



MIRA Converter

HEIC → JPG 一括変換アプリ

アルゴリズム解説

初めて作った自作アプリを「何が起きているか」から理解するための技術メモ

結論から言うと、このアプリはAIや難しい数式で動いているわけではありません。
画像を決められた順番で処理する「変換パイプライン」です。入力されたHEIC/HEIFを読み、表示方向を直し、必要なら縮小し、JPEG向けの色形式へ変え、圧縮してZIPにまとめます。

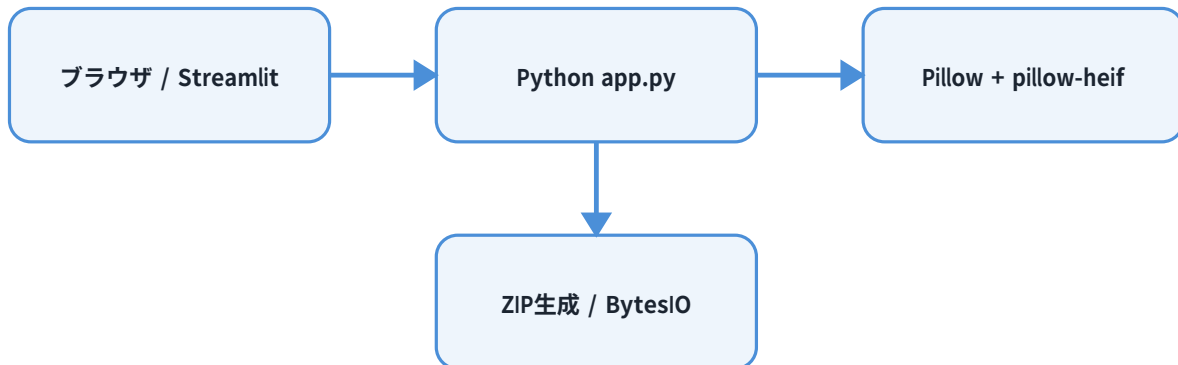


入力を「向き → サイズ → 色空間 → JPEG → ZIP」の順に処理する直列パイプライン

Version note: 2026-10-03時点の会話内で作成した app.py / run.ps1 構成を基準に解説。

1. このアプリの正体

MIRA Converterは、Webブラウザを操作画面として使うローカルPythonアプリです。ブラウザで動いているように見えますが、画像変換そのものはPC上のPythonが実行しています。



Streamlitは画面係、Pillow/pillow-heifは画像処理係。app.pyが全体を指揮する。

役割分担

部品	役割
Streamlit	Uploadボタン、品質スライダー、ダウンロードボタンなどの画面を作る
app.py	「何を、どの順番で処理するか」を記述したアプリ本体
Pillow	画像の回転補正、リサイズ、RGB変換、JPEG保存を担当
pillow-heif	PillowがHEIC/HEIFを開けるように橋渡しする
BytesIO / zipfile	変換後のJPGをメモリ上に作り、ZIPへまとめる
run.ps1	PowerShellからStreamlitアプリを起動する入口

ポイント：ブラウザは「操作盤」。エンジンはPythonです。だからブラウザを閉じてもコードは消えず、run.ps1を再実行すれば同じアプリをまた起動できます。

2. 画像変換アルゴリズムを順番に読む

STEP 0 | HEIC対応を有効化

`register_heif_opener()` を最初呼び、Pillowの `Image.open()` がHEIC/HEIFを扱えるようにします。

STEP 1 | ユーザーが複数ファイルを選択

`st.file_uploader(..., accept_multiple_files=True)` により、複数のHEIC/HEIFを一度に受け取ります。以後は `for` ループで1枚ずつ処理します。

STEP 2 | 画像を開く

`Image.open(uploaded_file)` で、アップロードされた画像をPillowのImageオブジェクトに変換します。

STEP 3 | Exifの向きを反映

`ImageOps.exif_transpose(image)`

を使います。iPhone写真などは見た目の向きをExif情報で保持する場合があるため、正しい向きへ直します。

STEP 4 | 必要ならリサイズ

`max_width` が指定され、元画像の横幅がそれより大きい場合だけ縮小します。縦横比を保つよう高さも同じ倍率で計算します。

STEP 5 | JPEG向けに色形式を調整

JPEG向けに、RGBAなど別モードなら `image.convert("RGB")` で変換します。

STEP 6 | JPEGとして圧縮保存

`image.save(..., format="JPEG", quality=quality, optimize=True)`

でJPEGへ再エンコードします。`quality`は画質とファイルサイズのトレードオフです。

STEP 7 | ZIPへまとめる

各JPGをBytesIO上で作成し、そのバイト列をzipfileへ追加します。最後に1つのZIPとしてダウンロードさせます。

3. リサイズ計算の中身

このアプリで数式らしい部分は、主に縦横比を守るリサイズです。

```
ratio = max_width / image.width
new_height = int(image.height * ratio)
new_size = (max_width, new_height)
```

例：元画像が 4032 × 3024 px、最大横幅を 1920 px にした場合

計算	結果
縮小率	$1920 \div 4032 = 0.47619$
新しい高さ	$3024 \times 0.47619 = 1440 \text{ px}$
出力サイズ	$1920 \times 1440 \text{ px}$

横だけ1920にして縦を固定するのではなく、比率を同じ倍率で縮めるので写真が歪みません。

JPEG品質 92 の意味

quality=92 はJPEG圧縮時の品質パラメータです。数値を上げるほど一般に画質は高くなり、ファイルサイズも大きくなります。92は高品質寄りの実用的な初期値です。

optimize=True はJPEGの符号化を少し最適化してファイルサイズ削減を狙います。

4. なぜZIPなのか

複数のJPGを1枚ずつダウンロードさせると何度も保存操作が必要です。そこで変換したJPGを1つのZIPへまとめます。

```
zip_buffer = io.BytesIO()
with zipfile.ZipFile(zip_buffer, "w", zipfile.ZIP_DEFLATED) as zf:
    zf.writestr(jpg_name, jpg_buffer.getvalue())
```

BytesIOは「メモリの中にある仮想ファイル」のようなものです。途中生成物を必ずディスクへ書かずに処理できます。

5. run.ps1 は何をしている？

現在の軽量版 run.ps1 は、ほぼ次の2行です。

```
Set-Location "C:\ProjectG\NOTE\heic-to-jpg-web"
python -m streamlit run app.py
```

デスクトップのMIRA Converterショートカットは、このrun.ps1を呼ぶ入口です。

だから毎回 pip install は不要

必要なライブラリは初回に一度インストールすれば、そのPCのPython環境に残ります。普段は起動だけでよく、別PCへ移したときや環境を作り直したときだけ requirements.txt を使います。

6. GitHubに保存した意味

GitHubに入れたことで、MIRA

Converterは「このPCに偶然存在するファイル」から「履歴を持つソフトウェア」になりました。

操作	意味
git init	このフォルダをGitで管理開始
git add .	現在のファイルを次の保存対象へ登録
git commit	その時点の状態へ名前を付けて履歴保存
gh repo create	GitHub上にリポジトリを新規作成
git push	ローカルの履歴をGitHubへ送信

GitHubにあるもの：app.py、requirements.txt、run.ps1、push.ps1、README.md、.gitignore、mira-converter.ico。これらが揃っているので、将来別PCで復活させる材料が残っています。

7. これは「アルゴリズム」なのか？

はい。ただし複雑な推論ではなく、手順型アルゴリズムです。入力に対し、決められた工程を順番に適用し、出力を作ります。

入力 HEIC/HEIF → 読込 → 向き補正 → 条件分岐（縮小するか） → 色変換 → JPEG圧縮 → ZIP集約 → 出力

アルゴリズムを理解するときは、「何を入力するか」「途中で何を判断するか」「どんな変換をするか」「最後に何を返すか」の4点を見ると、コード全文を読めなくても骨格が見えます。

8. 次に育てるなら

機能	追加する考え方
PNG / WEBP → JPG	アップロード可能な拡張子を増やし、同じJPEG保存処理へ流す
WordPress用1920px	最大横幅のプリセットを「WordPress」として固定化
note用1280px	1280pxプリセットと品質をセットで用意
ファイル名一括変更	JPG名を連番や日付、接頭辞で生成する関数を追加
画像プレビュー	変換前後のサムネイルをStreamlit上に表示
EXIF保持	必要なメタデータだけ引き継ぐ設計を追加
エラーログ	壊れたHEICなどをスキップして一覧表示する

いちばん大事な理解

今回やったことは「Pythonを全部理解してからアプリを作った」ではありません。先に目的を決め、動く最小構成を作り、PowerShellで起動方法を固定し、GitHubへ履歴を残しました。コードの理解は、動くものを分解しながら後から深められます。

MIRA Converter / necokoucan