Python advanced class

Module 8, Organizing projects

- Virtual environment
- Third-party libraries
- Typical file structure for projects
- Building sdist and whl packages
- Publishing to PyPI

Virtual environment

Creating environment

- Virtual environments allow us to install packages specific for the project
- A virtual environment is a subdir where packages are installed
- After the environment is created it must be activated (and each time the project is worked on)

```
C:\projectdir> python -m venv env
C:\projectdir> .\env\Scripts\activate.ps1 (eller activate.bat for CMD)
```

```
user@pc:~/projectdir$ python -m venv env
user@pc:~/projectdir$ source ./env/bin/activate
```

Third-party libraries

- To install third-perty libraries use the `pip` tool
  - `pip install package1 package2` to install packages from PyPI
  - `pip install -U package` to update a package to latest version
  - `pip show package` to show information about a particular package
  - `pip list` to list all installed packages
  - ~ `pip search keyword` ~ doesn't work any longer

```
> pip install 'requests>=2.20,<2.29'
...
Successfully installed requests-2.28.2 urllib3-1.26.20
```

requirements.txt

- `pip freeze` outputs a full list of installed packages
- `pip freeze > requirements.txt` directs the output to a file
- This can be saved with the project to recreate the same environment
- `pip install -r requirements.txt` will install the packages listed in the file

Typical file structure for projects

Basic structure

- The project directory can be called anything, but it should be called the same as the project name
- Inside is a project description/control file, a README file, and other files
- The package should be placed in a `src/` directory
- A `tests/` directory with test scripts

```
MyPackage
+-- pyproject.toml
+-- README.md
+-- src/
|   +-- mypackage/
|       +-- __init__.py
|       +-- other.py
+-- tests/
    +-- test_this.py
    +-- test_that.py
```

The source files

- As described earlier, a package is a directory with a file called `__init__.py`
- This package files are typically placed in a `src/package/` directory
- Typically will the `__init__.py` file hold the version in the `__version__` variable

In [1]:
```
__version__ = '0.1.21'
```

README.md, License, and documentation

- The project should also contain a `README.md` file written in Markdown format
- A `LICENCE` file describing the condition for use (Open Source licence/confidentiality information)
- A full manual may be placed in a `documentation/` directory in ReST format (out of the scope)

Setuptools

pyproject.toml

- The `pyproject.toml` file contains all relevant information about the project
- It consist of a number of sections for the build tool and other processors

Build system

- The first section is about the choice of build system
- We choose the simplest and most vanilla system, setuptools

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

Project name, description, author(s) and version

- This information is central to find meta data about the project

```
[project]
name = "converterz"
version = "0.1.12"
description = "Converting tool for temperature, distance, and volume"
authors = [
  { name="Example Author", email="author@example.com" },
]
```

Classifiers

- Classifiers are used to inform downloaders about things like maturity, licence type, viablilty

```
classifiers = [
    "Development Status :: 4 - Beta",
    "Programming Language :: Python :: 3",
    "License :: OSI Approved :: MIT License",
    "Operating System :: OS Independent",
]
```

Dependencies and files

- Files, like readme files, should be specified
- Also runtime dependencies (development dependencies should go into `requirements.txt`)

```
readme = "README.md"
requires-python = ">=3.8"

dependencies = [
    "requests>=2.20,<2.29",
    "PyYAML~=6.0.2",
]
```

CLI scripts

- Instead of including script files, it's better to use setuptools to create CLI tools

```
[project.scripts]
converterzcli = "converterz.cli:main"
```

Publishing to PyPI

Build the package

- Check version is set!
- Run `python -m build`

```
shell> python -m build
...
Successfully built converterz-0.1.12.tar.gz and converterz-0.1.12-py3-none-any.whl
```

Create account

- Go to https://test.pypi.org/ and create an account
- Verify email
- Create an API token (and save it!)

create a `.pypirc` file

- Create a `.pypirc` in the root of the homedir (or `C:\Users\YourName\.pypirc` on Windows)
- Place the saved token as `password`

```
[testpypi]
username = __token__
password = pypi-AxEIf5...
```

Push

- Install `twine` package
- Run `twine` specifying the testpypi prepository

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