Creating a Python Library Tutorial

Topics

1. Creating a Python Library
2. Using a Library in an Application

1 Creating a Python Library

Please follow the tutorial on *Packaging Python Projects* available at https://packaging.python.org/tutorials/packaging-projects/.

In conjunction with the tutorial, please consult the sections below for additional information about the sample library we create in this tutorial.

1. Create a Library

   - **Task** Create a simple Python library that determines whether a number is even, is odd, or is a multiple of a given number N. Once the library is fully tested that it works according to the specification we can pack it and reuse it later in other applications.
   - In Python we can distribute and reuse code by creating distribution packages/libraries.
   - The hierarchical file structure for creating a distribution package/library, and in particular the hierarchical file structure of today's library, is depicted below

```
smart_numbers
        ---> numbers_properties_pkg
                ---> __init__.py
                ---> numbers_properties.py
        ---> setup.py
        ---> tests
        ---> README.md
        ---> LICENSE
        ---> CHANGELOG.txt
```

- Next, let's create the directory/file structure for our package/library (i.e. according to the information presented on the previous page). Note that in our example, *smart_numbers* is the "parent" directory/folder that contains all the files and folders pertaining to the library we want to create. Next, in our examples, there is another directory/folder named *numbers_properties_pkg* which contains the actual source code for the library. Usually, the folder *tests* is used for unit test files. All the other items listed on the previous page are files. If you use the terminal to create the hierarchical file structure, then you can use the command *mkdir* to create a directory, and the command *touch* to create a file.
  - For example, to create the *smart_numbers* directory at the terminal run the following command:
    $ mkdir smart_numbers
  - **Note** you should run all the commands pertaining to the creation of the library (for example, the command to build/generate the distribution packages) from the parent folder of your library (i.e. in our example, that is the folder *smart_numbers*). Therefore your current working directory should be in our example, *smart_numbers*.
    $ cd smart_numbers

2. Let's write the code for our library.

   (a) Create the directory named *numbers_properties_pkg*. This will serve as the "parent" i.e. top-level package for all the modules and packages[1] we create.
   $ mkdir numbers_properties_pkg

   (b) Next, in the *numbers_properties_pkg* folder create the files which will contain the source code for the library. Remember to navigate to that folder if you use the terminal to create the files i.e.
   $ cd numbers_properties_pkg

   (c) In the *numbers_properties_pkg* folder, create the *__init__.py* file. The *__init__.py* files are required to enable Python to treat directories containing this file as packages.
   $ touch __init__.py

[1] https://docs.python.org/3/tutorial/modules.html

(d) In the *numbers_properties_pkg* folder, create the *numbers_properties.py* file, by running at the terminal the following command

$ touch numbers_properties.py

(e) Open the *numbers_properties.py* file, and implement a solution to determine whether a number is even, or is odd, or is multiple of a given number *N*. A sample solution is included below.

```
class NumberProperties:

    def is_even(self, number):
        remainder = number % 2
        if remainder == 0:
            return True
        else:
            return False


    def is_odd(self, number):
        remainder = number % 2
        if remainder != 0:
            return True
        else:
            return False


    def is_multiple_of_n(self, number, n):
        remainder = number % n
        if remainder == 0:
            return True
        else:
            return False



if __name__ == '__main__':
    np = NumberProperties()
    number = 24
    n = 5
```

```
print('\n{} is even?: {}'.format(number, np.is_even(number)))
print('{} is odd?: {}'.format(number, np.is_odd(number)))
property = np.is_multiple_of_n(number, n)
print('{} is multiple of {}?: {}'.format(number, n, property))

number = 25
n = 5
print('\n{} is even?: {}'.format(number, np.is_even(number)))
print('{} is odd?: {}'.format(number, np.is_odd(number)))
property = np.is_multiple_of_n(number, n)
print('{} is multiple of {}?: {}'.format(number, n, property))

number = 31
n = 5
print('\n{} is even?: {}'.format(number, np.is_even(number)))
print('{} is odd?: {}'.format(number, np.is_odd(number)))
property = np.is_multiple_of_n(number, n)
print('{} is multiple of {}?: {}'.format(number, n, property))
```

3. Let's create the *setup.py* file. *setup.py* is the build script for setuptools[2]. The *setup.py* file provides information about your distribution package, and also which source code files to be included in the distribution package/library.

   (a) In the *smart_numbers* folder, create the *setup.py* file, by running at the terminal the following command (recall that your current working directory should be *smart_numbers*)

   $ touch setup.py

   (b) Open the *setup.py* file, include the following content and update the required information with your own information about your distribution package/library. Note that the name of your distribution package/library should be unique! For the remaining of this tutorial, we assume that the value of the *name* argument in the *setup.py* file is 'example-pkg-aec'.

[2] https://packaging.python.org/key_projects/#setuptools

```
import setuptools

with open("README.md", "r", encoding="utf-8") as fh:
    long_description = fh.read()

setuptools.setup(
    name="example-pkg-YOUR-USERNAME-HERE",
    # Replace with your own username above
    version="0.0.1",
    author="Example Author",
    author_email="author@example.com",
    description="A small example package",
    long_description=long_description,
    long_description_content_type="text/markdown",
    url="https://github.com/pypa/sampleproject",
    packages=setuptools.find_packages(),
    # if you have libraries that your module/package/library
    #you would include them in the install_requires argument
    install_requires=[''],
    classifiers=[
        "Programming Language :: Python :: 3",
        "License :: OSI Approved :: MIT License",
        "Operating System :: OS Independent",
    ],
    python_requires='>=3.6',
)
```

Don't forget to consult the tutorial on *Packaging Python Projects* available at https://packaging.python.org/tutorials/packaging-projects/ for details, in particular for the items 3 to 7 of this list.

4. Create all the other files according to the hierarchical file structure presented on page one of this tutorial.

5. Build the library i.e. generate the distribution package.

    - **Prerequisite!** In order to be able to create a package distribution/library we need to have the libraries *setuptools* and *wheel* installed on our machine. If you use a Python virtual environment

to create your library then install them running the following command from the terminal:

```
$  python3 -m pip install --upgrade setuptools wheel
```

If you don't use a Python virtual environment to create your library then install using the following command

```
$  python3 -m pip install --user --upgrade setuptools wheel
```

6. Upload/publish the distribution package files

   - Usually, we publish our distribution packages/libraries via the *Python Package Index (PyPI)*[3]. However, as in this tutorial we all create the very same library for educational purposes, we will not publish it to PyPI. Instead, we will publish the library from today's tutorial to Test PyPI[4], which is a distinct instance of the Python Package Index created for testing and experimentation.
   - Note that for your project, you would publish your Python library/libraries to PyPI.
   - **Prerequisites!**
     (a) You need to create an account on Test PyPI and an account on PyPI to be able to publish your libraries to Test PyPI and PyPI respectively.
     (b) To publish the distribution files/library we need to have *twine* installed on our machine.
     If you use a Python virtual environment to create and publish your library then install *twine* running the following command from the terminal:

     ```
     $  python3 -m pip install --upgrade twine
     ```

     If you don't use a Python virtual environment to create and publish your library then install using the following command

     ```
     $  python3 -m pip install --user --upgrade twine
     ```

     (c) Publishing your distribution package/library

[3] https://pypi.org/
[4] https://test.pypi.org

- To publish your library to Test PyPI run the following command form the terminal

```
$  python3 -m twine upload --repository testpypi dist/*
```

  * Note that you will be prompted for your Test PyPi credentials (you can either use an API token or your account credentials)

- To publish your library to PyPI run the following command form the terminal

```
$  twine upload dist/*
```

  * Note that you will be prompted for your PyPi credentials (you can either use an API token or your account credentials)

7. Install (i.e. via pip) the distribution package/library you just published (i.e. at item 6)

   (a) You have to install the library on the machine where you develop your projects so that you can reuse the library, in our case that is the EC2 instance that hosts Cloud9. Let's assume that the value you provided for the *name* argument in the *setup.py* file is 'example-pkg-aec' (see item 3).

   - If you published the library to Test PyPI then you would install the library by running from the terminal the following command (note that below there is only one command written on two lines! – ensure that you press the *Enter* key only after you have finished writing the entire command!):

```
$  python -m pip install --index-url https://test.pypi.org/simple/
--no-deps example-pkg-aec
```

   where

   `--index-url https://test.pypi.org/simple/`

   informs pip that it should install the package from Test PyPI

   - If you published the library to PyPI then you would install the library by running from the terminal the following command: **$ pip install example-pkg-aec**

(b) We can use *pip* to display all the installed Python packages/libraries on our platform by running the following command from the terminal

$ pip list

(c) We can use *pip* to display information about one or more installed packages/libraries. Let's display the information about our library, in this example, 'example-pkg-aec'. Recall to use the same name as the one you provided for the *name* argument in the *setup.py* file (see item 3).

$ pip show example-pkg-aec

(d) Once the library is installed, you can quickly verify that the library was properly installed and that you can use the library by using the Python interpreter in interactive mode. To use the Python interpreter in interactive mode from the terminal run *python* without any other arguments.

$ python

- Once you run the above command, you are in the Python interpreter shell (note that the prompt changed, you should see three greater than signs (i.e. >>>)
- In the Python interpreter shell, import the package or module that you'd like to use from your library. In our example, let's import the module *numbers_properties* from the package *numbers_properties_pkg* by executing the following statement
  >>> **from numbers_properties_pkg import numbers_properties**
- Next we could, for example, instantiate the class (i.e. *NumberProperties*) declared in the imported module (i.e. *numbers_properties*), and call any of the available methods on that instance/object by executing the following statement
  >>> **numbers_properties.NumberProperties().is_even(4)**
  - if the module was successfully imported, for the above example you should see *True* as an output as 4 is an even number
- Exit the interactive mode by pressing CTRL + d at the same time, or by running the command quit()

2 Using a Library in an Application

Prerequisite: This task assumes that you have already completed section 1.

Task Create a Python application which uses the library, in this example 'example-pkg-aec', created in section 1 to determine whether a number is even, is odd, or is multiple of a given number *N*. First, the application should prompt the user to provide two numbers. Next, the application uses the library to verify whether the first number is even, is odd, or is a multiple of the second number, and will inform the user about the results.

Let's create an application according to the above requirements

- Create a top-level directory/folder that will serve as the "parent" i.e. top-level folder for the application

  $ mkdir smart_number_app

- Navigate to the parent folder of the application

  $ cd smart_number_app

- We will develop this application using a virtual environment. Recall that in Python virtual environments allow us to manage separate package installations for different projects (see *Appendix: Python – Working with virtual environments* for details). Create a virtual environment by running the following command from the terminal

  $ python3 -m venv env

- Activate the virtual environment. Recall that once the virtual environment is activated the terminal's prompt is prefixed by the name of your virtual environment (i.e. (env) in this example).

  $ source env/bin/activate

- Create a directory/folder that will contain all the relevant files (i.e. source code) of the application

  $ mkdir app

- Navigate to that folder

  $ cd app

- Install (i.e. via pip) the distribution package/library, in this example 'example-pkg-aec', to be used in the application. Consult section 1, item 7 for detailed information how to install the Python package/library.

- Now, that the library is installed we can use it in our application. Let's implement an application which works according to the specified requirements. Create a file named *numbers_ app.py*

  $ touch number_app.py

- Open the *number_ app.py* file, and implement a solution according to the specified requirements. A sample solution is included below.

```
# import from the installed library, in this tutorial, example-pkg-aec

from numbers_properties_pkg.numbers_properties import NumberProperties


class NumberApp:

    def check_properties(self, first_number, second_number):
        n = NumberProperties()

        if n.is_even(first_number):
            print(f'{first_number} is even.')
            print(f'{first_number} is not odd.')
        else:
            print(f'{first_number} is not even.')
            print(f'{first_number} is odd.')

        if n.is_multiple_of_n(first_number, second_number):
            print(f'{first_number} is a multiple of {second_number}.')
        else:
            print(f'{first_number} is not a multiple of {second_number}.')
```

```
def main():
    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument('first_number', help='The number to
    check if it\'s even, odd, or a multiple of another given
    second_number.')
    parser.add_argument('second_number', help='The number to
    check if the first_number is a multiple of.')

    args = parser.parse_args()


    my_n_app = NumberApp()
    number1 = int(args.first_number)
    number2 = int(args.second_number)
    my_n_app.check_properties(number1, number2)




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

- Run the application from the terminal with suitable input data

  $ python number_app.py 25 5

Resources

- Packaging Python Projects https://packaging.python.org/tutorials/packaging-projects/
- Test Python package publishing with the Test Python Package Index https://test.pypi.org/
- Python Package Index https://pypi.org/

- pip - The Python Package Installer https://pip.pypa.io/en/stable/
- Modules https://docs.python.org/3/tutorial/modules.html
- pytest: helps you write better programs https://docs.pytest.org/en/stable/

Appendix: Python – Working with virtual environments

Virtual environments allow us to manage separate package installations for different projects. For more information please consult https://docs.python.org/3/tutorial/venv.html and https://packaging.python.org/guides/installing-using-pip-and-virtual-environments/

1. Create a virtual environment: to create a virtual environment, navigate (by using the command *cd*) to your project's directory and run the command *venv* at the terminal. The following command will create the virtual environment named env.

   ```
   $ python3 -m venv env
   ```

2. Activate a virtual environment: before we can start installing or using packages in the virtual environment we have to activate the virtual environment we created, by running the following command at the terminal

   ```
   $ source env/bin/activate
   ```

   Note that once the virtual environment is activated the terminal's prompt is prefixed by the name of your virtual environment (i.e. (env) in this example).

3. Deactivate the virtual environment: when you want to exit the virtual environment (for example when you finish a working session, or would like to work on another project) run the following command at the terminal

   ```
   $ deactivate
   ```

4. Each time you want to enter again the virtual environment for your project follow step 2. Note that typically we run only once *item 1* per project, and as many times needed *item 2* and *item 3*.