(month, page_num)

    # Iterate through days in chunks
    for i in range(0, len(month_days), DAYS_PER_PAGE):
        chunk = month_days[i:i + DAYS_PER_PAGE]

        # Check if we should generate this page
        is_chunk_test = False
        if not test_mode:
            is_chunk_test = True
        else:
            for _, d in chunk:
                if is_test_content("DAILY", month=month, day=d):
                    is_chunk_test = True
                    break

        if not is_chunk_test:
            page_num += 1
            continue

        ensure_parity(page_num)
        f.write(rf"\setcounter{{page}}{{{page_num}}}" + "\n")

        for col_idx, (month, day) in enumerate(chunk):
            month_name = calendar.month_name[month].upper()

            # Separator between columns
            if col_idx > 0:
                f.write(r"\hfill" + "\n")

            # Start Column Minipage
            f.write(rf"\begin{{minipage}}[t]{{{COL_WIDTH}mm}}" + "\n")

            # Determine Alignment for this column
            align_right = False

            # Determine if this is an Inner or Outer column
            # Even Page (Left): Col 0 = Outer, Col 1 = Inner
            # Odd Page (Right): Col 0 = Inner, Col 1 = Outer
            is_inner_col = False
            if page_num % 2 == 0:  # Even
                if col_idx == 1:
                    is_inner_col = True
            else:  # Odd
                if col_idx == 0:
                    is_inner_col = True

            if align_mode == "mirrored":
                if page_num % 2 != 0:  # Odd/Right Page
                    align_right = True
                else:  # Even/Left Page
                    align_right = False
            elif align_mode == "left":
                align_right = False

            # --- HEADER LOGIC ---
            f.write(rf"\begin{{minipage}}[t][{HEADER_H}mm]{{\textwidth}}
                  ")

            # Determine content parts
            day_str = rf"\huge \textbf{{{day}}}"
            month_str = rf"\huge \textbf{{{month_name}}}"

            # Determine if we show month
            show_month = True
            if DAYS_PER_PAGE == 2 and is_inner_col:
                # Generally hide month on inner columns to reduce
                      clutter
                show_month = False
                # EXCEPTION: Always show month on the last day of the
                      month
                if day == days_in_month:
                    show_month = True

            # Build the header line
            if align_right:
                # Labels on Right (Right Page)
                f.write(r"\hfill ")
                if show_month:
                    f.write(rf"{month_str} \quad ")
                f.write(rf"\makebox[{YEAR_LABEL_WIDTH}mm][r]{{{day_str
                      }}}")
            else:
                # Labels on Left (Left Page)
                f.write(rf"\makebox[{YEAR_LABEL_WIDTH}mm][l]{{{day_str
                      }}}")
                if show_month:
                    f.write(rf" \quad {month_str}")
                f.write(r" \hfill")

            f.write(r"\end{minipage}")
            f.write(r"\par \nointerlineskip")

            # --- 10 YEAR BLOCKS ---
            for y_idx in range(NUM_YEARS):
                curr_year = START_YEAR + y_idx
                weekday = get_day_of_week(curr_year, month, day)

                is_leap_year = calendar.isleap(curr_year)
                is_feb_29 = (month == 2 and day == 29)
                skip_content = is_feb_29 and not is_leap_year

                if not skip_content:
                    label_year = f"{curr_year}"
                    label_day = f"{weekday}"
                    if SUNDAYS_RED and weekday == "Sun":
                        day_color = "sundayred"
                    else:
                        day_color = "textgray"

                # --- DRAW THE BLOCK ---
                f.write(rf"\begin{{tikzpicture}}[x=1mm, y=1mm, trim left
                      =0mm, trim right={COL_WIDTH}mm]" + "\n")

                w = COL_WIDTH
                h = BLOCK_H

                f.write(rf"\path[use as bounding box] (0,0) rectangle ({
                      w}, {h});" + "\n")

                line_spacing = h / NUM_WRITING_LINES

                if not skip_content:
                    # Align labels to match header alignment
                    if align_right:
                        f.write(rf"\node[anchor=north east, text width={
                              YEAR_LABEL_WIDTH}mm, align=right, inner
                              sep=0pt, yshift={LABEL_Y_SHIFT}mm] at ({w
                              },{h}) {{\textbf{{{label_year}}}\\ \small
                              \color{{{day_color}}} {label_day}}};" + "\
                              n")
                    else:
                        f.write(rf"\node[anchor=north west, text width={
                              YEAR_LABEL_WIDTH}mm, align=left, inner sep
                              =0pt, yshift={LABEL_Y_SHIFT}mm] at (0,{h})
                               {{\textbf{{{label_year}}}\\ \small \color
                              {{{day_color}}} {label_day}}};" + "\n")

                # Top Border (First block only)
                if y_idx == 0:
                    f.write(rf"\draw[bordergray] (0, {h}) -- ({w}, {h});
                          " + "\n")

                # Guide Lines
                if not skip_content:
                    guide_gap = YEAR_LABEL_WIDTH + 1

                    # Circles for first two lines (Inside end)
                    circle_radius = line_spacing * 0.25
                    for s in range(2):  # First two spaces
                        y_circle = h - (s + 0.5) * line_spacing
                        if align_right:  # Inner is Left
                            cx = circle_radius + 1
                        else:  # Inner is Right
                            cx = w - circle_radius - 1
                        f.write(rf"\draw[guidegray] ({cx}, {y_circle})
                              circle ({circle_radius});" + "\n")

                    # Continuation 'p' prompt
                    f.write(rf"\node[anchor=base east, inner sep=0, text
                          =textgray] at ({w}-6, 2.5) {{\small $\vec{{p}}
                          $}};" + "\n")

                    for l in range(1, NUM_WRITING_LINES):
                        y_pos = h - l * line_spacing
                        if l == 1:
                            # Shortened Guide Line
                            if align_right:
                                f.write(rf"\draw[guidegray, dash pattern
                                      =on 0.5pt off 1pt] (0, {y_pos}) --
                                       ({w} - {guide_gap}, {y_pos});" +
                                      "\n")
                            else:
                                f.write(rf"\draw[guidegray, dash pattern
                                      =on 0.5pt off 1pt] ({guide_gap}, {
                                      y_pos}) -- ({w}, {y_pos});" + "\n"
                                      )
                        else:
                            f.write(rf"\draw[guidegray, dash pattern=on
                                  0.5pt off 1pt] (0, {y_pos}) -- ({w}, {
                                  y_pos});" + "\n")

                # Bottom Divider
                f.write(rf"\draw[bordergray] (0, 0) -- ({w}, 0);" + "\n"
                      )

                f.write(r"\end{tikzpicture}" + "\n")
                f.write(r"\par \nointerlineskip" + "\n")

            # End Column Minipage
            f.write(r"\end{minipage}" + "\n")

        # End of Page Chunk
        f.write(r"\newpage" + "\n")
        physical_page_count += 1
        page_num += 1

    # --- YEAR/MONTH SUMMARY (For Short Months) ---
    # Feb, Apr, Jun, Sep, Nov
    if month in [2, 4, 6, 9, 11]:
        page_num = generate_year_month_summary(month, page_num)

# --- CONTINUATION PAGES ---
# 20 pages (10 sheets) of lined notes
# We ensure the Source Code starts on an Odd page (Right side / Fresh
      sheet).
# If after 20 pages, the next page is Even, we add one more continuation
       page.
MIN_CONTINUATION_PAGES = 20

# Calculate how many pages we need
# Current page_num is the start of continuation.
# If (page_num + 20) is Even, next page is Even. We want Odd. So we need
```

```
               21.
        # If (page_num + 20) is Odd, next page is Odd. Good. We need 20.
        if (page_num + MIN_CONTINUATION_PAGES) % 2 == 0:
            num_continuation_pages = MIN_CONTINUATION_PAGES + 1
        else:
            num_continuation_pages = MIN_CONTINUATION_PAGES

        # Calculate lines for full page
        line_spacing = BLOCK_H / NUM_WRITING_LINES

        # Usable height for continuation pages
        CONT_USABLE_H = ESTIMATED_TEXT_HEIGHT - HEADER_H - 10

        num_lines_cont = int(CONT_USABLE_H / line_spacing)

        for i in range(num_continuation_pages):
            if is_test_content("CONTINUATION", page_idx=i):
                ensure_parity(page_num)
                f.write(rf"\setcounter{{page}}{{{page_num}}}" + "\n")

                if i == 0:
                    f.write(r"\label{sec:continuation}" + "\n")

                # Header (Empty, just spacing to match main pages)
                f.write(rf"\begin{{minipage}}[t][{HEADER_H}mm]{{\textwidth}}")
                f.write(r"\mbox{}")
                f.write(r"\end{minipage}")
                f.write(r"\par \nointerlineskip")

                # Full page lines
                f.write(rf"\begin{{tikzpicture}}[x=1mm, y=1mm]" + "\n")
                w_cont = CALC_TEXT_WIDTH
                h_cont = CONT_USABLE_H

                f.write(rf"\path[use as bounding box] (0,0) rectangle ({w_cont},
                        {h_cont});" + "\n")

                for l in range(1, num_lines_cont):
                    y_pos = h_cont - l * line_spacing
                    f.write(rf"\draw[guidegray, dash pattern=on 0.5pt off 1pt]
                           (0, {y_pos}) -- ({w_cont}, {y_pos});" + "\n")

                # Bottom Border
                f.write(rf"\draw[bordergray] (0, 0) -- ({w_cont}, 0);" + "\n")

                f.write(r"\end{tikzpicture}")
                f.write(r"\newpage" + "\n")
                physical_page_count += 1

            page_num += 1

        # --- SOURCE CODE APPENDIX ---
        # Self-preservation: Print the source code of this script at the end of
              the journal.
        if include_source and is_test_content("SOURCE"):
            ensure_parity(page_num)
            # Ensure the page number is correct (continuing from the last
                  logical page)
            f.write(rf"\setcounter{{page}}{{{page_num}}}" + "\n")

            # Reset geometry to maximize space for code (this forces a new page)
            # Respect inner margin for binding/hole punches
            f.write(rf"\newgeometry{{top=10mm, bottom=10mm, inner={
                  TARGET_MARGIN_INNER}mm, outer=10mm}}" + "\n")

            # Landscape mode for source code
            f.write(r"\begin{landscape}" + "\n")
            f.write(r"\section*{Source Code: forever\_journal.py}" + "\n")
            f.write(r"\label{sec:source}" + "\n")

            # Configure listings
            f.write(r"\lstset{" + "\n")
            f.write(r"  language=Python," + "\n")
            f.write(r"  basicstyle=\tiny\ttfamily," + "\n")
            f.write(r"  keywordstyle=\color{blue}," + "\n")
            f.write(r"  stringstyle=\color{codepurple}," + "\n")
            f.write(r"  commentstyle=\color{codegreen}," + "\n")
            f.write(r"  breaklines=true," + "\n")
            f.write(r"  showstringspaces=false," + "\n")
            f.write(r"  numbers=none," + "\n")
            f.write(r"  frame=single," + "\n")
            f.write(r"  rulecolor=\color{lightgray}" + "\n")
            f.write(r"}" + "\n")

            # 3 Columns
            f.write(r"\begin{multicols}{3}" + "\n")
            f.write(r"\begin{lstlisting}" + "\n")

            # Read and write the source code of this file
            # We must be careful not to print the end-listing tag literally, or
                  it will break the LaTeX.
            try:
                with open(os.path.abspath(__file__), "r") as source_file:
                    for line in source_file:
                        f.write(line)
            except Exception as e:
                f.write(f"# Error reading source code: {e}")

            # Safe way to write the end tag without breaking the listing
            f.write(r"\end{lst" + "listing}" + "\n")
            f.write(r"\end{multicols}" + "\n")
            f.write(r"\end{landscape}" + "\n")

        f.write(r"\end{document}")

    print(f"Generated: {output_tex}")
    print(f"Configuration: Paper={CURRENT_PAPER_KEY} ({PAGE_W}x{PAGE_H}mm)")
    print(f"Margins: Inner={TARGET_MARGIN_INNER}mm, Outer={TARGET_MARGIN_OUTER}
          mm, Top={TARGET_MARGIN_TOP}mm, Bottom={TARGET_MARGIN_BOTTOM}mm")
    print(f"Layout: {spread_mode} ({DAYS_PER_PAGE} days/page), Align: {
          align_mode}")

    # --- AUTO-COMPILE LOGIC ---
    if not no_compile:
        pdflatex_path = shutil.which("pdflatex")
        if pdflatex_path:
            print(f"Found pdflatex at: {pdflatex_path}")
            print("Compiling PDF...")
            try:
                # Run pdflatex with output directory
                # Note: We pass the full path to the tex file.
                # pdflatex will write aux/log/pdf to the directory specified by
                      -output-directory
                cmd = [
                    pdflatex_path,
                    f"-output-directory={OUTPUT_DIR}",
                    "-interaction=nonstopmode", # Don't hang on errors
                    output_tex
                ]

                # Run twice to resolve references (ToC page numbers) if ToC is
                      enabled
                if toc_enabled:
                    print("Pass 1/2...")
                    subprocess.run(cmd, check=True)

                    print("Pass 2/2 (Resolving references)...")
                    subprocess.run(cmd, check=True)
                else:
                    print("Compiling...")
                    subprocess.run(cmd, check=True)

                print(f"Success! PDF generated at: {os.path.join(OUTPUT_DIR,
                      output_base + '.pdf')}")
            except subprocess.CalledProcessError as e:
                print("Error during PDF compilation.")
                print(e)
        else:
            print("\n[NOTICE] pdflatex not found in PATH.")
            print("To generate the PDF, please install a LaTeX distribution (e.g
                  ., TeX Live, MacTeX).")
            print(f"Then run: pdflatex -output-directory {OUTPUT_DIR} {
                  output_tex}")
    else:
        print(f"Skipping compilation. To compile manually: pdflatex -output-
              directory {OUTPUT_DIR} {output_tex}")


if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Generate Forever Journal LaTeX
          ")
    parser.add_argument("--test", action="store_true", help="Generate a test PDF
           with specific leap year spreads")
    parser.add_argument("--spread", choices=["2up", "4up"], default="2up", help=
          "2up = 1 day/page, 4up = 2 days/page")
    parser.add_argument("--align", choices=["mirrored", "left"], default="
          mirrored", help="mirrored = Outer aligned, left = Left aligned")
    parser.add_argument("--no-compile", action="store_true", help="Skip
          automatic PDF compilation")
    parser.add_argument("--include-source", action="store_true", help="Append
          source code to the PDF")
    parser.add_argument("--toc", action="store_true", help="Include Table of
          Contents (requires 2-pass compilation)")
    args = parser.parse_args()

    generate_tex(test_mode=args.test, spread_mode=args.spread, align_mode=args.
          align, no_compile=args.no_compile, include_source=args.include_source,
           toc_enabled=args.toc)
```