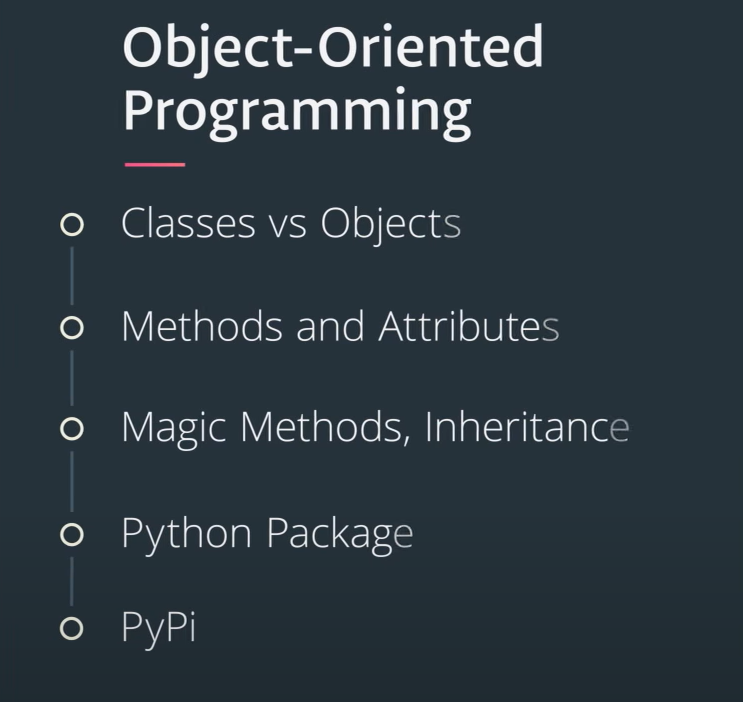
**Lesson Outline**

* Object-oriented programming syntax
  + procedural vs object-oriented programming
  + classes, objects, methods and attributes
  + coding a class
  + magic methods
  + inheritance
* Using object-oriented programming to make a Python package
  + making a package
  + tour of scikit-learn source code
  + putting your package on PyPi



**Why Object-Oriented Programming?**

Object-oriented programming has a few benefits over procedural programming, which is the programming style you most likely first learned. As you'll see in this lesson,

* object-oriented programming allows you to create large, modular programs that can easily expand over time;
* object-oriented programs hide the implementation from the end-user.

Consider Python packages like [**Scikit-learn**](https://github.com/scikit-learn/scikit-learn), [**pandas**](https://pandas.pydata.org/), and [**NumPy**](http://www.numpy.org/). These are all Python packages built with object-oriented programming. Scikit-learn, for example, is a relatively large and complex package built with object-oriented programming. This package has expanded over the years with new functionality and new algorithms.

When you train a machine learning algorithm with Scikit-learn, you don't have to know anything about how the algorithms work or how they were coded. You can focus directly on the modeling.

Here's an example taken from the [**Scikit-learn website**](http://scikit-learn.org/stable/modules/svm.html):

from sklearn import svm

X = [[0, 0], [1, 1]]

y = [0, 1]

clf = svm.SVC()

clf.fit(X, y)

How does Scikit-learn train the SVM model? You don't need to know because the implementation is hidden with object-oriented programming. If the implementation changes, you as a user of Scikit-learn might not ever find out. Whether or not you SHOULD understand how SVM works is a different question.

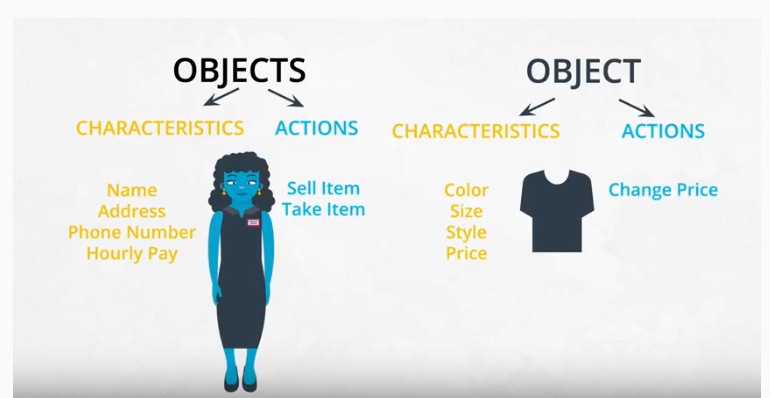
In this lesson, you'll practice the fundamentals of object-oriented programming. By the end of the lesson, you'll have built a Python package using object-oriented programming.

**Lesson Files**

This lesson uses classroom workspaces that contain all of the files and functionality you will need. You can also find the files in the [**data scientist nanodegree term 2 GitHub repo**](https://github.com/udacity/DSND_Term2/tree/master/lessons/ObjectOrientedProgramming).

# Objects are defined by characteristics and actions

Here is a reminder of what is a characteristic and what is an action.



Objects are defined by their characteristics and their actions

# Characteristics and Actions in English Grammar

Another way to think about characteristics and actions is in terms of English grammar. A characteristic would be a noun. On the other hand, an action would be a verb.

Let's pick something from the real-world: a dog. A few characteristics could be the dog's weight, color, breed, and height. These are all nouns. What actions would a dog take? A dog can bark, run, bite and eat. These are all verbs.

**Object-Oriented Programming (OOP) Vocabulary**

* class - a blueprint consisting of methods and attributes
* object - an *instance* of a class. It can help to think of objects as something in the real world like a yellow pencil, a small dog, a blue shirt, etc. However, as you'll see later in the lesson, objects can be more abstract.
* attribute - a descriptor or characteristic. Examples would be color, length, size, etc. These attributes can take on specific values like blue, 3 inches, large, etc.
* method - an action that a class or object could take
* OOP - a commonly used abbreviation for object-oriented programming
* encapsulation - one of the fundamental ideas behind object-oriented programming is called encapsulation: you can combine functions and data all into a single entity. In object-oriented programming, this single entity is called a class. Encapsulation allows you to hide implementation details much like how the scikit-learn package hides the implementation of machine learning algorithms.

In English, you might hear an attribute described as a *property*, *description*, *feature*, *quality*, *trait*, or *characteristic*. All of these are saying the same thing.

Here is a reminder of how a class, object, attributes and methods relate to each other.



A class is a blueprint consisting of attributes and methods.

# Function vs Method

In the video above at 1:44, the dialogue mistakenly calls ***init*** a function rather than a method. Why is ***init*** not a function?

A function and a method look very similar. They both use the def keyword. They also have inputs and return outputs. The difference is that a method is inside of a class whereas a function is outside of a class.

# What is self?

If you instantiate two objects, how does Python differentiate between these two objects?

shirt\_one = Shirt('red', 'S', 'short-sleeve', 15)

short\_two = Shirt('yellow', 'M', 'long-sleeve', 20)

That's where self comes into play. If you call the change\_price method on shirt\_one, how does Python know to change the price of shirt\_one and not of shirt\_two?

shirt\_one.change\_price(12)

Behind the scenes, Python is calling the change\_price method:

**def** **change\_price**(self, new\_price):

self.price = new\_price

Self tells Python where to look in the computer's memory for the shirt\_one object. And then Python changes the price of the shirt\_one object. When you call the change\_price method, shirt\_one.change\_price(12), self is implicitly passed in.

The word self is just a convention. You could actually use any other name as long as you are consistent; however, you should always use self rather than some other word or else you might confuse people.

# Set and Get methods

The last part of the video mentioned that accessing attributes in Python can be somewhat different than in other programming languages like Java and C++. This section goes into further detail.

The shirt class has a method to change the price of the shirt: shirt\_one.change\_price(20). In Python, you can also change the values of an attribute with the following syntax:

shirt\_one.price = 10

shirt\_one.price = 20

shirt\_one.color = 'red'

shirt\_one.size = 'M'

shirt\_one.style = 'long\_sleeve'

This code accesses and changes the price, color, size and style attributes directly. Accessing attributes directly would be frowned upon in many other languages **but not in Python**. Instead, the general object-oriented programming convention is to use methods to access attributes or change attribute values. These methods are called set and get methods or setter and getter methods.

A get method is for obtaining an attribute value. A set method is for changing an attribute value. If you were writing a Shirt class, the code could look like this:

**class** **Shirt**:

**def** **\_\_init\_\_**(self, shirt\_color, shirt\_size, shirt\_style, shirt\_price):

self.\_price = shirt\_price

**def** **get\_price**(self):

**return** self.\_price

**def** **set\_price**(self, new\_price):

self.\_price = new\_price

Instantiating and using an object might look like this:

shirt\_one = Shirt('yellow', 'M', 'long-sleeve', 15)

print(shirt\_one.get\_price())

shirt\_one.set\_price(10)

In the class definition, the underscore in front of price is a somewhat controversial Python convention. In other languages like C++ or Java, price could be explicitly labeled as a private variable. This would prohibit an object from accessing the price attribute directly like shirt\_one.\_price = 15. However, Python does not distinguish between private and public variables like other languages. Therefore, there is some controversy about using the underscore convention as well as get and set methods in Python. Why use get and set methods in Python when Python wasn't designed to use them?

At the same time, you'll find that some Python programmers develop object-oriented programs using get and set methods anyway. Following the Python convention, the underscore in front of price is to let a programmer know that price should only be accessed with get and set methods rather than accessing price directly with shirt\_one.\_price. However, a programmer could still access \_price directly because there is nothing in the Python language to prevent the direct access.

To reiterate, a programmer could technically still do something like shirt\_one.\_price = 10, and the code would work. But accessing price directly, in this case, would not be following the intent of how the Shirt class was designed.

One of the benefits of set and get methods is that, as previously mentioned in the course, you can hide the implementation from your user. Maybe originally a variable was coded as a list and later became a dictionary. With set and get methods, you could easily change how that variable gets accessed. Without set and get methods, you'd have to go to every place in the code that accessed the variable directly and change the code.

You can read more about get and set methods in Python on this [**Python Tutorial site**](https://www.python-course.eu/python3_properties.php).

# A Note about Attributes

There are some drawbacks to accessing attributes directly versus writing a method for accessing attributes.

In terms of object-oriented programming, the rules in Python are a bit looser than in other programming languages. As previously mentioned, in some languages, like C++, you can explicitly state whether or not an object should be allowed to change or access an attribute's values directly. Python does not have this option.

Why might it be better to change a value with a method instead of directly? Changing values via a method gives you more flexibility in the long-term. What if the units of measurement change, like the store was originally meant to work in US dollars and now has to handle Euros? Here's an example:

#### Example Dollars versus Euros

If you've changed attribute values directly, you'll have to go through your code and find all the places where US dollars were used, like:

shirt\_one.price = 10 *# US dollars*

and then manually change to Euros

shirt\_one.price = 8 *# Euros*

If you had used a method, then you would only have to change the method to convert from dollars to Euros.

**def** **change\_price**(self, new\_price):

self.price = new\_price \* 0.81 *# convert dollars to Euros*

shirt\_one.change\_price(10)

For the purposes of this introduction to object-oriented programming, you will not need to worry about updating attributes directly versus with a method; however, if you decide to further your studies of object-oriented programming, especially in another language such as C++ or Java, you'll have to take this into consideration.

# Modularized Code

Thus far in the lesson, all of the code has been in Jupyter Notebooks. For example, in the previous exercise, a code cell loaded the Shirt class, which gave you access to the Shirt class throughout the rest of the notebook; however, if you were developing a software program, you would want to modularize this code.

You would put the Shirt class into its own Python script called, say, shirt.py. And then in another Python script, you would import the Shirt class with a line like: from shirt import Shirt.

For now, as you get used to OOP syntax, you'll be completing exercises in Jupyter notebooks. But midway through the lesson, you'll modularize object-oriented code into separate files.

# Commenting Object-Oriented Code

Did you notice anything special about the answer key in the previous exercise? The Pants class and the SalesPerson class contained docstrings! A docstring is a type of comment that describes how a Python module, function, class or method works. Docstrings, therefore, are not unique to object-oriented programming. This section of the course is merely reminding you to use docstrings and to comment your code. It's not just going to help you understand and maintain your code. It will also make you a better job candidate.

From this point on, please always comment your code. Use both in-line comments and document level comments as appropriate.

Check out this [**link**](http://sphinxcontrib-napoleon.readthedocs.io/en/latest/example_google.html) to read more about docstrings.

**Docstrings and Object-Oriented Code**

Below is an example of a class with docstrings and a few things to keep in mind:

* Make sure to indent your docstrings correctly or the code will not run. A docstring should be indented one indentation underneath the class or method being described.
* You don't have to define 'self' in your method docstrings. It's understood that any method will have self as the first method input.

**class** **Pants**:

"""The Pants class represents an article of clothing sold in a store

"""

**def** **\_\_init\_\_**(self, color, waist\_size, length, price):

"""Method for initializing a Pants object

Args:

color (str)

waist\_size (int)

length (int)

price (float)

Attributes:

color (str): color of a pants object

waist\_size (str): waist size of a pants object

length (str): length of a pants object

price (float): price of a pants object

"""

self.color = color

self.waist\_size = waist\_size

self.length = length

self.price = price

**def** **change\_price**(self, new\_price):

"""The change\_price method changes the price attribute of a pants object

Args:

new\_price (float): the new price of the pants object

Returns: None

"""

self.price = new\_price

**def** **discount**(self, percentage):

"""The discount method outputs a discounted price of a pants object

Args:

percentage (float): a decimal representing the amount to discount

Returns:

float: the discounted price

"""

**return** self.price \* (1 - percentage)

# Resources for Review

The example in the next part of the lesson assumes you are familiar with Gaussian and binomial distributions.

Here are a few formulas that might be helpful:

#### Gaussian Distribution Formulas

##### probability density function

f(x \space | \space \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}e^-\frac{(x-\mu)^2}{2\sigma^2}*f*(*x* ∣*μ*,*σ*2)=2*πσ*2​1​*e*−2*σ*2(*x*−*μ*)2​

\begin{aligned} \\ \text{where:} \\ \mu& \space \text{is the mean} \\ \sigma& \space \text{ is the standard deviation} \\ \sigma&^2 \space \text{is the variance} \end{aligned}where:*μσσ*​ is the mean  is the standard deviation2 is the variance​

#### Binomial Distribution Formulas

##### mean

\mu = n \* p*μ*=*n*∗*p*

In other words, a fair coin has a probability of a positive outcome (heads) p = 0.5. If you flip a coin 20 times, the mean would be 20 \* 0.5 = 10; you'd expect to get 10 heads.

##### variance

\sigma^2 = n \* p \* (1 - p)*σ*2=*n*∗*p*∗(1−*p*)

Continuing with the coin example, n would be the number of coin tosses and p would be the probability of getting heads.

##### standard deviation

\sigma = \sqrt{n \* p \* (1 - p)}*σ*=*n*∗*p*∗(1−*p*)​

or in other words, the standard deviation is the square root of the variance.

##### probability density function

f(k, n, p) = \frac{n\footnotesize{!}}{k!(n - k)!}p^k(1-p)^{(n-k)}*f*(*k*,*n*,*p*)=*k*!(*n*−*k*)!*n*!​*pk*(1−*p*)(*n*−*k*)

# Further Resources

If you would like to review the Gaussian (normal) distribution and binomial distribution, here are a few resources:

This free Udacity course, [**Intro to Statistics**](https://www.udacity.com/course/intro-to-statistics--st101), has a lesson on Gaussian distributions as well as the Binomial distribution.

This free course, [**Intro to Descriptive Statistics**](https://www.udacity.com/course/intro-to-descriptive-statistics--ud827), also has a Gaussian distributions lesson.

Here are the wikipedia articles:

* [**Gaussian Distributions Wikipedia**](https://en.wikipedia.org/wiki/Normal_distribution)
* [**Binomial Distributions Wikipedia**](https://en.wikipedia.org/wiki/Binomial_distribution)

**Advanced OOP Topics**

Inheritance is the last object-oriented programming topic in the lesson. Thus far you've been exposed to:

* classes and objects
* attributes and methods
* magic methods
* inheritance

Classes, object, attributes, methods, and inheritance are common to all object-oriented programming languages.

Knowing these topics is enough to start writing object-oriented software. What you've learned so far is all you need to know to complete this OOP lesson. However, these are only the fundamentals of object-oriented programming.

Here is a list of resources for advanced Python object-oriented programming topics.

* [**class methods, instance methods, and static methods**](https://realpython.com/instance-class-and-static-methods-demystified/) - these are different types of methods that can be accessed at the class or object level
* [**class attributes vs instance attributes**](https://www.python-course.eu/python3_class_and_instance_attributes.php) - you can also define attributes at the class level or at the instance level
* [**multiple inheritance, mixins**](https://easyaspython.com/mixins-for-fun-and-profit-cb9962760556) - A class can inherit from multiple parent classes
* [**Python decorators**](https://realpython.com/primer-on-python-decorators/) - Decorators are a short-hand way for using functions inside other functions

# Windows vs. macOS vs. Linux

Linux, which our Udacity classroom workspaces use, is an operating system like Windows or macOS. One important difference is that Linux is free and open source whereas Windows is owned by Microsoft and macOS by Apple.

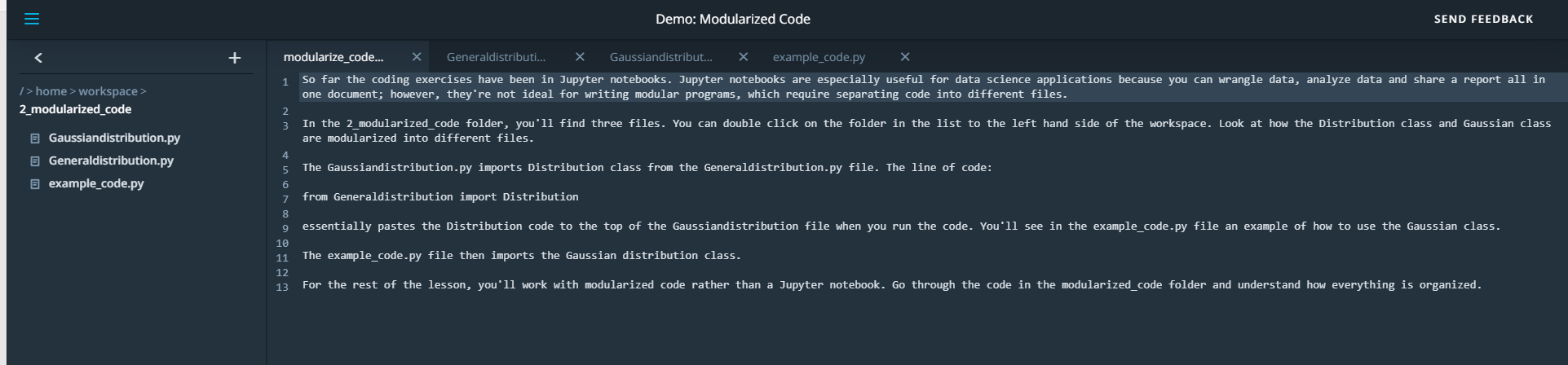
Throughout the lesson, you can do all of your work in a classroom workspace. These workspaces provide interfaces that connect to [**virtual machines in the cloud**](https://aws.amazon.com/getting-started/tutorials/launch-a-virtual-machine/). However, if you want to run this code locally on your computer, the commands to use might be slightly different.

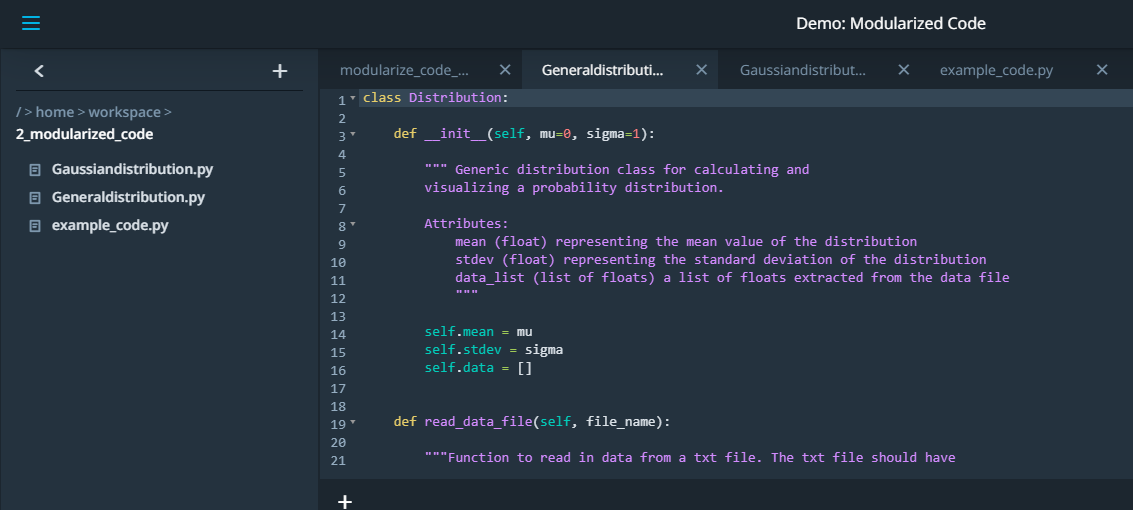
If you are using macOS, you can open an application called "Terminal" and essentially use the same commands that you use in the workspace. That is because [**Linux and MacOS are related**](https://www.quora.com/Is-macOS-essentially-built-on-top-of-Linux).

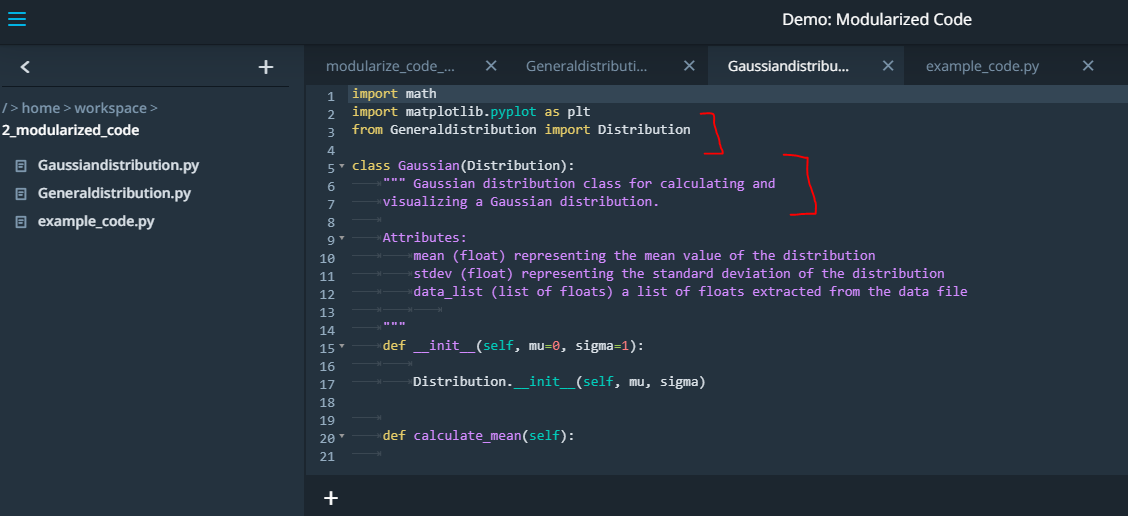
If you are using Windows, the analogous application is called console. The console commands can be somewhat different than the terminal commands. A search engine like Google is your best friend for finding the right commands in a Windows environment.

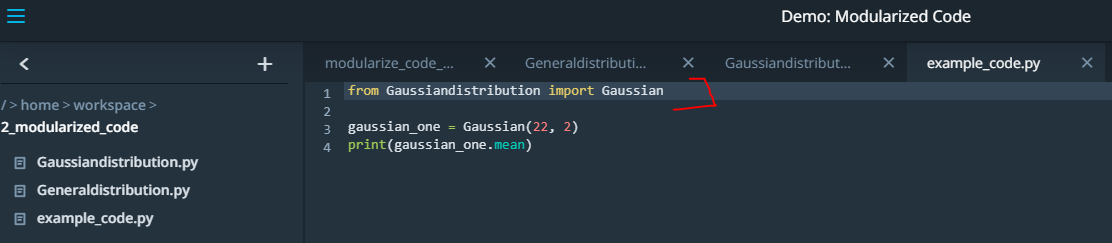
The classroom workspace has one major benefit. You can do whatever you want to the workspace including installing Python packages. And if something goes wrong, you can reset the workspace and start with a clean slate; however, always download your code files or commit your code to github or gitlab before resetting a workspace. Otherwise, you will lose your code!

**Modularized Coding**

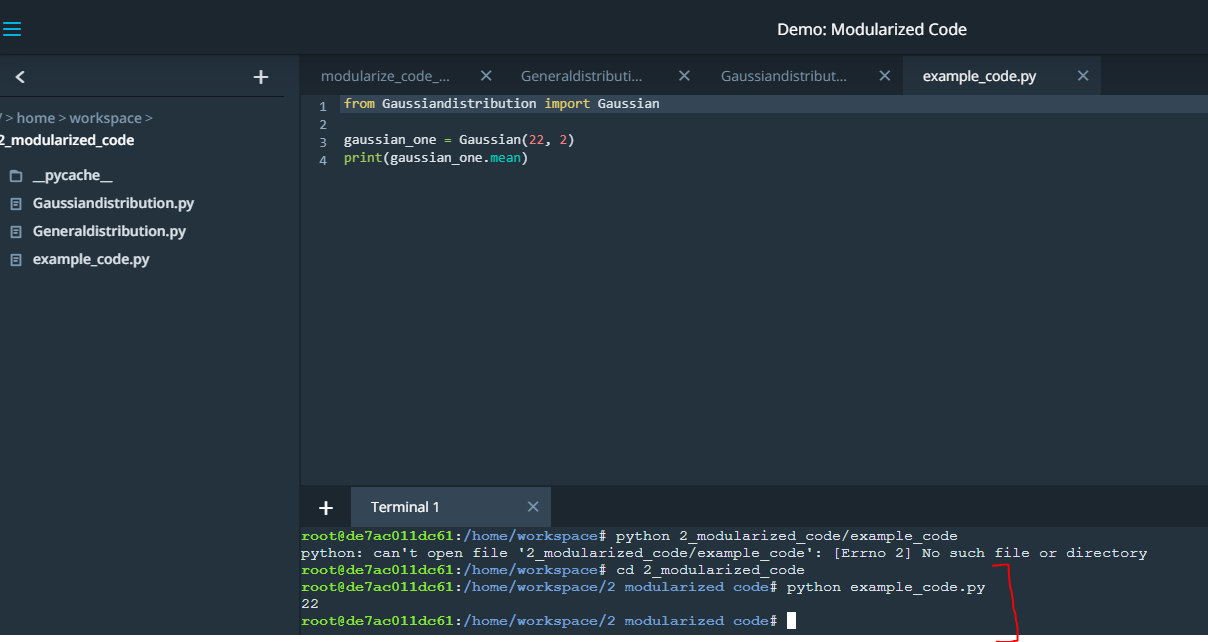








**Now run the code in terminal:**



**Making a Package**

In the previous section, the Distribution and Gaussian code was refactored into individual modules. A Python module is just a Python file containing code.

In this next section, you'll convert the Distributions code into a Python package. A package is a collection of Python modules. Although the previous code might already seem like it was a Python package because it contained multiple files, a Python package also needs an \_\_init\_\_.py file. In this section, you'll learn how to create this \_\_init\_\_.py file and then pip install the package into your local Python installation.

**What is pip?**

Pip is a [**Python package manager**](https://pip.pypa.io/en/stable/) that helps with installing and uninstalling Python packages. You might have used pip to install packages using the command line: pip install numpy. When you execute a command like pip install numpy, pip will download the package from a Python package repository called [**PyPi**](https://pypi.org/). However, for this next exercise, you'll use pip to install a Python package from a local folder on your computer. The last part of the lesson will focus on uploading packages to PyPi so that you can share your package with the world.

You can complete this entire lesson within the classroom using the provided workspaces; however, if you want to develop a package locally on your computer, you should consider setting up a virtual environment. That way if you install your package on your computer, the package won't install into your main Python installation. Before starting the next exercise, the next part of the lesson will discuss what virtual environments are and how to use them.

**Object-Oriented Programming and Python Packages**

A Python package does not need to use object-oriented programming. You could simply have a Python module with a set of functions. However, most if not all of the popular Python packages take advantage of object-oriented programming for a few reasons:

1. Object-oriented programs are relatively easy to expand especially because of inheritance
2. Object-oriented programs obscure functionality from the user. Consider scipy packages. You don't need to know how the actual code works in order to use its classes and methods.

# Virtual Environments

# Python Environments

In the next part of the lesson, you'll be given a workspace where you can upload files into a Python package and pip install the package. If you decide to install your package on your local computer, you'll want to create a virtual environment. A virtual environment is a silo-ed Python installation apart from your main Python installation. That way you can install packages and delete the virtual environment without affecting your main Python installation

Let's talk about two different Python environment managers: conda and venv. You can create virtual environments with either one. Below you'll read about each of these environment managers including some advantages and disadvantages. If you've taken other data science, machine learning or artificial intelligence courses at Udacity, you're probably already familiar with [**conda**](https://conda.io/docs/).

#### Conda

Conda does two things: manages packages and manages environments.

As a package manager, conda makes it easy to install Python packages especially for data science. For instance, typing conda install numpy will install the numpy package.

As an environment manager, conda allows you to create silo-ed Python installations. With an environment manager, you can install packages on your computer without affecting your main Python installation.

The command line code looks something like this:

conda **create** *--name environmentname*

**source** **activate** environmentname

conda **install** numpy

#### Pip and Venv

There are other environmental managers and package managers besides conda. For example, venv is an environment manager that comes pre-installed with Python 3. Pip is a package manager.

Pip can only manage Python packages whereas conda is a language agnostic package manager. In fact, conda was invented because pip could not handle data science packages that depended on libraries outside of Python. If you look at the [**history of conda**](https://jakevdp.github.io/blog/2016/08/25/conda-myths-and-misconceptions/#Myth-#5:-conda-doesn't-work-with-virtualenv,-so-it's-useless-for-my-workflow), you'll find that the software engineers behind conda needed a way to manage data science packages (NumPy, Matplotlib, etc.) that relied on libraries outside of Python.

Conda manages environments AND packages. Pip only manages packages.

To use venv and pip, the commands look something like this:

python3 -m venv environmentname

**source** environmentname/bin/activate

pip install numpy

##### Which to Choose

Whether you choose to create environments with venv or conda will depend on your use case. Conda is very helpful for data science projects, but conda can make generic Python software development a bit more confusing; that's the case for this project.

If you create a conda environment, activate the environment, and then pip install the distributions package, you'll find that the system installs your package [**globally rather than in your local conda environment**](https://github.com/ContinuumIO/anaconda-issues/issues/1429). However, if you create the conda environment and install pip simultaneously, you'll find that pip behaves as expected installing packages into your local environment:

conda **create** *--name environmentname pip*

On the other hand, using pip with venv works as expected. Pip and venv tend to be used for generic software development projects including web development. For this lesson on creating packages, you can use conda or venv if you want to develop locally on your computer and install your package.

The video below shows how to use venv, which is what we recommend for this project.

##### Instructions for venv

Here are instructions about how to set up virtual environments on a macOS, Linux, or Windows machine using the terminal: [**instructions link**](https://packaging.python.org/guides/installing-using-pip-and-virtualenv/).

These are a few notes for understanding the tutorial:

* If you are using Python 2.7.9 or later (including Python 3), the Python installation should already come with the Python package manager called pip. There is no need to install it.
* env is the name of the environment you want to create. You can call env anything you want.
* Python 3 comes with a virtual environment package pre-installed. So instead of typing python3 -m virtualenv env, you can type python3 -m venv env to create a virtual environment.

Once you've activated a virtual environment, you can then use terminal commands to go into the directory where your Python library is stored. And then you can run pip install .. In the next section, you can practice pip installing and/or creating virtual environments in the classroom workspace. You'll see that creating a virtual environment actually creates a new folder containing a Python installation. Deleting this folder will remove the virtual environment.

Note that if you install packages on the workspace and run into issues, you can always reset the workspace; however, you will lose all of your work. So be sure to download any files you want to keep before resetting a workspace.

NEXT

**FOR EXAMPLE:**

**Following the instructions from the previous video, convert the modularized code into a Python package.**

**You can put your code into the 3a\_python\_package folder in the workspace. Inside the 3a\_python\_package folder, you'll need to create a few folders and files:**

**\* a setup.py file, which is required in order to use pip install**

**\* a folder called 'distributions', which is the name of the Python package**

**\* inside the 'distributions' folder, you'll need the Gaussiandistribution.py file, Generaldistribution.py and an \_\_init\_\_.py file.**

**Once everything is set up, open a new terminal window in the workspace by clicking 'NEW TERMINAL'. Then type:**

**cd 3a\_python\_package**

**pip install .**

**If everything is set up correctly, pip will install the distributions package into the workspace. You can then start the python interpreter from the terminal typing:**

**python**

**Then within the Python interpreter, you can use the distributions package:**

**from distributions import Gaussian**

**gaussian\_one = Gaussian(25, 2)**

**gaussian\_one.mean**

**gaussian\_one + gaussian\_one**

**etcetera...In other words, you can import and use the Gaussian class because the distributions package is now officially installed as part of your Python installation.**

**If you get stuck, there's a solution in the 3b\_answer\_python\_package folder.**

**If you want to install the Python package locally to your computer, you might want to set up a virtual environment first. A virtual environment is a siloed Python installation apart from your main Python installation. That way you can easily delete the virtual enviornment without affecting your Python installation.**

**If you want to try using virtual environments in this workspace first, here is how to do it:**

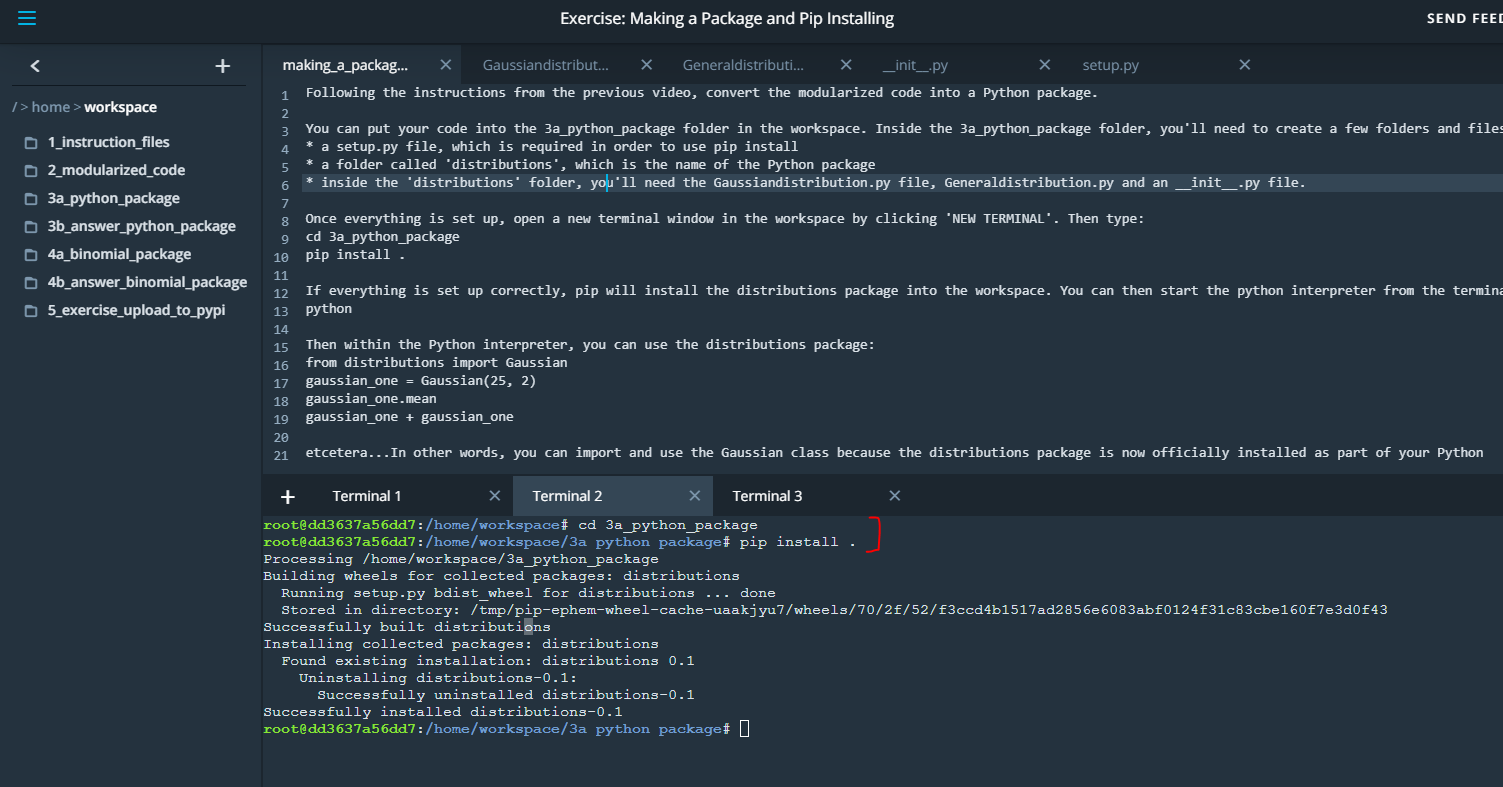
**- There is an issue with the Ubuntu operating system and Python3 where the venv package isn't installed correctly. In the workspace, one way to fix this is by running this command in the workspace terminal: `conda update python` See: https://stackoverflow.com/questions/26215790/venv-doesnt-create-activate-script-python3 Then type `y` when prompted. It might take a couple of minutes for the workspace to update. If you are not using anaconda on your local computer, you can skip this first step.**

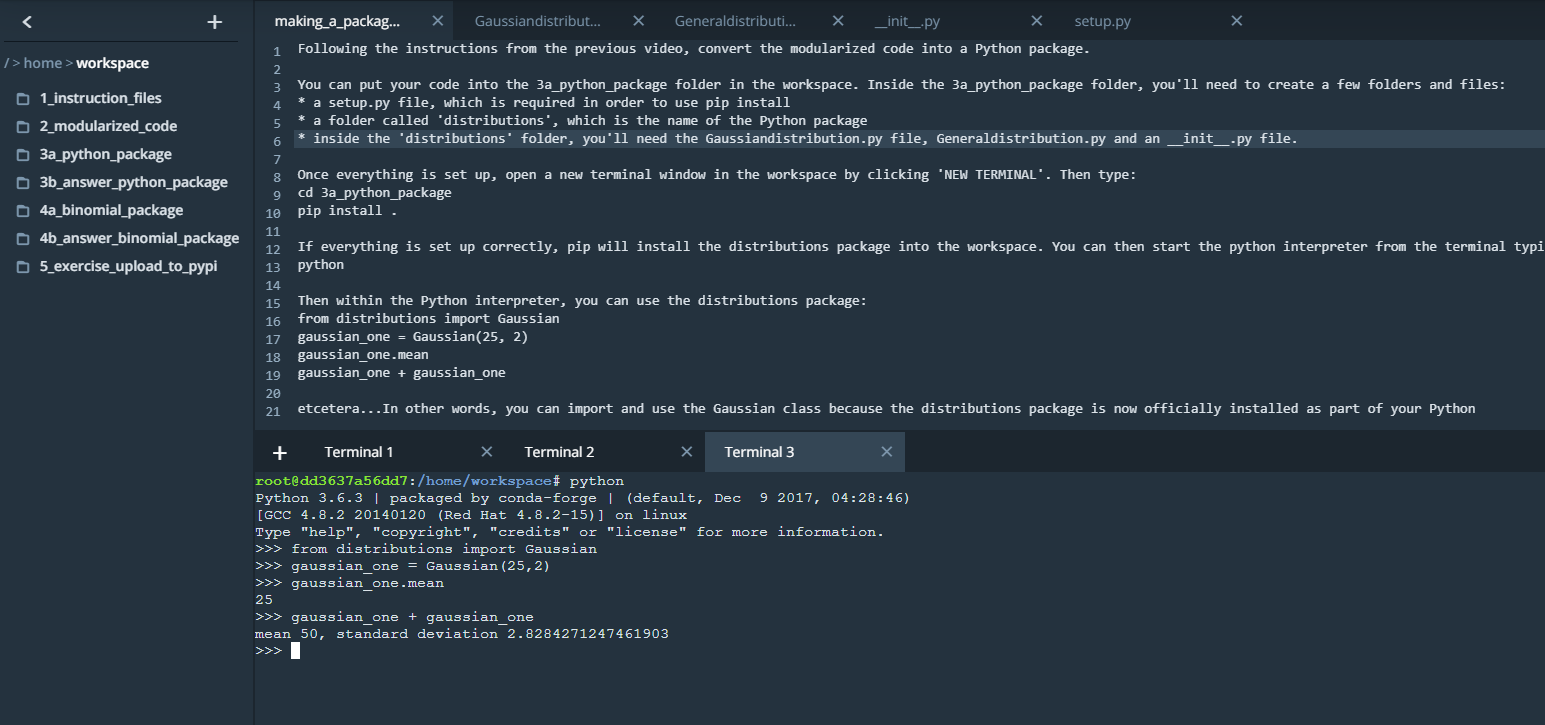
**- now, type this command to create a virtual environment `python -m venv venv\_name` where venv\_name is the name you want to give to your virtual environment. You'll see a new folder appear with the Python installation named venv\_name**

**- In the terminal, type `source venv\_name/bin/activate`. You'll notice that the command line now shows (venv\_name) at the beginning of the line to indicate you are using the venv\_name virtual environment**

**- Now, you can type `pip install python\_package/.` That should install your distributions Python package.**

**- Try using the package in a program to see if everything works!**





**Contributing to a GitHub project**

Here are a few links about how to contribute to a github project:

* [**Beginner's Guide to Contributing to a Github Project**](https://akrabat.com/the-beginners-guide-to-contributing-to-a-github-project/)
* [**Contributing to a Github Project**](https://github.com/MarcDiethelm/contributing/blob/master/README.md)

**Advanced Python OOP Topics**

Here are a few links to more advanced OOP topics that appear in the Scikit-learn package:

* [**Decorators**](https://realpython.com/primer-on-python-decorators/)
* [**Mixins**](https://easyaspython.com/mixins-for-fun-and-profit-cb9962760556)

# Putting Code on PyPi

# 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.

Also, 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.

# Summary of the Terminal Commands Used in the Video

cd binomial\_package\_files

python setup.py sdist

pip install twine

# commands to upload to the pypi test repository

twine upload --repository-url https://test.pypi.org/legacy/ dist/\*

pip install --index-url https://test.pypi.org/simple/ dsnd-probability

# command to upload to the pypi repository

twine upload dist/\*

pip install dsnd-probability

# More PyPi Resources

##### Tutorial on distributing packages

This link has a good tutorial on distributing Python packages including more configuration options for your setup.py file: [**tutorial on distributing packages**](https://packaging.python.org/tutorials/distributing-packages/). You'll notice that the python command to run the setup.py is slightly different with

python3 setup.py sdist bdist\_wheel

This command will still output a folder called dist. The difference is that you will get both a .tar.gz file and a .whl file. The .tar.gz file is called a source archive whereas the .whl file is a built distribution. The .whl file is a newer type of installation file for Python packages. When you pip install a package, pip will first look for a whl file (wheel file) and if there isn't one, will then look for the tar.gz file.

A tar.gz file, ie an sdist, contains the files needed to [**compile**](https://en.wikipedia.org/wiki/Compiler) and install a Python package. A whl file, ie a built distribution, only needs to be copied to the proper place for installation. Behind the scenes, pip installing a whl file has fewer steps than a tar.gz file.

Other than this command, the rest of the steps for uploading to PyPi are the same.

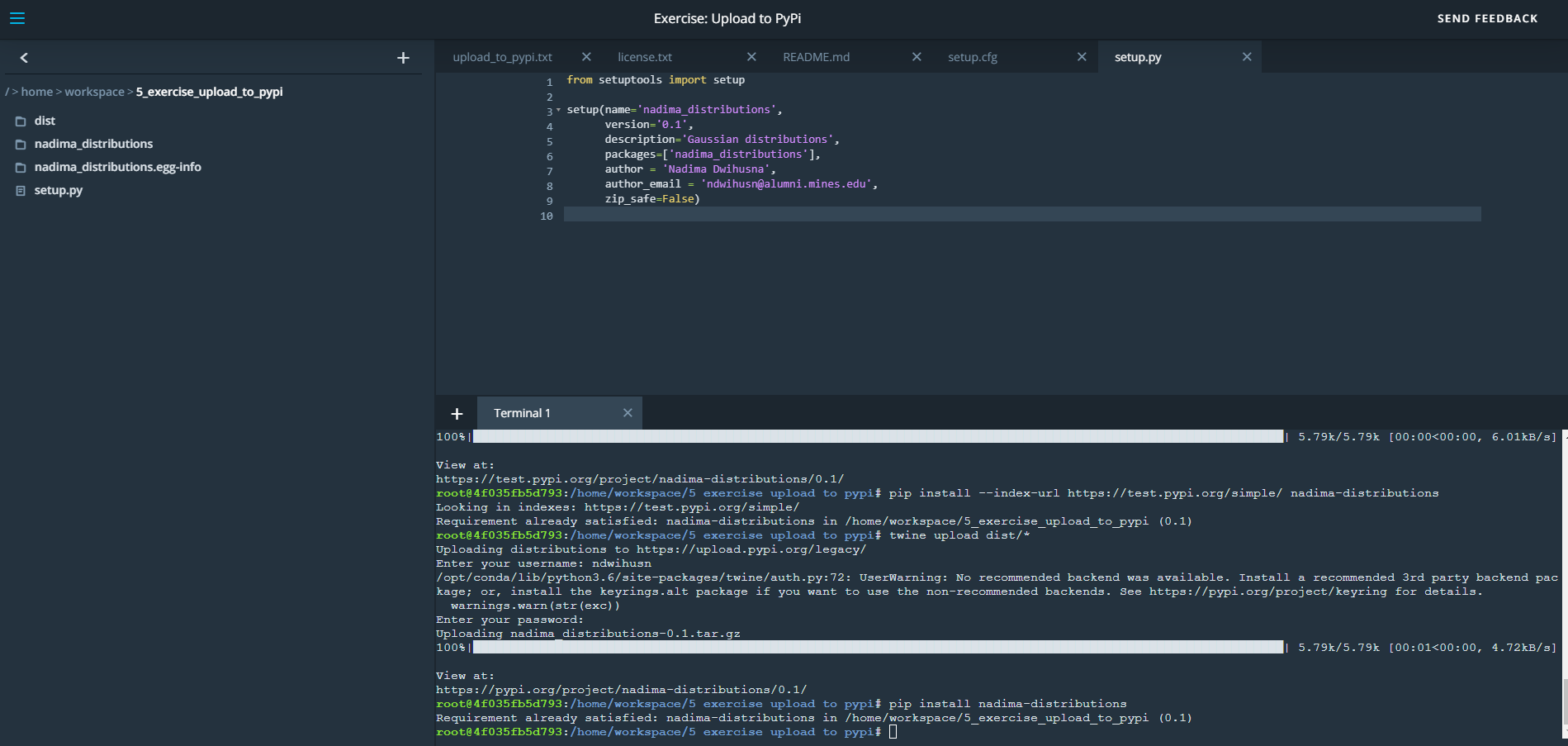
##### Other Links

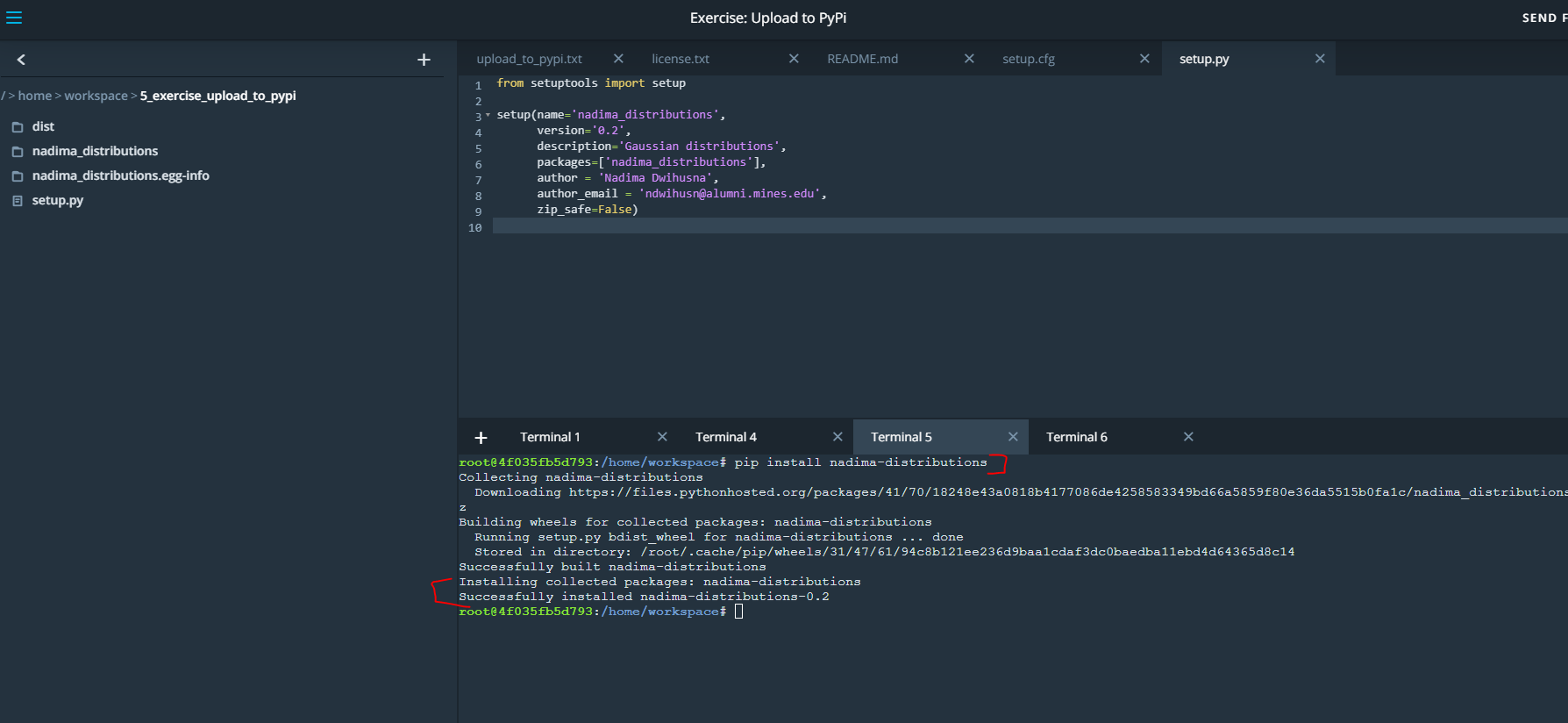
If you'd like to learn more about PyPi, here are a couple of resources:

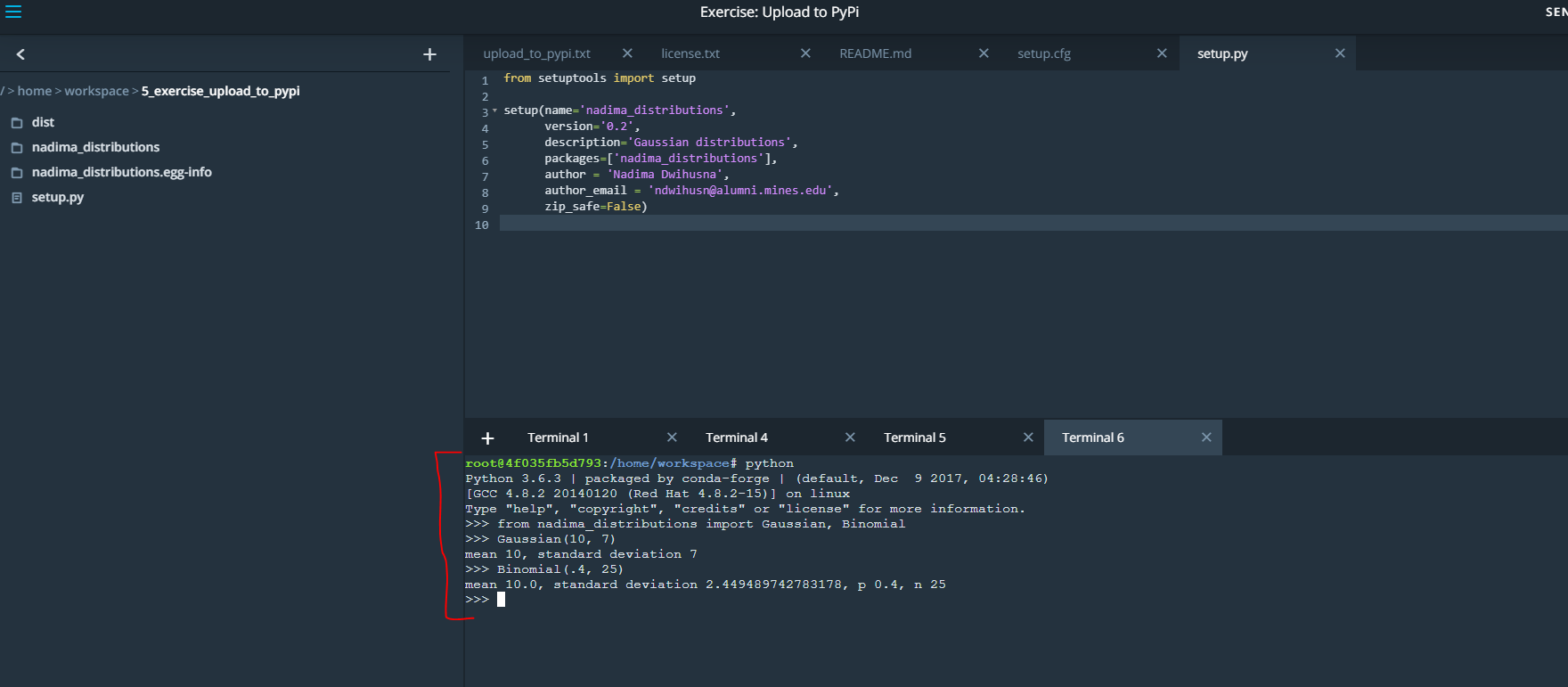
* [**Overview of PyPi**](https://docs.python.org/3/distutils/packageindex.html)
* [**MIT License**](https://opensource.org/licenses/MIT)

**TRY IT OUT!! Usind the terminal commands to install to PiPY.**

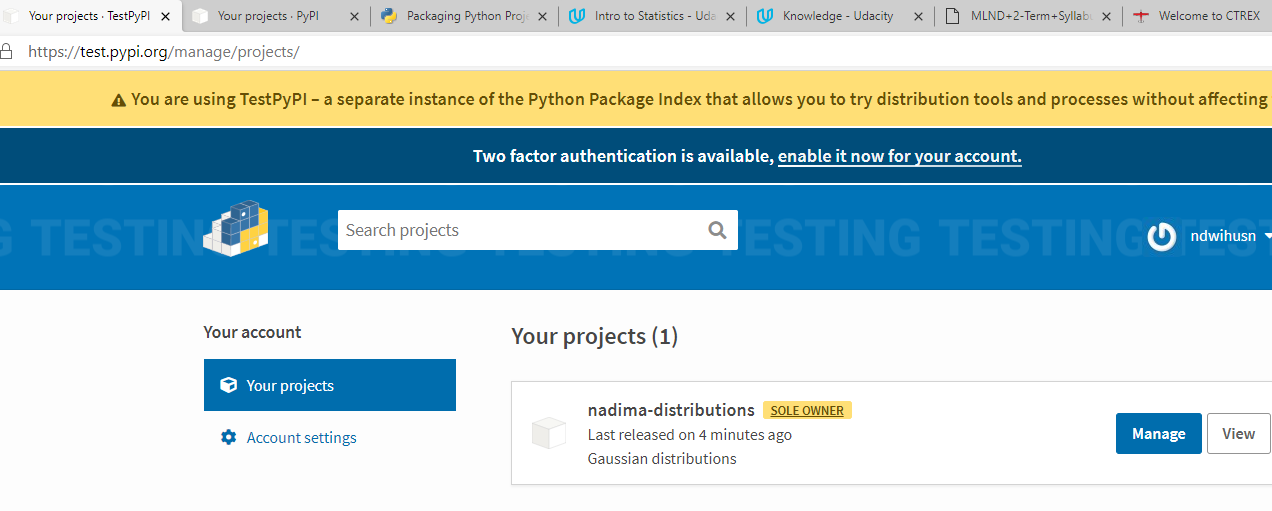
**My user name is ndwihusn. Password is same as email password**

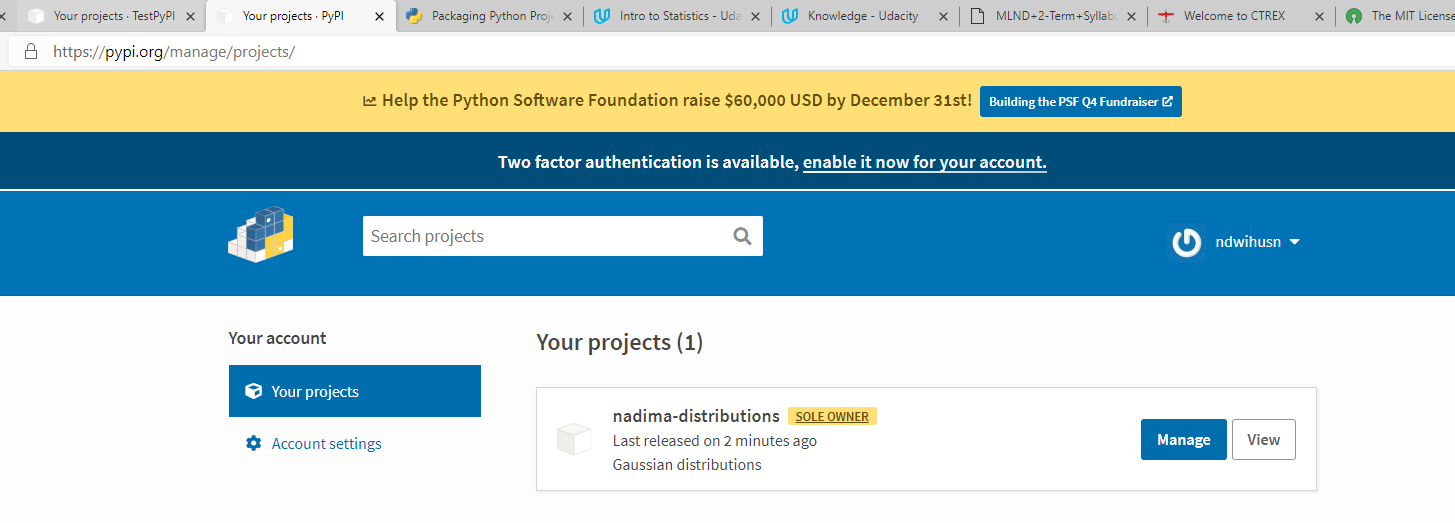






**It works.. you can also see you package uploaded in PyPi and test Pypi**





# Portfolio Exercise

In the next lesson, you'll find instructions for a portfolio exercise involving object-oriented programming and Python packages.

A portfolio exercise is an assignment meant to show off your skills and expand your professional portfolio. They are an excellent way to demonstrate to potential employers your knowledge and creativity. Note that these exercises are not reviewed and are thus not official graduation requirements.