**Portfolio Exercise: Upload a Package to PyPi**

Personal portfolios are an excellent way to demonstrate your knowledge and creativity. In fact, they are little by little becoming a must-have for people working in the tech industry. In this portfolio building exercise, you will build a Python package and upload the package to PyPi.

**NOTE that a portfolio exercise like this is NOT reviewed. So you will not submit your work on this, and you do not need to complete this assignment in order to graduate.**

**Getting Started**

Next, you'll find a blank classroom workspace where you can do your work. The benefits of using the workspace over your own computer are two fold:

* you shouldn't run into set-up issues
* if you install something on the workspace that causes issues, you can always reset everything

The workspace is the same Ubuntu Linux environment you've been using to do the exercises. You are also welcome to develop your package locally on your own computer if you prefer.

For local development, if you are developing on macOS, you can use the exact same terminal commands. On a windows machine, the commands are slightly different and you'll need to use the command prompt. This link contains a [**comparison of MS-DOS vs Linux commands**](https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/4/html/Step_by_Step_Guide/ap-doslinux.html).

If at some point you decide to reset your workspace, make sure you have saved your work somewhere else like in a GitHub repository; otherwise, all of your work will be deleted. You can connect to a remote GitHub repo from within the workspace terminal. Here is a reminder of how to do that: [**adding an existing repository to GitHub using the command line**](https://help.github.com/articles/adding-an-existing-project-to-github-using-the-command-line/). Another option is to download files or folders directly from the workspace. You right click on the file/folder name and click "Download".

In the workspace, you'll see that the files and folders have already been set up for you in terms of the structure, but the files themselves are empty.

Every time you type in the workspace, your work is automatically saved. But you should still save and commit your work to GitHub as well.

**Instructions**

Build a Python package and upload the package to PyPi. We encourage you to use object-oriented programming and tackle a problem that interests you. The Introduction to Object-Oriented Programming lesson has all of the information needed to create a package and upload the package. As mentioned in the lesson material, you'll need to use a unique name to upload the package to PyPi since each package needs a unique name; hence 'dsnd-probability' will not work since that package name is already taken.

Here are a few ideas for packages:

* create a package that does basic matrix algebra such as addition, subtraction, multiplication, matrix inversion, etc.
* make a calculus package that implements algorithms such as Newton's method
* create a package built on top of Matplotlib or Seaborn that creates well-formatted, well labeled visualizations
* create an object-oriented package to play a game of tic-tac-toe between two human players.
* or make a number guessing game where the computer randomly chooses an integer and then tells a human player if a guess is higher or lower than the number

Feel free to come up with your own ideas. The main goal is to develop and demonstrate your skills in object-oriented programming and uploading packages to PyPi.

**Reminders**

* include a README file detailing the files in your package and how to install the package.
* Comment your code - use docstrings and inline comments where appropriate.
* Refactor code when possible - if you find your functions are getting too long, then refactor your code!
* Use object-oriented programming whenever it makes sense to do so.
* You're encouraged to write unit tests! The coding exercises in this lesson contained unit tests, so you can use those tests as a model for your package.
* Use GitHub for version control, and commit your work often.

As a reminder, your package should be placed in a folder with the following folders and files:

* a folder with the name of your package that contains:
  + the Python code that makes up your package
  + a README.md file
  + an \_\_init\_\_.py
  + license.txt
  + setup.cfg
* setup.py file

**Hints and Helpful Links**

Because this exercise requires writing and organizing code in a specific way, you might have to rewatch some of the lecture videos especially the "Putting Code on PyPi" concept.

Before you upload your code to PyPi, you should first pip install the package locally to make sure everything works as expected. The "Making a Package" and "Virtual Environments" lesson concepts should be helpful. You should also consider writing unit tests to test the functionality of your package. In the object-oriented programming lesson workspace, there were unit tests inside the 4a\_binomial\_package folder that you can use to help you get started. Those were in a file called test.py.

The object-oriented programming lesson included a complete, working package called dsnd-probability. We encourage you to code a project from scratch; however, if you get stuck, use the dsnd-probability package code as a template. It already contains all of the necessary files you'll need for creating a package. It's also a simple example of object-oriented code. You can use these files, including the setup.py and setup.cfg files, to help structure your own code.

Those files are located in the "Exercise: Upload to PyPi" section inside the "5\_exercise\_upload\_to\_pypi" folder.

Besides the lesson on object-oriented programming, you might find [**this package building summary guide from the Python website**](https://packaging.python.org/guides/distributing-packages-using-setuptools/) helpful.

For a much more detailed explanation of distributing Python packages, check out the documentation on Distutils.

1. [**Introduction**](https://docs.python.org/3/distutils/introduction.html)
2. [**setup.py script**](https://docs.python.org/3/distutils/setupscript.html)
3. [**config file**](https://docs.python.org/3/distutils/configfile.html)
4. [**source distributions**](https://docs.python.org/3/distutils/sourcedist.html)
5. [**built distributions**](https://docs.python.org/3/distutils/builtdist.html)
6. [**uploading to PyPi**](https://docs.python.org/3/distutils/packageindex.html)

**Uploading to PyPi**

When you are ready to upload your package, you can first upload to the **[PyPi test repository](https://test.pypi.org/" \t "_blank)**. Once everything is working as expected, you can upload to the public facing **[PyPi repository](https://pypi.org/" \t "_blank)**.

As a reminder, you'll need to create a username for both the test and public facing repositories. You'll also need to pip install the twine package with: pip install twine. You can rewatch the lesson videos to see how to upload your package.

Continue on to the next sections to get some troubleshooting tips, and access the project workspace. When you're finished with your project, show off your work on a personal website, GitHub, and LinkedIn.

If you are having trouble uploading your package to PyPi, here are a few issues that other students have faced.

# PyPi vs. Test PyPi

Note that [**pypi.org**](https://pypi.org/) and [**test.pypy.org**](https://test.pypi.org/) are two different websites. You'll need to register separately at each website. If you only register at [**pypi.org**](https://pypi.org/), you will not be able to upload to the [**test.pypy.org**](https://test.pypi.org/) repository.

# Unique Name

Remember that your package name must be unique. If you use a package name that is already taken, you will get an error when trying to upload the package.

# Re-uploading and Versioning

Once you upload your package to PyPi, you cannot upload the same version again. All that means is that you need to go into your setup.py file and change the version number. For example, if you uploaded a package with version = 0.1.1, then you'll need to change this to something else like version = 0.1.2.

If you've ever wondered what all these version numbers mean, here is a tutorial about [**semantic versioning**](https://semver.org/). Semantic versioning is a standard for deciding when to move up to the next number in your versioning scheme.