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: 2025-12-22 23:55:31

1 JANUARY

2026
Thu

2027
Fri

2028
Sat

2029
Mon

2030
Tue

2031
Wed

2032
Thu

2033
Sat

2034
Sun

2035
Mon

2

2026
Fri

2027
Sat

2028
Sun

2029
Tue

2030
Wed

2031
Thu

2032
Fri

2033
Sun

2034
Mon

2035
Tue

3

2026
Sat

2027
Sun

2028
Mon

2029
Wed

2030
Thu

2031
Fri

2032
Sat

2033
Mon

2034
Tue

2035
Wed

4 JANUARY

2026
Sun

2027
Mon

2028
Tue

2029
Thu

2030
Fri

2031
Sat

2032
Sun

2033
Tue

2034
Wed

2035
Thu

5 JANUARY

2026
Mon

2027
Tue

2028
Wed

2029
Fri

2030
Sat

2031
Sun

2032
Mon

2033
Wed

2034
Thu

2035
Fri

6

2026
Tue

2027
Wed

2028
Thu

2029
Sat

2030
Sun

2031
Mon

2032
Tue

2033
Thu

2034
Fri

2035
Sat

27

2026
Fri

2027
Sat

2028
Sun

2029
Tue

2030
Wed

2031
Thu

2032
Fri

2033
Sun

2034
Mon

2035
Tue

28 FEBRUARY

2026
Sat

2027
Sun

2028
Mon

2029
Wed

2030
Thu

2031
Fri

2032
Sat

2033
Mon

2034
Tue

2035
Wed

29 FEBRUARY

2028
Tue

$\vec{p}$

2032
Sun

$\vec{p}$

1 MARCH

2026
Sun

2027
Mon

2028
Wed

2029
Thu

2030
Fri

2031
Sat

2032
Mon

2033
Tue

2034
Wed

2035
Thu

2

2026
Mon

2027
Tue

2028
Thu

2029
Fri

2030
Sat

2031
Sun

2032
Tue

2033
Wed

2034
Thu

2035
Fri

31 DECEMBER

2026
Thu

2027
Fri

2028
Sun

2029
Mon

2030
Tue

2031
Wed

2032
Fri

2033
Sat

2034
Sun

2035
Mon

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):
    """
    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.
    """
    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")

    # Test Page Ranges (Inclusive)
    # 1-4: Title, Jan 1-3
    # 31-34: Feb/Mar transition
    # 193-196: Dec/Continuation transition
    TEST_PAGE_RANGES = [(1, 4), (31, 34), (193, 196)]

    def should_write_page(p):
        if not test_mode:
            return True
        for start, end in TEST_PAGE_RANGES:
            if start <= p <= end:
                return True
        return False

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

    # 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

\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

\begin{document}
""")

        # --- COVER PAGE ---
        if should_write_page(1):
            f.write(r"\begin{titlepage}" + "\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")
            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(r"  \node[anchor=south west, xshift=1cm, 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")

        # 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

        # 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 should_write_page(page_num):
                    f.write(rf"\setcounter{{page}}{{{page_num}}}" + "\n")
                    f.write(r"\mbox{} \newpage" + "\n")
                page_num += 1

            # Iterate through days in chunks
            for i in range(0, len(month_days), DAYS_PER_PAGE):
                if not should_write_page(page_num):
                    page_num += 1
                    continue

                f.write(rf"\setcounter{{page}}{{{page_num}}}" + "\n")
                chunk = month_days[i:i + DAYS_PER_PAGE]

                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=0
          mm, 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=0
                  pt, 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 0.5
                      pt 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")
        page_num += 1

# --- CONTINUATION PAGES ---
# 20 pages (10 sheets) of lined notes
NUM_CONTINUATION_PAGES = 20

# 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 _ in range(NUM_CONTINUATION_PAGES):
    if should_write_page(page_num):
        f.write(rf"\setcounter{{page}}{{{page_num}}}" + "\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")

    page_num += 1

# --- SOURCE CODE APPENDIX ---
# Self-preservation: Print the source code of this script at the end of the
      journal.
if include_source:
    # 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)
    f.write(r"\newgeometry{top=10mm, bottom=10mm, left=10mm, right=10mm}" +
          "\n")
    f.write(r"\twocolumn" + "\n")
    f.write(r"\section*{Source Code: forever\_journal.py}" + "\n")

    # Configure listings
    f.write(r"\lstset{" + "\n")
    f.write(r"  language=Python," + "\n")
    f.write(r"  basicstyle=\tiny\ttfamily," + "\n") # Tiny font to fit ~450
          lines
    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"}" + "\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{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
            ]
            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="2
          up = 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")
    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)
```