Python Packaging Made Simple

Creating Wheels & Standalone Executables

A Practical Guide to Distribution

Our Sample Python Code

We'll start with a very simple Python function.

File: hello.py

```
def say_hello(): """Prints a greeting message."""
print("Hello, World, from the Python package!")
```

What you'll learn: How to package this code in two different ways—as a reusable library and as a standalone executable.

Example 1: Creating a Wheel (.whl) File ⚙️

Goal: Package our say_hello function as a library that other developers can install and use in their own projects.

A Wheel file is the Python equivalent of a Java .jar library file —it allows developers to install your code using pip and import it into their projects.

Key benefits:

- Easy distribution to other developers
- Simple installation with pip
- Reusable across multiple projects

Step 1: Organize Your Project

For Python to understand how to package your code, you need a specific folder structure and a configuration file.

Create this folder structure:

```
helloworld-library/ ├── src/ │ └── mylib/ │ ├──
__init__.py # Package marker │ └── hello.py # Your code
here └── pyproject.toml # Configuration file
```

The __init__.py file tells Python that this directory is a package.

Step 2: Create the Configuration File

The pyproject.toml file tells Python's build tools everything they need to know about your project.

File: pyproject.toml

```
[build-system]

requires = ["setuptools"]

build-backend = "setuptools.build_meta"

[project]

name = "helloworld-lib"

version = "1.0.0"

description = "A simple Hello World library."
```

This configuration defines the project name, version, and build requirements.

Step 3 & 4: Build the Wheel

Open your terminal and navigate to the helloworld-library folder.

Install the build tool:

```
pip install build
```

Build your package:

```
python -m build
```

Output: You'll find your .whl file in the dist/ folder!

```
helloworld_lib-1.0.0-py3-none-any.whl
```

Example 2: Creating a Standalone Executable 🖥️

Goal: Turn a script into a single .exe file (on Windows) that anyone can run, even without Python installed.

An executable is the Python equivalent of a Java runnable .jar file—users can double-click and run it immediately.

Key benefits:

- No Python installation required for users
- Single file distribution
- Professional user experience

Step 1: Create Your Application Script

This time, we'll make a script that is meant to be run directly, not imported.

File: app.py

```
# A simple script that prints a message and waits for the
user print("Hello, World! This is a standalone
application.") input("Press Enter to close this window...")
```

Note: The input() function keeps the window open so users can see the output before closing.

Step 2 & 3: Use PyInstaller

PyInstaller is the most popular tool for converting Python scripts to executables.

Install PyInstaller:

```
pip install pyinstaller
```

Build your executable:

```
pyinstaller --onefile app.py
```

Your output:

```
dist/ └── app.exe (or 'app' on Linux/macOS)
```

The --onefile flag bundles everything into a single executable file.

Summary: Two Paths for Python Distribution

Your Goal	The Java Way	The Python Way	Final Output
Create a library for developers	Package as .jar library	Use build tool with pyproject.toml	.whl (Wheel) file
Create a runnable app for users	Package as runnable .jar	Use PyInstaller	.exe or executable

Choose the path that matches your distribution goals!