Converting Python Script to Mac Executable

Overview

This guide explains how to convert a Python script that processes files between input and output folders into a standalone executable for macOS.

Installation

First, install PyInstaller, the tool we'll use to create the executable:

```
pip install pyinstaller
```

Basic Conversion

Assuming your script is named `process_files.py`, create a basic executable with:

```
pyinstaller --onefile process_files.py
```

This creates a single executable file in the `dist` directory.

Improved Script with Command-Line Arguments

Modify your script to better handle input and output folders:

```python
import os
import sys
import argparse

def process_files(input_folder, output_folder):
    # Your file processing code here
    # Read files from input_folder
    # Write files to output_folder
    pass

def main():
    parser = argparse.ArgumentParser(description='Process files
        from input folder to output folder')
    parser.add_argument('input', help='Input folder path')
    parser.add_argument('output', help='Output folder path')
    # Check if running as a script or as a frozen executable
    if getattr(sys, 'frozen', False):
        # If frozen, use the bundled arguments or show GUI
        if len(sys.argv) > 2:
            args = parser.parse_args()
        else:
            # Simple fallback for drag-and-drop or double-click
            print("Please provide input and output folders:")
            input_folder = input("Input folder: ")
            output_folder = input("Output folder: ")
            if not os.path.isdir(input_folder) or not
        os.path.isdir(output_folder):
                print("Invalid folder paths")
                return
            process_files(input_folder, output_folder)
            return
    else:
        # If running as script, parse arguments normally
        args = parser.parse_args()

    # Process the files
    process_files(args.input, args.output)

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

Creating an App Bundle

For a better macOS experience, create an app bundle:

```
pyinstaller --windowed --onefile process_files.py
```

Maximum Compatibility

To ensure compatibility across different macOS versions:

```
pyinstaller --onefile --target-architecture universal2
        process_files.py
```

Adding a Simple GUI (Optional)

For a more user-friendly experience, add a GUI using tkinter:

```
import os
import sys
import tkinter as tk
from tkinter import filedialog, messagebox
```

```python
def process_files(input_folder, output_folder):
    # Your file processing code here
    # Read files from input_folder
    # Write files to output_folder
    pass

def select_folder(entry):
    folder_path = filedialog.askdirectory()
    if folder_path:
        entry.delete(0, tk.END)
        entry.insert(0, folder_path)

def run_process(input_entry, output_entry, status_label):
    input_folder = input_entry.get()
    output_folder = output_entry.get()

    if not input_folder or not output_folder:
        messagebox.showerror("Error", "Both input and output
        folders must be specified")
        return

    if not os.path.isdir(input_folder) or not
        os.path.isdir(output_folder):
        messagebox.showerror("Error", "Invalid folder paths")
        return

    status_label.config(text="Processing...")
    try:
        process_files(input_folder, output_folder)
        status_label.config(text="Completed successfully!")
    except Exception as e:
        status_label.config(text=f"Error: {str(e)}")
        messagebox.showerror("Error", str(e))

def main_gui():
    root = tk.Tk()
    root.title("File Processor")
    root.geometry("600x200")

    # Input folder
    tk.Label(root, text="Input Folder:").grid(row=0, column=0,
        padx=10, pady=10, sticky="w")
    input_entry = tk.Entry(root, width=50)
    input_entry.grid(row=0, column=1, padx=10, pady=10)
    tk.Button(root, text="Browse...", command=lambda:
        select_folder(input_entry)).grid(row=0, column=2, padx=10,
        pady=10)

    # Output folder
    tk.Label(root, text="Output Folder:").grid(row=1, column=0,
        padx=10, pady=10, sticky="w")
    output_entry = tk.Entry(root, width=50)
    output_entry.grid(row=1, column=1, padx=10, pady=10)
    tk.Button(root, text="Browse...", command=lambda:
        select_folder(output_entry)).grid(row=1, column=2, padx=10,
        pady=10)

    # Status label
    status_label = tk.Label(root, text="Ready")
    status_label.grid(row=2, column=0, columnspan=3, padx=10,
        pady=10)

    # Process button
    tk.Button(root, text="Process Files", command=lambda:
        run_process(input_entry, output_entry,
        status_label)).grid(row=3, column=1, padx=10, pady=10)

    root.mainloop()

def main():
    # Use GUI if no arguments provided or running as executable
    if len(sys.argv) <= 1 or getattr(sys, 'frozen', False):
        main_gui()
    else:
        # Command line mode
        import argparse
        parser = argparse.ArgumentParser(description='Process
        files from input folder to output folder')
        parser.add_argument('input', help='Input folder path')
        parser.add_argument('output', help='Output folder path')
        args = parser.parse_args()

        if not os.path.isdir(args.input) or not
        os.path.isdir(args.output):
            print("Invalid folder paths")
            return

        process_files(args.input, args.output)

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

To build this version with the GUI:

```
pip install pyinstaller
pyinstaller --windowed --onefile --name FileProcessor
        process_files.py
```

Distribution

After building, your executable will be in the `dist` directory. You can distribute this file to other Mac users, who can run it by double-clicking in Finder or from the terminal.

Troubleshooting

If your executable doesn’t work on other Macs, check: 1. macOS version compatibility 2. Architecture compatibility (Intel vs. Apple Silicon) 3. Missing dependencies (ensure all imports are captured) 4. Code signing issues (may need to sign your app for distribution)

For advanced distribution, consider code signing your application with your Apple Developer ID.