Learning Objectives: Wheels

Learners will be able to…

- **Define and explain the benefits of a Python wheel**
- **Differentiate between a wheel and source distribution**
- **Build a pure-Python wheel**
- **Identify places where packages can be hosted**

info

Make Sure You Know

You are comfortable with basic terminal commands.

Limitations

This content only covers building a simple, pure-Python wheel. This content does not go into the specifics of building wheels for different platforms, testing, etc. You will not actually post your wheel to GitHub or PyPI. Instead, a general overview is given to these processes.

What are Wheels?

Installing Wheels

Before introducing wheels, let’s create a virtual environment.

```
mkdir wheels
cd wheels
python3 -m venv wheels-env
source wheels-env/bin/activate
```

Before we begin working with wheels, we need to make sure our system has the latest tools we need. Remember, the `-U` flag means to upgrade the package.

```
python -m pip install -U pip wheel setuptools
```

Installing a package often means downloading a wheel. You probably never noticed this as most people don’t read the terminal output during the installation process. Take a look at what happens when we install the `numpy` package.

```
python3 -m pip install numpy
```

Python downloads a file that ends with the `.whl` file extension. That is a wheel. We can break this process down into two steps. One, download the wheel. Two, install the wheel. This whole process does not take much time at all.

```
Collecting numpy
  Downloading numpy-1.19.5-cp36-cp36m-manylinux2010_x86_64.whl
(14.8 MB)
     |████████████████████████████████| 14.8 MB 10.0 MB/s
Installing collected packages: numpy
Successfully installed numpy-1.19.5
```

Compare this to installing a package that downloads the source distribution instead of a wheel. Install version 2 of the `uwsgi` package. Pay particular attention to the lines of output in the terminal.

```
python -m pip install 'uwsgi==2.0.*'
```

Python downloads a `tar.gz` file. This is a zipped version of the source distribution of the package. After that has been downloaded, Python says that it is building a wheel. Once Python built the wheel, the actual installation process begins. This is a noticeably longer process as there are more steps. Plus, building a wheel takes time.

```
Collecting uwsgi==2.0.*
  Using cached uwsgi-2.0.20.tar.gz (804 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: uwsgi
  Building wheel for uwsgi (setup.py) ... done
  Created wheel for uwsgi: filename=uWSGI-2.0.20-cp36-cp36m-
linux_x86_64.whl size=520166
sha256=924eb41be86b0f8ac789f51bb4916fdd320168e458b5e2b2d739936a1
cba1c83
  Stored in directory:
/home/codio/.cache/pip/wheels/4a/60/83/bd5b22ba1a9298cc00c9f6400
101b3757d958e10b1b38348f8
Successfully built uwsgi
Installing collected packages: uwsgi
Successfully installed uwsgi-2.0.20
```

Wheels and Source Distribution

So, what is a wheel and what is source distribution? A source distribution is the necessary source files and associated metadata that `pip` needs to install a package. In contrast, a wheel is a built distribution. That is, the source distribution is built into a form that is ready to be installed on your system.

Regardless of how you install a package, the process goes from source distribution, to built distribution, to installation on your system. Using wheels means the build step is done before you download the package. That is why the installation process with wheels is faster than source distribution. In addition, wheels are smaller than the source distribution, so the initial download should be quicker as well.

▼ **Why are they called wheels?**

Python the language was named for Monty Python, a popular British comedy group. There are lots of Easter eggs from Monty Python sketches throughout the Python ecosystem. For example, Python's bundled IDE is called IDLE after Eric Idle, a member of Monty Python. The code name for PyPI was the cheese shop, named for a Monty Python sketch. So the term “wheel” comes from a wheel of cheese, which is sold at a cheese shop.

Building a Wheel

Wheel Types

There are three types of wheels:

- Universal wheel - this wheel can run on either Python 2 or Python 3
- Pure-Python wheel - this wheel only runs on Python 2 or Python 3, not both
- Platform wheel - this wheel is designed for a specific version of Python and platform (process and/or operating system)

A quick note on Python 2, this version of Python is no longer being officially supported, so any discussion of wheels will not focus on Python 2. We are also not going to get into specifics of platform-specific wheels. You can look at the Python packaging user guide for more information.

Instead, we are going to focus on building a pure-Python wheel for Python 3.

Setup

In a previous assignment, we installed the `shapes` package from GitHub. We are going to create a wheel for this package. This package already resides on the system, there is no need to get it again from GitHub.

The package resides in the `shapes-package` directory. Change into this directory first. Then create a virtual environment called `shapes-env`. Finally, activate the virtual environment.

```
cd shapes-package
python3 -m venv shapes-env
source shapes-env/bin/activate
```

Update `pip`, `wheel`, and `setuptools`. These packages should be updated just to make sure that the process goes smoothly.

```
python -m pip install -U pip wheel setuptools
```

You need a `setup.py` file. This file is used to set features and options for the desired wheel. Click on the link below to open the setup file for the `shapes` package.

Start by importing `setup` from `setuptools`. Then we are going to pass named arguments to the `setup` function. At the very least, your `setup.py` file needs values for:

- `name` - the name of your package
- `version` - the version of your package
- `packages` - the list of where you can find the Python files for the package

Here is a sample `setup.py` file that lists all of the possible options that can be set for the wheel. Our wheel is going to add a description and an author. There is a directory called `shapes` that contains the actual Python package. Put this name in a list for the `packages` option.

```
from setuptools import setup


setup(
   name='shapes',
   version='0.1',
   description='Package for calculating area and perimeter for
        circles, squares, and rectangles.',
   author='Your name here',
   packages=['shapes'],
)
```

Building the Wheel

Now that the `setup.py` file is ready, we need to change into the `shapes` directory. The image below shows the hierarchy for `shapes-package`.

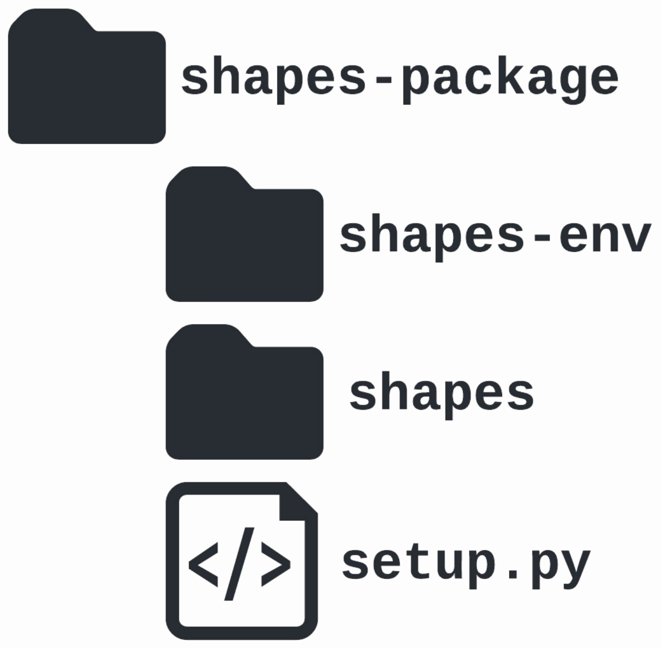

Image depicts the directory structure for this example package. The parent directory is called “shapes-package”. Located in the parent is the directory for the virtual environment, the “shapes” directory contains the Python code for the package, and the “setup” Python file.

The following command takes the source distribution (found in the next `shapes` directory) and creates a build distribution in the form of a wheel.

```
python3 setup.py sdist bdist_wheel
```

The build process created the `dist` directory. Change into this directory and then use the `ls` command, which lists the contents of the directory.

```
cd dist
ls
```

You should see the following output. The file that ends with `.whl` is the wheel. Notice that the file name for the wheel tells us lots of useful information. `shapes-0.1` is the name and version number. `py3` means the wheel works on Python 3. The `none` relates to the application binary interface (ABI). Our package does not interact with the ABI, which is why Python uses `none`. Finally, the `any` means the package can run on any platform. The `.tar.gz` file is the zipped source distribution of the package.

```
shapes-0.1-py3-none-any.whl
shapes-0.1.tar.gz
```

Hosting

Version Control Systems

We already saw how Python allows you to install packages from version control systems like GitHub. Python also supports Mercurial, Subversion, and Bazaar. We are going to talk about how a package would be pushed to GitHub. **Note**, we are not going to push the `shapes` package to GitHub.

The first thing you need to do is create a GitHub account. Then you need to create a repository for the package. If we were to push the `shapes-package` directory to GitHub, it would include the directory for the virtual environment. Virtual environments are great for local Python development. However, they are not required for the package once it has been created.

GitHub allows you to create a `.gitignore` file. This is a plain text file which tells git to not upload certain files and directories. First create the `.gitignore` file. Then add the following line of code to ignore the `shapes-env` directory which is the virtual environment:

▼ **Did you notice?**
There is a leading dot (`.`) for the `.gitignore` file. It is required.

```
/shapes-env
```

Once you are ignoring the virtual environment, you can follow the normal git workflow. Adding a license and `README` file would be helpful for people looking to use your package. See the documentation for more information on how to use the git.

PyPI

Preparing your package for use on PyPI is much more complex than the simple wheel creation from the previous page. The setup file provides the metadata, which is still required. You also need to include a license, a `.toml` file that tells Python which build tools to use. The `README` file gives users useful information on how to use the package. You are encouraged to have tests for your packages. If you do not have any tests, create the directory but leave it empty.

```
packaging_tutorial/
├── LICENSE
├── pyproject.toml
├── README.md
├── setup.cfg
├── src/
│   └── example_package/
│       ├── __init__.py
│       └── example.py
└── tests/
```

The process does not end here. You need to upload your package to TestPyPI. TestPyPI requires an account and a token. Once your package has passed all of your testing, it is ready for PyPI itself.

Formative Assessment 1

Formative Assessment 2