```
"""
PDF to Audiobook Converter
==========================
Converts PDF books into audio files using:
- PyMuPDF for text extraction
- Edge-TTS for text-to-speech (free, no API key needed)

Usage:
    python pdf_to_audiobook.py "my_book.pdf"
    python pdf_to_audiobook.py "my_book.pdf" --voice "en-US-AndrewNeural"
    python pdf_to_audiobook.py "my_book.pdf" --start-page 10 --end-page 50
    python pdf_to_audiobook.py "my_book.pdf" --chapter-mode
    python pdf_to_audiobook.py --list-voices

Requirements:
    pip install pymupdf edge-tts pydub
    Also needs ffmpeg installed (for merging audio chunks):
        - Windows: choco install ffmpeg  OR  download from ffmpeg.org
        - Mac: brew install ffmpeg
        - Linux: sudo apt install ffmpeg
"""

import argparse
import asyncio
import os
import re
import sys
import time
from pathlib import Path

try:
    import fitz  # PyMuPDF
except ImportError:
    sys.exit("Missing dependency: pip install pymupdf")

try:
    import edge_tts
except ImportError:
    sys.exit("Missing dependency: pip install edge-tts")


# ── Popular Edge-TTS Voices ──────────────────────────────────────
# These sound great for audiobooks. Use --list-voices to see all options.
RECOMMENDED_VOICES = {
```

```
    "andrew": "en-US-AndrewNeural",       # Male, warm, great for narration
    "ava": "en-US-AvaNeural",             # Female, clear and natural
    "brian": "en-US-BrianNeural",          # Male, deep and smooth
    "emma": "en-US-EmmaNeural",            # Female, friendly
    "jenny": "en-US-JennyNeural",          # Female, conversational
    "guy": "en-US-GuyNeural",              # Male, professional
    "aria": "en-US-AriaNeural",            # Female, expressive
    "davis": "en-US-DavisNeural",          # Male, calm narrator style
    "sonia": "en-GB-SoniaNeural",          # Female, British accent
    "ryan": "en-GB-RyanNeural",            # Male, British accent
}

DEFAULT_VOICE = "en-US-AndrewNeural"


# ── Text Extraction ──────────────────────────────────────────────────

def extract_text_from_pdf(pdf_path: str, start_page: int = None, end_page: int =
    """
    Extract text from a PDF, returning a list of {page_number, text} dicts.
    """
    doc = fitz.open(pdf_path)
    total_pages = len(doc)

    start = (start_page - 1) if start_page else 0
    end = min(end_page, total_pages) if end_page else total_pages

    print(f"📖 Extracting text from pages {start + 1} to {end} (of {total_pages}

    pages = []
    for i in range(start, end):
        page = doc[i]
        text = page.get_text("text")
        if text.strip():
            pages.append({"page_number": i + 1, "text": text})

    doc.close()
    return pages


def clean_text(text: str) -> str:
    """
    Clean extracted PDF text for better TTS output.
    Removes artifacts, fixes spacing, and normalizes formatting.
    """
    # Remove page headers/footers (common patterns)
    lines = text.split("\n")
```

```
    cleaned_lines = []

    for line in lines:
        stripped = line.strip()

        # Skip likely headers/footers
        if not stripped:
            cleaned_lines.append("")
            continue

        # Skip standalone page numbers
        if re.match(r'^\d{1,4}$', stripped):
            continue

        # Skip very short lines that look like headers (all caps, < 5 words)
        # but keep them as section markers
        if stripped.isupper() and len(stripped.split()) <= 8 and len(stripped) >
            cleaned_lines.append(f"\n{stripped}\n")
            continue

        cleaned_lines.append(stripped)

    text = " ".join(cleaned_lines)

    # Fix common PDF extraction issues
    text = re.sub(r'(\w)-\s+(\w)', r'\1\2', text)          # Fix hyphenated line
    text = re.sub(r'\s{2,}', ' ', text)                     # Collapse multiple s
    text = re.sub(r'\n{3,}', '\n\n', text)                  # Limit consecutive n
    text = re.sub(r'(?<=[.!?])\s*\n\s*(?=[A-Z])', '. ', text)  # Join broken sent
    text = re.sub(r'\s+([.,;:!?])', r'\1', text)            # Fix space before pu

    # Add pauses for TTS at paragraph breaks
    text = text.replace('\n\n', '. ... ')
    text = text.replace('\n', ' ')

    # Clean up any remaining artifacts
    text = re.sub(r'\.{4,}', '...', text)                   # Limit ellipsis
    text = re.sub(r'\s{2,}', ' ', text)                     # Final space cleanup

    return text.strip()


def split_into_chunks(text: str, max_chars: int = 4000) -> list[str]:
    """
    Split text into chunks suitable for TTS processing.
    Edge-TTS handles up to ~5000 chars well, but smaller chunks are more reliable
    Splits at sentence boundaries to avoid cutting mid-sentence.
```

```
    """
    sentences = re.split(r'(?<=[.!?])\s+', text)
    chunks = []
    current_chunk = ""

    for sentence in sentences:
        if len(current_chunk) + len(sentence) + 1 > max_chars:
            if current_chunk:
                chunks.append(current_chunk.strip())
            current_chunk = sentence
        else:
            current_chunk += " " + sentence if current_chunk else sentence

    if current_chunk.strip():
        chunks.append(current_chunk.strip())

    return chunks


# ─── Audio Generation ──────────────────────────────────────────────

async def text_to_audio(text: str, output_path: str, voice: str = DEFAULT_VOICE,
    """
    Convert a text chunk to an MP3 file using Edge-TTS.
    """
    communicate = edge_tts.Communicate(text, voice, rate=rate)
    await communicate.save(output_path)


async def generate_audiobook(
    pages: list[dict],
    output_dir: str,
    voice: str = DEFAULT_VOICE,
    rate: str = "+0%",
    chapter_mode: bool = False
):
    """
    Generate audio files from extracted PDF pages.
    """
    os.makedirs(output_dir, exist_ok=True)

    # Combine all page text
    full_text = ""
    for page in pages:
        cleaned = clean_text(page["text"])
        if cleaned:
            full_text += cleaned + " ... "  # Add pause between pages
```

```
    if not full_text.strip():
        print("❌ No text could be extracted from the PDF.")
        return []

    total_chars = len(full_text)
    print(f"📝 Total text extracted: {total_chars:,} characters (~{total_chars //

    # Estimate audio duration (Edge-TTS speaks ~150 words/min)
    est_minutes = (total_chars // 5) / 150
    est_hours = est_minutes / 60
    print(f"⏱️  Estimated audio duration: {int(est_hours)}h {int(est_minutes % 60

    # Split into chunks
    chunks = split_into_chunks(full_text)
    total_chunks = len(chunks)
    print(f"🔊 Generating audio in {total_chunks} chunks...\n")

    audio_files = []
    start_time = time.time()

    for i, chunk in enumerate(chunks):
        chunk_file = os.path.join(output_dir, f"chunk_{i:04d}.mp3")

        try:
            await text_to_audio(chunk, chunk_file, voice, rate)
            audio_files.append(chunk_file)

            # Progress bar
            progress = (i + 1) / total_chunks
            elapsed = time.time() - start_time
            eta = (elapsed / (i + 1)) * (total_chunks - i - 1) if i > 0 else 0
            bar = "█" * int(progress * 30) + "░" * (30 - int(progress * 30))
            print(f"\r  [{bar}] {progress:.0%} | Chunk {i+1}/{total_chunks} | ETA

        except Exception as e:
            print(f"\n⚠️  Error on chunk {i+1}: {e}")
            continue

    print(f"\n\n✅ Generated {len(audio_files)} audio chunks in {output_dir}/")
    return audio_files


def merge_audio_files(audio_files: list[str], output_path: str):
    """
    Merge multiple MP3 chunks into a single audiobook file.
    Requires pydub and ffmpeg.
```

```
    """
    try:
        from pydub import AudioSegment
    except ImportError:
        print("⚠️  pydub not installed. Skipping merge. Install with: pip install
        print("   Individual chapter files are still available in the output dire
        return False

    print(f"\n🔗 Merging {len(audio_files)} chunks into final audiobook...")

    combined = AudioSegment.empty()
    silence = AudioSegment.silent(duration=500)  # 0.5s pause between chunks

    for i, file_path in enumerate(audio_files):
        if os.path.exists(file_path):
            segment = AudioSegment.from_mp3(file_path)
            combined += segment + silence

            if (i + 1) % 20 == 0:
                print(f"  Merged {i + 1}/{len(audio_files)} chunks...")

    print(f"  Exporting final file ({len(combined) / 1000 / 60:.1f} minutes)...")
    combined.export(output_path, format="mp3", bitrate="128k")

    # Get file size
    size_mb = os.path.getsize(output_path) / (1024 * 1024)
    print(f"✅ Audiobook saved: {output_path} ({size_mb:.1f} MB)")

    return True


def cleanup_chunks(output_dir: str, audio_files: list[str]):
    """Remove individual chunk files after successful merge."""
    for f in audio_files:
        if os.path.exists(f):
            os.remove(f)
    print("🧹 Cleaned up temporary chunk files.")


# ── Voice Listing ──────────────────────────────────────────────────────

async def list_voices(language_filter: str = "en"):
    """List available Edge-TTS voices."""
    voices = await edge_tts.list_voices()

    print(f"\n🎙️  Available voices (filtered: '{language_filter}'):\n")
    print(f"{'Voice Name':<35} {'Gender':<10} {'Locale':<10}")
```

```
    print("—" * 60)

    for v in sorted(voices, key=lambda x: x["ShortName"]):
        if language_filter and not v["Locale"].lower().startswith(language_filter
            continue
        print(f"{v['ShortName']:<35} {v['Gender']:<10} {v['Locale']:<10}")

    print(f"\n⭐ Recommended voices for audiobooks:")
    for name, voice_id in RECOMMENDED_VOICES.items():
        print(f"  --voice \"{voice_id}\"  ({name})")


# ── Main ──────────────────────────────────────────────────────────────────

async def main():
    parser = argparse.ArgumentParser(
        description="📚 PDF to Audiobook Converter — Turn any PDF into an MP3 auc
        formatter_class=argparse.RawDescriptionHelpFormatter,
        epilog="""
Examples:
  python pdf_to_audiobook.py book.pdf
  python pdf_to_audiobook.py book.pdf --voice "en-US-AvaNeural"
  python pdf_to_audiobook.py book.pdf --start-page 1 --end-page 50
  python pdf_to_audiobook.py book.pdf --rate "+10%"    # Speed up narration
  python pdf_to_audiobook.py book.pdf --rate "-10%"    # Slow down narration
  python pdf_to_audiobook.py --list-voices
  python pdf_to_audiobook.py --list-voices --language ko  # Korean voices
        """
    )

    parser.add_argument("pdf", nargs="?", help="Path to the PDF file")
    parser.add_argument("--voice", "-v", default=DEFAULT_VOICE,
                        help=f"TTS voice to use (default: {DEFAULT_VOICE})")
    parser.add_argument("--rate", "-r", default="+0%",
                        help="Speech rate adjustment, e.g. '+10%%' or '-15%%' (de
    parser.add_argument("--start-page", "-s", type=int, default=None,
                        help="Start from this page number")
    parser.add_argument("--end-page", "-e", type=int, default=None,
                        help="End at this page number")
    parser.add_argument("--output", "-o", default=None,
                        help="Output file path (default: <pdf_name>_audiobook.mp3
    parser.add_argument("--no-merge", action="store_true",
                        help="Skip merging into single file (keep individual chun
    parser.add_argument("--keep-chunks", action="store_true",
                        help="Keep individual chunk files after merging")
    parser.add_argument("--list-voices", action="store_true",
                        help="List available TTS voices and exit")
```

```
    parser.add_argument("--language", "-l", default="en",
                        help="Language filter for --list-voices (default: en)")
    parser.add_argument("--chapter-mode", action="store_true",
                        help="Attempt to split by chapters (experimental)")

    args = parser.parse_args()

    # List voices mode
    if args.list_voices:
        await list_voices(args.language)
        return

    # Validate input
    if not args.pdf:
        parser.print_help()
        print("\n❌ Error: Please provide a PDF file path.")
        sys.exit(1)

    if not os.path.exists(args.pdf):
        print(f"❌ File not found: {args.pdf}")
        sys.exit(1)

    # Set up output paths
    pdf_name = Path(args.pdf).stem
    output_dir = f"{pdf_name}_audio_chunks"
    output_file = args.output or f"{pdf_name}_audiobook.mp3"

    print(f"""
╔══════════════════════════════════════════════════════════╗
║           📚 PDF to Audiobook Converter                  ║
╚══════════════════════════════════════════════════════════╝

  📄 Input:  {args.pdf}
  🎙️  Voice:  {args.voice}
  ⚡ Rate:   {args.rate}
  📁 Output: {output_file}
""")

    # Step 1: Extract text
    pages = extract_text_from_pdf(args.pdf, args.start_page, args.end_page)

    if not pages:
        print("❌ No text found in the PDF. It might be a scanned/image-based PDF
        print("   For scanned PDFs, you'll need OCR (pytesseract). Let me know if
        sys.exit(1)

    print(f"   Extracted text from {len(pages)} pages.\n")
```

```
    # Step 2: Generate audio
    audio_files = await generate_audiobook(pages, output_dir, args.voice, args.ra

    if not audio_files:
        print("❌ No audio files were generated.")
        sys.exit(1)

    # Step 3: Merge into single file
    if not args.no_merge:
        success = merge_audio_files(audio_files, output_file)

        if success and not args.keep_chunks:
            cleanup_chunks(output_dir, audio_files)
            # Remove the chunks directory if empty
            try:
                os.rmdir(output_dir)
            except OSError:
                pass

    print(f"""
╔═══════════════════════════════════════════════════════╗
║                    🎉 All Done!                       ║
╠═══════════════════════════════════════════════════════╣
║  Your audiobook is ready to listen!                   ║
║  Transfer the MP3 to your phone or any audio player.  ║
╚═══════════════════════════════════════════════════════╝
""")


if __name__ == "__main__":
    asyncio.run(main())
```