Packaging and Distributing Projects

UBCO Master of Data Science – DATA 533

Today's Class

Python Package Index (PyPI)

Create a python package

Upload the package to PyPi

Install the uploaded package using pip

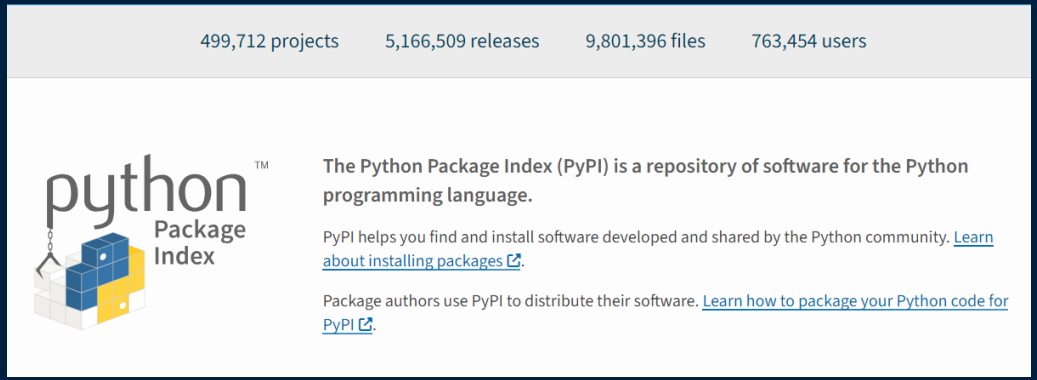
Modify the package, re-upload and re-use it

Python Package Index (PyPI)

The Python Package Index (PyPI) is a repository of software for the Python programming language.

PyPI helps you find and install software developed and shared by the Python community.

PIP is a package management system used to install and manage software packages/libraries written in Python



Why Do We Upload Packages?

Sharing and Re-use code

- Assembling code in packages makes it really easy to re-use old code
- We will be be able to access packages/modules a single `import` command

Structuring Your Project

How to best leverage Python's features to create clean, effective code

- making clean code whose logic and dependencies are clear
- how the files and folders are organized in the filesystem.

Which functions should go into which modules?

Directory Structure

```
myCalculatorPackage
|
|__ src
|     |__ myCalculatorPackage
|           |__ division
|                 |__ __init__.py
|                 |__ divide.py
|           |__ multiplication
|                 |__ __init__.py
|                 |__ multiply.py
|__ tests
|__ LICENSE
|__ pyproject.toml
|__ README.md
|__ setup.cfg
```

Project name, will not be used later

Package name, will be used later

Package and Module

```
|__ myCalculatorPackage
          |__ division
              |__ __init__.py
              |__ divide.py
          |__ multiplication
              |__ __init__.py
              |__ multiply.py
```

Your modules/packages are the core focus of the repository

If your module consists of only a single file, you can place it directly in the root of your repository

Package and Module

```
#divide.py
def divide_by_three(num):
    return num / 3

#multiply.py
def multiply_by_three(num):
     return num * 3
```

Directory Structure

```
project
.
|__ tests
|      |__ test_divide.py
```

Small test code often exists in a single file, under the project directory

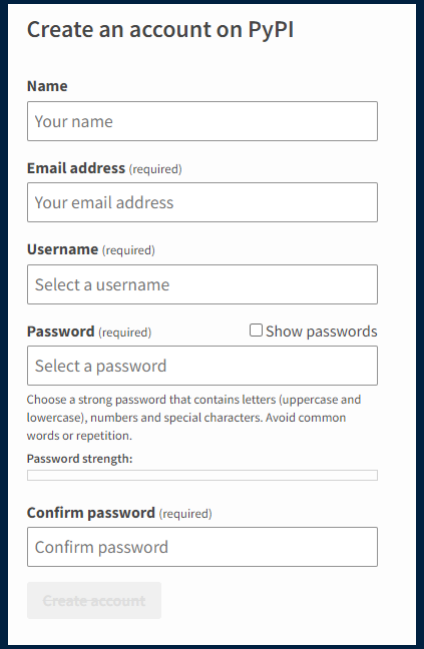
Once a test code grows, you can move your tests to a directory:

```
tests/test_divide.py
tests/test_multiply.py
```

Create a PyPi Account

A PyPi account is needed to publish your work

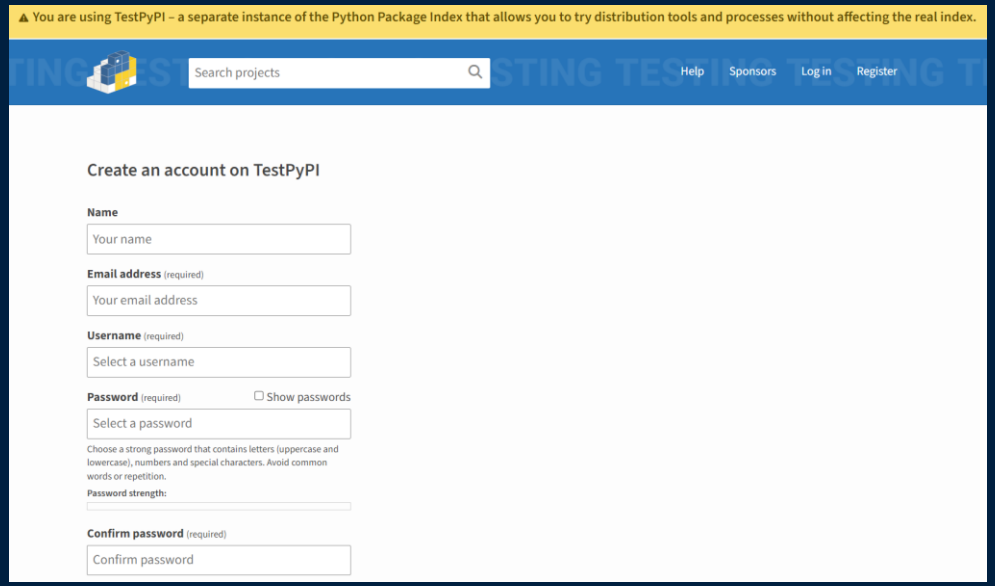
You can register: https://pypi.org/account/register/



TestPyPI Account

TestPyPI is a separate instance of the package index intended for testing and experimentation.

To register an account, go to https://test.pypi.org/account/register/



Directory Structure

Create a directory structure

```
myCalculatorPackage
|
|__ src
|      |__ myCalculatorPackage
|             |__ division
|                    |__ __init__.py
|                    |__ divide.py
|             |__ multiplication
|                    |__ __init__.py
|                    |__ multiply.py
|__ tests
```

Choosing a build backend

Tools like pip and build do not actually convert sources into a distribution package

- that job is performed by a build backend.

The build backend determines how your project will specify its configuration including

- metadata (information about the project, for example, the name and tags that are displayed on PyPI)
- input files.

Choosing a build backend

You can choose from a number of backends, all of them will work identically

- Hatchling
- Setuptools
- Flit
- PDM

Choosing a build backend

Add the corresponding code to the `pyproject.toml`

Setuptools

```
[build-system]
requires = ["setuptools>=61.0"]
build-backend = "setuptools.build_meta"
```

Configuring metadata

```
[build-system]
requires = ["setuptools>=61.0"]
build-backend = "setuptools.build_meta"
[project]
name = "myCalculatorPackage"
version = "0.0.1"
authors = [
  { name="Khalad Hasan", email="khalad.hasan@gmail.com" },
]
description = "An example package"
readme = "README.md"
```

Configuring metadata

```
requires-python = ">=3.8"
classifiers = [
    "Programming Language :: Python :: 3",
    "License :: OSI Approved :: MIT License",
    "Operating System :: OS Independent",
]
[project.urls]
Homepage = "https://github.com/khalad-hasan/myCalculatorPackage"
Issues = "https://github.com/khalad-hasan/myCalculatorPackage/issues"
```

Descriptions of the Parameters

Parameter	Comments
name	Distribution name of your package and how your project is uniquely listed on PyPI. This can be any name as long as it only contains letters, numbers, ., _ , and -.
version	This is the current version of your project, allowing your users to determine whether or not they have the latest version.
authors	Identify the author of the package; you specify a name and an email for each author
description	A short, one-sentence summary of the package
readme	a path to a file containing a detailed description of the package

Descriptions of the Parameters

Parameter	Comments
requires-python	Gives the versions of Python supported by your project.
classifiers	Gives the index and pip some additional metadata about your package More information: https://pypi.org/classifiers/
urls	lets you list any number of extra links to show on PyPI. Generally this could be to the source, documentation, issue trackers, etc.

README File

This is an optional file

It contains description about the package

It can be a markdown or text file

Licensing

Packaging code is all about reuse and sharing.

A license is required to allow others to reuse your work.

To choose an open source license: https://choosealicense.com/

Popular Licenses:

- Apache License 2.0
- GNU General Public License (GPL)
- MIT license
- Mozilla Public License 2.0
- Eclipse Public License

Copy your selected license full text of the license into a file called LICENSE.txt in the top directory of your project.

Fill in your name and the date on the copyright line.

Upload the Package to PyPi

Navigate to your project folder

```
cd "C://PATH//TO//YOUR//FOLDER"
```

Create a source distribution (Navigate to the directory where the setup.py file is and run the following command):

```
python setup.py sdist bdist_wheel
```

We will need `twine` for the upload process, install `twine` via `pip`:

```
pip install twine
```

Then, run the following command to upload your package to PyPi:

```
twine upload dist/*
```

Upload the Package to PyPi

Navigate to your project folder

```
cd "C://PATH//TO//YOUR//FOLDER"
```

Make sure you have the latest version of PyPA's build installed:

```
pip install --upgrade build
```

Now run this command from the same directory where `pyproject.toml` is located:

```
python -m build
```

Upload the Package to PyPi

This command should output a lot of text and once completed should generate two files in the `dist` directory:

```
dist/
├── myCalculatorPackage-0.0.2.tar.gz
└── myCalculatorPackage-0.0.2-py3-none-any.whl
```

The `tar.gz` file is a source distribution whereas the `.whl` file is a built distribution.

Newer `pip` versions preferentially install built distributions, but will fall back to source distributions if needed.

Upload the Package to PyPi

We will need `twine` for the upload process, install `twine` via `pip`:

```
pip install twine
```

Then, run the following command to upload your package to PyPi:

```
twine upload dist/*
```

Output:

```
Uploading distributions to https://upload.pypi.org/legacy/
Enter your username: khalad
Enter your password:
Uploading myCalculatorPackage-0.0.2-py3-none-any.whl
```

With two factor authentication

Without a two factor authentication, you will be prompted for a username and password.

You can set the two-factor authentication in pypi to avoid entering a username and password

- Login to pypi -> Account settings -> Add API token
- For the username, use `__token__`
- For the password, use the token value, including the `pypi-` prefix.
- Save the information to the .pypirc file

.pypirc file

Store your PyPi settings used to upload new Python packages

Primarily it is used to store your private token to be used when uploading packages

The default location is:

- Linux and MacOS: $HOME/. pypirc.
- Windows: %USERPROFILE%\. pypirc.

Typical PyPi .pypirc file contents

Typically the PyPi file will include information for the `pypi` and `testpypi` server.

```
[distutils]
index-servers =
    pypi
    testpypi
```

Typical PyPi .pypirc file contents

Typically the PyPi file will include information for the `pypi` and `testpypi` server.

```
[pypi]
repository = https://upload.pypi.org/legacy/
username = __token__
password = <PyPI token>

[testpypi]
repository = https://test.pypi.org/legacy/
username = __token__
password = <PyPI token>
```

Typical PyPi .pypirc file contents

If the file is configured correctly, we don’t need to type username and password.

It will show the following output when executing the following command

```
twine upload dist/*
```

Output:

```
Uploading distributions to https://upload.pypi.org/legacy/
Uploading myCalculatorPackage-0.0.2-py3-none-any.whl
Uploading myCalculatorPackage-0.0.2.tar.gz
```

Try Out the Package

Installing the package:

```
sudo pip install python-package-example
```

or

```
pip install python-package-example
```

Using the package:

```
from myCalculatorPackage.division import divide as dv
dv.divide_by_three(9)
```

Using TestPyPI with Twine

You can upload your distributions to TestPyPI using twine by specifying the --repository flag:

```
twine upload --repository testpypi dist/*
```

You can tell pip to download packages from `TestPyPI` instead of `PyPI` by specifying the `--index-url` flag:

```
pip install --index-url https://test.pypi.org/simple/ your-package
```

Change to Your package

You may will need to change the source code form time to time.

Increase the version number.

Create a new build

```
python -m build
```

Upload the `dist` folder with `twine`

```
twine upload dist/*
```

Update your package via `pip`:

```
pip install YOURPACKAGE --upgrade
```

Package Question

Question: How many of the following statements are TRUE for `setup.py` file?

1) `author` and `author_email` are used to identify the author of the package.

2) `description` is a short summary of the package.

3) `name` is the name of your package - It also must not already taken on pypi.org.

4) `version` is the package's version.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Try it: Distributing Python Packages

Question: Publish your python package that you created as a part of this course.

Ask other students to install your package and use it.

Objectives

- Understand Python Package Index (PyPI)
- Learn how to create a python package
- Learn how to upload the package to PyPi
- Be able to install the uploaded package using pip
- Modify the package, re-upload and re-use it


UBC
THE UNIVERSITY OF BRITISH COLUMBIA