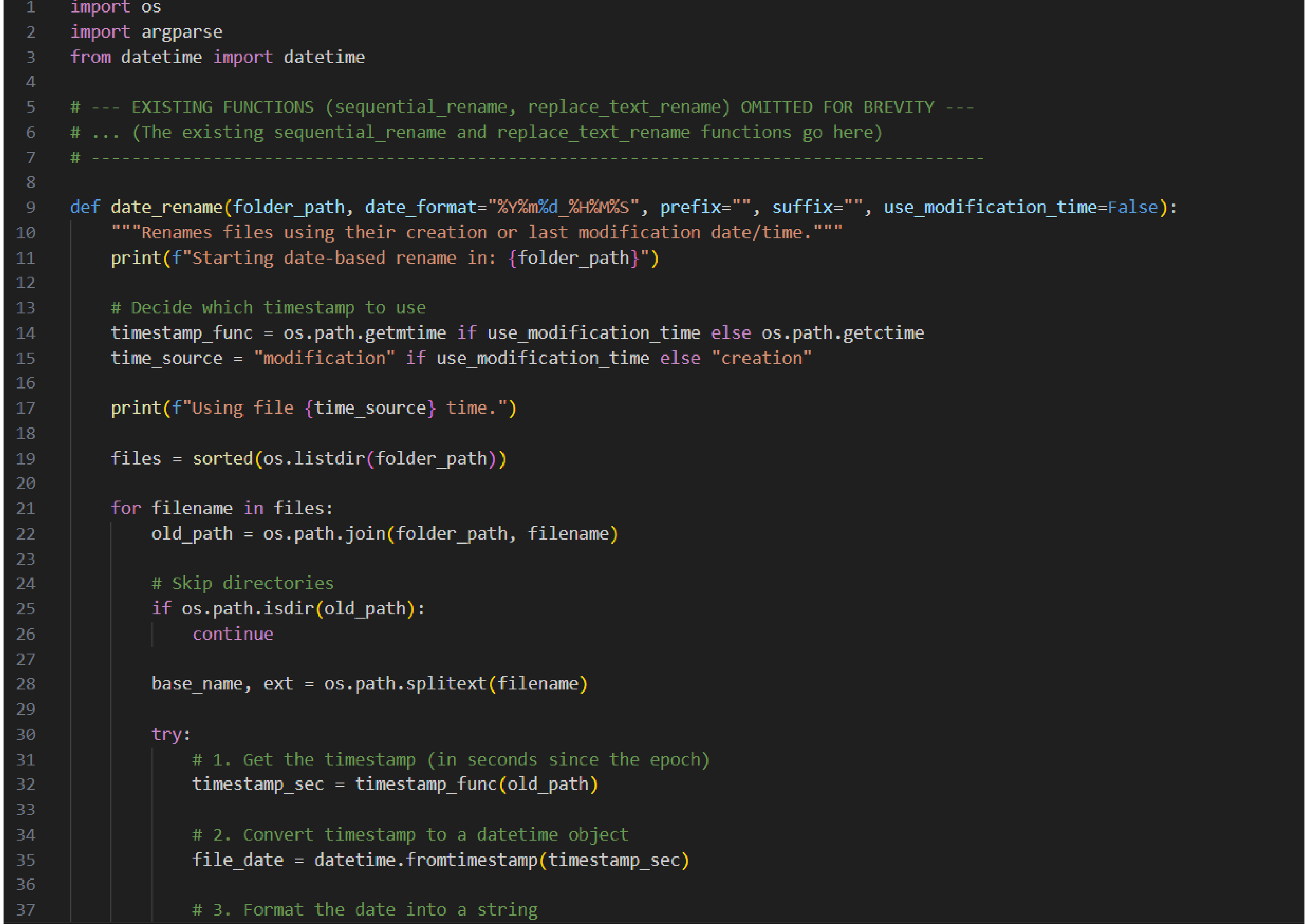
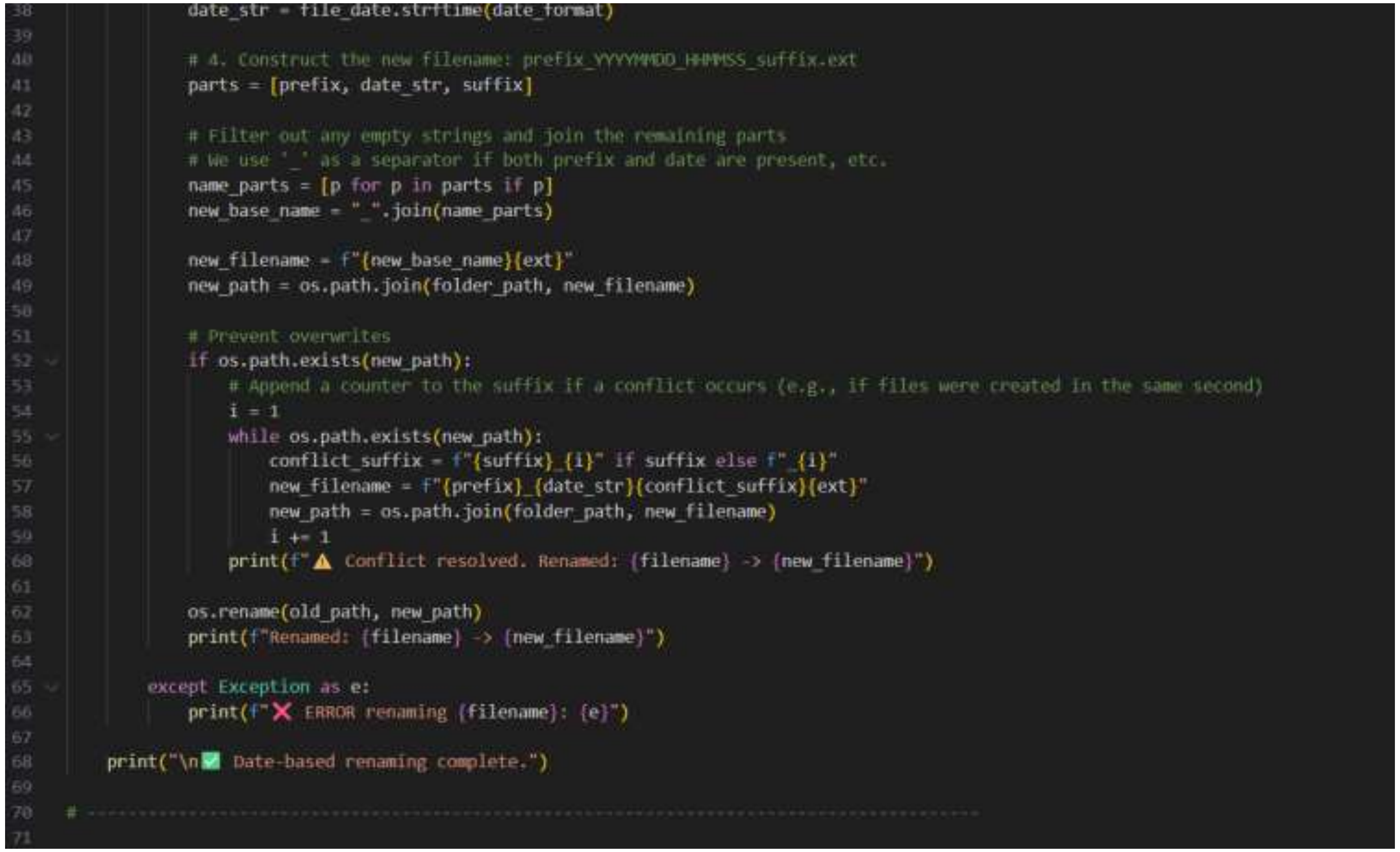
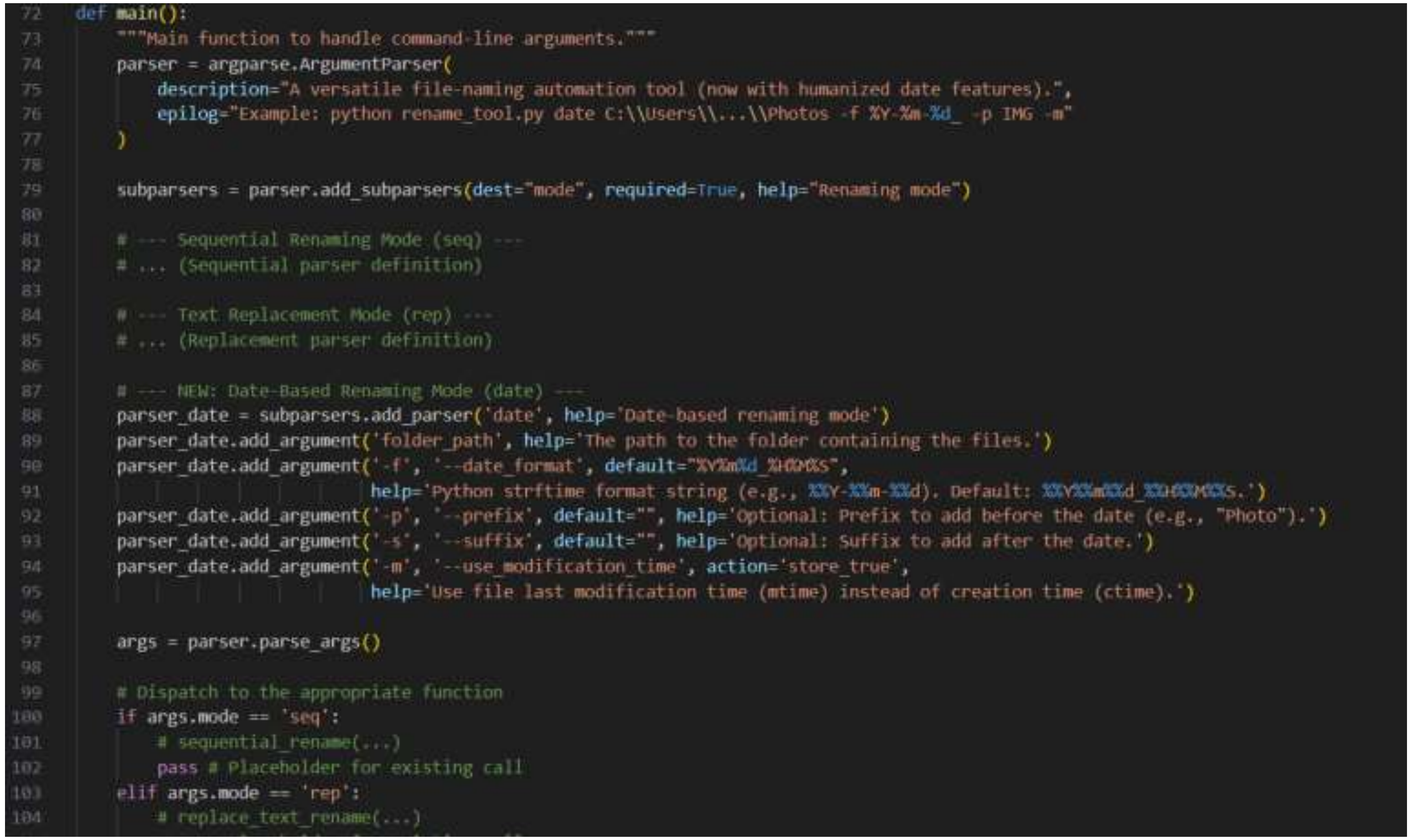


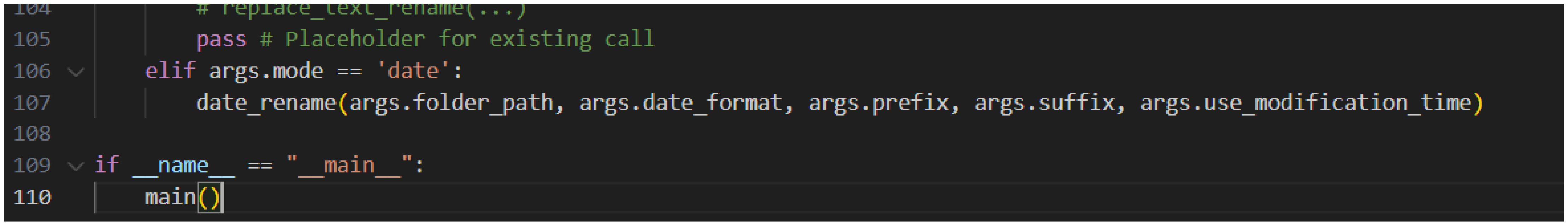
```python
import os
import argparse
from datetime import datetime

# --- EXISTING FUNCTIONS (sequential_rename, replace_text_rename) OMITTED FOR BREVITY ---
# ... (The existing sequential_rename and replace_text_rename functions go here)
# -------------------------------------------------------------------------------------

def date_rename(folder_path, date_format="%Y%m%d_%H%M%S", prefix="", suffix="", use_modification_time=False):
    """Renames files using their creation or last modification date/time."""
    print(f"Starting date-based rename in: {folder_path}")

    # Decide which timestamp to use
    timestamp_func = os.path.getmtime if use_modification_time else os.path.getctime
    time_source = "modification" if use_modification_time else "creation"

    print(f"Using file {time_source} time.")

    files = sorted(os.listdir(folder_path))

    for filename in files:
        old_path = os.path.join(folder_path, filename)

        # Skip directories
        if os.path.isdir(old_path):
            continue

        base_name, ext = os.path.splitext(filename)

        try:
            # 1. Get the timestamp (in seconds since the epoch)
            timestamp_sec = timestamp_func(old_path)

            # 2. Convert timestamp to a datetime object
            file_date = datetime.fromtimestamp(timestamp_sec)

            # 3. Format the date into a string
```



```python
            date_str = file_date.strftime(date_format)

            # 4. Construct the new filename: prefix_YYYYMMDD_HHMMSS_suffix.ext
            parts = [prefix, date_str, suffix]

            # Filter out any empty strings and join the remaining parts
            # We use '_' as a separator if both prefix and date are present, etc.
            name_parts = [p for p in parts if p]
            new_base_name = "_".join(name_parts)

            new_filename = f"{new_base_name}{ext}"
            new_path = os.path.join(folder_path, new_filename)

            # Prevent overwrites
            if os.path.exists(new_path):
                # Append a counter to the suffix if a conflict occurs (e.g., if files were created in the same second)
                i = 1
                while os.path.exists(new_path):
                    conflict_suffix = f"{suffix}_{i}" if suffix else f"_{i}"
                    new_filename = f"{prefix}_{date_str}{conflict_suffix}{ext}"
                    new_path = os.path.join(folder_path, new_filename)
                    i += 1
                print(f"⚠ Conflict resolved. Renamed: {filename} -> {new_filename}")

            os.rename(old_path, new_path)
            print(f"Renamed: {filename} -> {new_filename}")

        except Exception as e:
            print(f"❌ ERROR renaming {filename}: {e}")

    print("\n✅ Date-based renaming complete.")

# ------------------------------------------------------------------------------------------
```



```python
def main():
    """Main function to handle command-line arguments."""
    parser = argparse.ArgumentParser(
        description="A versatile file-naming automation tool (now with humanized date features).",
        epilog="Example: python rename_tool.py date C:\\Users\\...\\Photos -f %Y-%m-%d_ -p IMG -m"
    )

    subparsers = parser.add_subparsers(dest="mode", required=True, help="Renaming mode")

    # --- Sequential Renaming Mode (seq) ---
    # ... (Sequential parser definition)

    # --- Text Replacement Mode (rep) ---
    # ... (Replacement parser definition)

    # --- NEW: Date-Based Renaming Mode (date) ---
    parser_date = subparsers.add_parser('date', help='Date-based renaming mode')
    parser_date.add_argument('folder_path', help='The path to the folder containing the files.')
    parser_date.add_argument('-f', '--date_format', default="%Y%m%d_%H%M%S",
                             help='Python strftime format string (e.g., %%Y-%%m-%%d). Default: %%Y%%m%%d_%%H%%M%%S.')
    parser_date.add_argument('-p', '--prefix', default="", help='Optional: Prefix to add before the date (e.g., "Photo").')
    parser_date.add_argument('-s', '--suffix', default="", help='Optional: Suffix to add after the date.')
    parser_date.add_argument('-m', '--use_modification_time', action='store_true',
                             help='Use file last modification time (mtime) instead of creation time (ctime).')

    args = parser.parse_args()

    # Dispatch to the appropriate function
    if args.mode == 'seq':
        # sequential_rename(...)
        pass # Placeholder for existing call
    elif args.mode == 'rep':
        # replace_text_rename(...)
```



```
        # replace_text_rename(...)
        pass # Placeholder for existing call
    elif args.mode == 'date':
        date_rename(args.folder_path, args.date_format, args.prefix, args.suffix, args.use_modification_time)

if __name__ == "__main__":
    main()
```