Py_Console Scripts – Technical Documentation

Comprehensive analysis and usage guide for the primary console utilities, including functional overview, code structure, dependencies, and sample scenarios.

Table of Contents

1. WebP to JPG Converter

File: c:\Codes\Py_Console\convert_webp_to_jpg.py

Core Functionality

Converts `.webp`, `.png`, and `.jpeg` images to `.jpg` format. Processes files recursively in a specified folder.

Key Features

- Batch processing with a progress bar (`tqdm`).
- Safety guards for decompression bombs via Pillow settings.
- Strict validation: ensure file exists, non-zero size, basic verify() pass.
- Deletes the source file only after successful conversion.

Note: The current implementation does not support quality adjustment or output directory selection via CLI. Conversion uses Pillow defaults and saves beside the source.

Inputs and Outputs

- Input: Folder path provided via CLI arg or interactive prompt.
- Output: JPG files saved in place; original sources deleted on success.
- Supported input extensions: `.webp`, `.png`, `.jpeg` (skips `.gif`, `.jpg`).

Example Usage

```
# Convert all convertible images under 'D:\Pictures\WebP' (Windows)
python c:\Codes\Py_Console\convert_webp_to_jpg.py "D:\Pictures\WebP"
```

Sample Runtime Output

```
Converting: 120img | Converted=115 | Deleted=115
[SUMMARY]
 Total files considered: 120
 Converted: 115
 Deleted originals: 115
 Failed: 5
```

Code Structure Overview

- Functions: list_files_recursive, should_convert, target_jpg_path, convert_one, delete_source, convert_images
- No classes used; single-file utility.

Dependencies

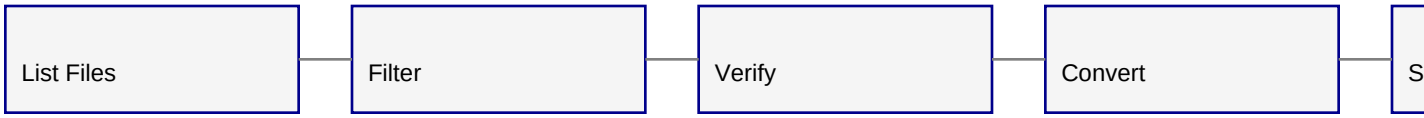
- Pillow (`PIL`): image loading and conversion.
- tqdm: progress bars.

Recommended Enhancements

- Add `--quality` CLI option to control JPEG quality (e.g., 90).
- Add `--output` CLI option to specify output directory.

- Add `--keep-source` flag to retain original files.

Conversion Pipeline



2. Enhanced Video Downloader

File: c:\Codes\Py_Console\enhanced_video_downloader.py

Core Functionality

Extracts and downloads videos from web pages without login. Supports GUI and CLI modes, multiple extraction strategies, and robust download flows.

Key Features

- Multiple format support: direct file links, social media embeds, streaming (HLS/DASH).
- Resolution/quality selection via `-q/--quality` (forwarded to `yt-dlp`).
- Progress tracking with `tqdm` for direct downloads and GUI status updates.
- Android-aware behavior (timeouts, chunk sizes, storage checks).
- Robust session with retries, headers, and fetch strategies (referrer/mobile/proxy).
- Termination via Escape key; graceful stopping.

Inputs and Outputs

- Input: Single URL or batch URLs (GUI/CLI).
- Output: Video files saved to output directory (`-o/--output`, default `downloads`).
- Quality: `-q best|worst|bestvideo|worstvideo` passed to `yt-dlp`.

Example Usage

```
# GUI mode
python c:\Codes\Py_Console\enhanced_video_downloader.py

# CLI single URL, 1080p (best MP4 if available)
python c:\Codes\Py_Console\enhanced_video_downloader.py "https://www.youtube.com/watch?v=XXXX" -q best -o "D
```

Code Structure Overview

- Class: EnhancedVideoDownloader
- Methods: __init__, _setup_enhanced_session, setup_keyboard_monitor, setup_logging, create_output_directory, fetch_page_enhanced, _fetch_direct, _fetch_with_referrer, _fetch_with_mobile_agent, _fetch_with_proxy_headers, extract_videos_enhanced, _extract_direct_video_links, _extract_embedded_videos, _extract_social_media_videos, _extract_streaming_videos, _extract_with_ytdlp, download_video_enhanced, _download_direct, _download_with_ytdlp, _download_streaming, create_gui, analyze_url, toggle_input_mode, load_urls_from_file, analyze_batch_urls, clear_batch_urls, _update_video_list, download_selected, download_all, _download_videos, choose_output_folder, run_gui, run_cli
- GUI helpers: create_gui(), analyze_url(), download_selected(), download_all(), run_gui()
- CLI: run_cli(), main()

Dependencies

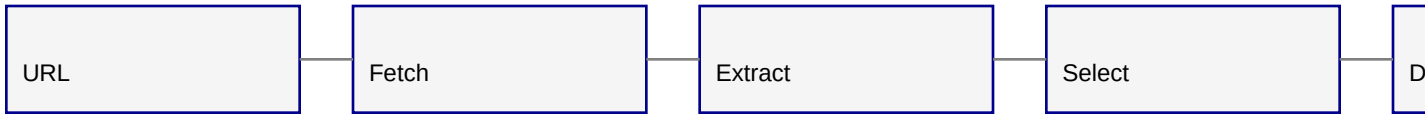
- requests, BeautifulSoup (bs4): page fetching and parsing.
- yt-dlp: resilient downloads for common platforms and playlists.

- tqdm: progress bars for direct downloads.
- tkinter: GUI (desktop environments).
- urllib3 Retry, HTTPAdapter: robust session retries.

Use Cases

- Download public videos from landing pages with embedded players.
- Batch extraction for multiple URLs with preview list and selective download.
- Fallback to `yt-dlp` for complex streaming or site-specific formats.

Download Flow



3. Secure Face Recognition Sorter

File: c:\Codes\Py_Console\secure_sort_by_known_faces.py

Core Functionality

Analyzes images and moves them into destination folders if they contain known faces. Interactive prompts when CLI args are missing.

Key Features

- Face detection and matching using `face_recognition` encodings.
- Known faces database via user-provided reference images (face->dest pairs).
- Secure processing: size/pixel caps, safe atomic moves, traversal prevention.
- Parallelized face analysis with ProcessPoolExecutor; single-threaded moves.
- Quiet output: single progress bar with concise final summary.

Inputs and Outputs

- Input: `--src` folder; optional `--recursive`.
- Parameters: `--tolerance`, `--workers`, `--jitter`; multiple `--face` and `--dest` pairs.
- Output: Images moved to first matched destination folder only.

Example Usage

```
# Interactive mode to define pairs:
python c:\Codes\Py_Console\secure_sort_by_known_faces.py

# CLI mode with pairs:
python c:\Codes\Py_Console\secure_sort_by_known_faces.py --src "D:\Photos" --recursive \
  --tolerance 0.5 --workers 8 --jitter 2 \
  --face "D:\Refs\Mom.jpg" --dest "D:\Photos\Sorted\Mom" \
  --face "D:\Refs\Dad.jpg" --dest "D:\Photos\Sorted\Dad"
```

Sample Runtime Output

```
[INFO] Processing 250 images from D:\Photos
Processing: 100%|##########| 250/250 [moved=180 Mom:120 | Dad:60]
[SUMMARY]
 D:\Photos\Sorted\Mom: 120 moved
 D:\Photos\Sorted\Dad: 60 moved
 Total moved: 180
 Unmatched/no-face: 70
 Errors: 0
```

Code Structure Overview

- Functions: is_image_path, sanitize_filename, unique_dest, within, safe_move, validate_image, load_ref_encoding, collect_images, _analyze_image_worker, parse_args, prompt_bool, prompt_float, prompt_int, gather_pairs_interactively, interactive_fill, main
- Encodings: load_ref_encoding() builds reference encodings for known faces.
- Worker: _analyze_image_worker() encodes and compares faces in parallel.
- Filesystem safety: unique_dest(), within(), safe_move() with atomic replace.

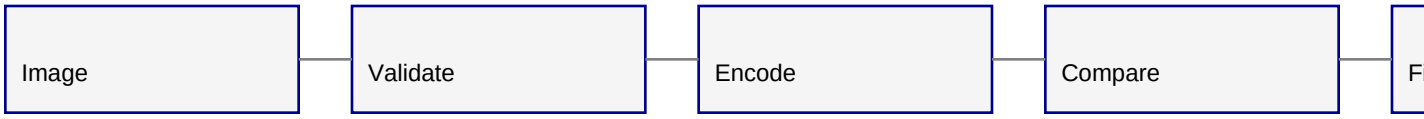
Dependencies

- face_recognition: face encodings and comparisons.
- Pillow: image validation and safety limits.
- tqdm: progress visualization.

Use Cases

- Organizing photo library by detected family members.
- Sorting event photos into folders per attendee.
- Filtering out unmatched images to a separate holding folder (user-defined).

Sorting Pipeline



4. Video to GIF Converter

File: c:\Codes\Py_Console\video_to_gif.py

Core Functionality

Converts video files to animated GIF or WebP with advanced optimization, time range selection, and optional batch mode.

Key Features

- Frame rate control via `--fps` and adaptive FPS calculation.
- Duration selection via `--start` and `--end` (seconds).
- Size optimization: multi-strategy GIF optimizer (quality/frame/palette).
- Resolution control via `--width` (maintains aspect ratio, capped by config).
- Batch conversion with concurrency (`--batch`, `--workers`).
- Output format selection: `--format gif|webp`.
- Optional source deletion with integrity verification (`--delete-source`).

Inputs and Outputs

- Input: Single video file or directory for batch mode.
- Output: GIF or WebP file(s) to specified output path/dir (`-o/--output`).
- Supported video formats: .mp4, .avi, .mov, .mkv, .wmv, .flv, .webm, .m4v, .3gp, .ogv, .ts, .mts

Example Usage

```
# Create a 5-second GIF at 15fps
python c:\Codes\Py_Console\video_to_gif.py "D:\Clips\sample.mp4" --format gif -o "D:\Clips\sample.gif" --fps

# Batch convert a folder to WebP with 12fps
python c:\Codes\Py_Console\video_to_gif.py "D:\Clips" --batch --format webp --fps 12 --workers 4 -o "D:\Clip
```

Sample Runtime Output

```
Convert: sample.mp4 -> sample.gif
[INFO] Extracting frames (fps=15, width=640)
[INFO] Optimizing GIF: quality→frames→palette
[INFO] Success: output sample.gif (11.8MB)
```

Code Structure Overview

- Classes: ConversionConfig, VideoToGifConverter, ConversionResult.
- VideoToGifConverter methods: __init__, _setup_logger, is_supported_video, _calculate_file_hash, _verify_gif_integrity, _verify_webp_integrity, _safe_delete_source, get_video_info, optimize_gif_size, _optimize_by_quality, _optimize_by_frame_reduction, _optimize_by_palette_reduction, _extract_frames_efficiently, calculate_optimal_fps, convert_video_to_gif, convert_video_to_webp, _perform_conversion, batch_convert, _batch_convert_sequential, _batch_convert_concurrent
- Batch conversion: batch_convert() with concurrent/sequential options.
- Optimization internals: _optimize_by_quality(), _optimize_by_frame_reduction(), _optimize_by_palette_reduction().

Dependencies

- moviepy: Video decoding and clip handling.
- Pillow: GIF/WebP frame processing and saving.
- numpy: frame manipulation.
- imageio: format IO helpers.
- tqdm: progress (when available).

Use Cases

- Creating short animations or meme GIFs from video clips.
- Generating lightweight previews (WebP) for web or mobile use.
- Batch processing event recordings into optimized shareable formats.

Conversion Flow