Getting started with Python environments (using Conda)

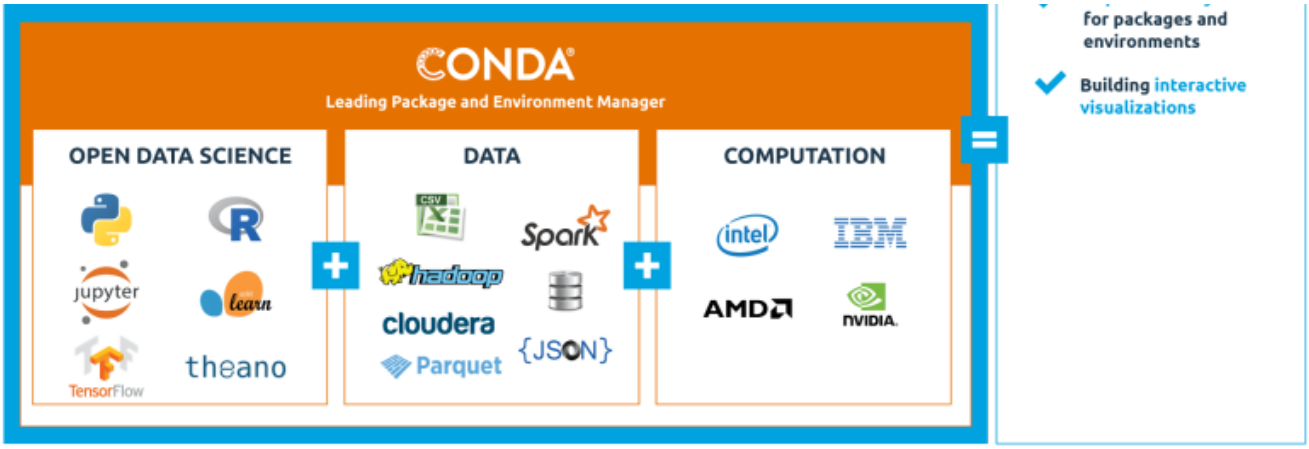
Robert Sandor Follow

Oct 21, 2018 · 6 min read

Whether you want one or have no idea what it is, you'll have to deal with environments in Python eventually. If you're a newbie to Python like myself or a newbie to generally setting up your workspace for any programming language, then you may have dealt with the pain of setting up your environment for the first time.

Between the lack of knowledge of what you're doing and why it's necessary, the many different online guides that may not have the exact same instructions and numerous Stack Overflow posts that help with very specific situations, I found it confusing to understand what exactly environments were, when they were needed, and how to set up my own. This guide is to help build an intuition for environments and provide some examples of dealing with them, particularly using **conda**, the package manager for **Anaconda**.

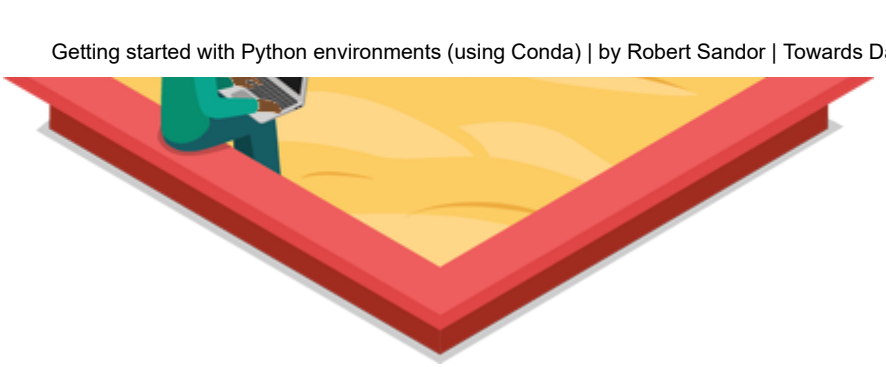
Your Code Is The Product Of Your Environment



Python, like many other programming languages, has different versions. And sometimes when we create software, the software needs to run on a specific version of the language because our software expects a certain behavior that is present in older versions but changes in newer versions. Likewise, we may need to use specific versions of the libraries for similar reasons. But we may have many projects on our computer, perhaps a **Flask** app that runs on version 0.11 (the first one you made!) and **Python 2.7** and even a more modern **Flask** app that runs on version 0.12 and **Python 3.4**. If I try running both at once on **Python 2** or **Python 3**, one of them may break because some of the code that runs on **Python 2** doesn't run on **Python 3** or vice versa. This is where virtual environments become useful.

Virtual environments keep these dependencies in separate "sandboxes" so you can switch between both applications easily and get them running. For those more familiar with programming, virtual environments are analogous to **Docker** containers. Additionally, package managers for other languages, like JavaScript's **NPM** (**Node Package Manager**), take care of most of these details for you, but you'll have to get your hands dirty in **Python** and deal with the environments yourself.

Create The Sandbox You Play In



There are multiple ways of creating an environment, including using virtualenv, venv (built in to the Python 3 standard library), and conda,
the package manager associated with **Anaconda**. There are some arguments as to why you should choose conda over **virtualenv** as outlined in Myth #5 in this blog post, but I'll just focus on how to use **conda** in this guide since that's a popular tool for data science, which is what I'm focused on right now.
This guide will presume that you already have **Anaconda** or **miniconda** installed; all the instructions will also be on the bash command line.
For reference, I run my commands on the Terminal on **Mac OS X**.

To quickly create an environment using **conda**, you can type in the command:

```
conda create --name your_env_name python=3.7 -y
```

In this command, the '**python=3.7**' portion specifies which version of python I want to set up the environment in; you can change the version to whatever suits your needs. In other snippets you see online, you may see '**-n**' instead of '**— name**'; they mean exactly the same thing. The '**-y**' flag essentially tells the command line to say 'yes' to all of the prompts that follow; it isn't strictly necessary but it does save you a little hassle.

```
conda create --name your_env_name python=3.7 scipy=0.15.0 astroid
babel
```

The '**conda create**' command will effectively load all of the packages at once, which is preferable to loading them in 1 at a time as that can lead to dependency conflicts. So you could somewhat manually add all of the packages you need, but that may be tedious if you have many packages;

besides that, there would be a lot of typing involved on the command line and a slip of the finger may cause you to reenter the command. Even worse, the command may not remain in your shell history and if you wanted to recreate the exact same environment in the future, it would be very tedious if not difficult.

If you didn't want to create the environment from the command line for those reasons or others, you could create a **YAML** (YAML Ain't Markup Language) file, which acts like a configuration file.
Your **YAML** file may look something like this:

```
name: your_env_name
channels:
 - defaults
dependencies:
 - ca-certificates=2018.03.07=0
prefix: /Users/your_username/anaconda3/envs/your_env_name
```

If that file were called '*environment.yml*', then I could create the environment using the command below:

```
conda env create -f environment.yml
```

The '**-f**' flag stands for file and the filename of the **YAML** file should immediately follow the '**-f**' flag.

Now that's great if you can easily create a **YAML** file and you know all of the packages you need. But what if you had an existing environment that you would like to duplicate? Perhaps you'd like to duplicate the application onto another server and want the exact same setup for consistency. If that's the case, then you can run the command below.

```
conda env export > my_environment.yml
```

The greater than symbol, '>', indicates that the output is writing to a file called '*my_environment.yml*'. If there were any contents in '*my_environment.yml*' before this command, they would be overwritten.

As a note, for **conda** you need a **YAML** file; if you decide to use **virtualenv**, a txt file would also suffice for everything done here, but **conda** specifically needs a **YAML** file.

Own Your Environment

So now that you have an environment created and assuming that you're using conda, let's quickly verify that it exists using the command:

```
conda info --envs
```

This command should display the current environments, which may look something like this:

```
practice /Users/your_username/anaconda3/envs/practice
base /Users/your_username/anaconda3
your_env_name /Users/your_username/anaconda3/envs/your_env_name
```

After confirming that you created the environment, you can now actually use it. We can accomplish this by typing (assuming your environment is underneath your base):

```
conda activate your_env_name
```

At this point, your terminal prompt should look something like this:

```
(your_env_name) Your_Machine:your_directory username$
```

If that command doesn't result in a similar output for whatever reason,
you can specify the full path by typing in something similar to the command below:

```
conda activate /Users/your_username/anaconda3/envs/your_env_name
```

If, like me, you want to know exactly what is happening when you type in the word 'activate', it runs a bash script that exists within a subdirectory of the environment.
In my case, the filepath to the script looked something like:

```
/Users/my_username/anaconda3/envs/my_env_name/lib/python3.6/venv/scri
pts/common/activate
```

To stop using the environment, type in

```
conda deactivate
```

In a similar fashion to the activate command, the deactivate command runs a function from the activate bash script.

If you would like to update the environment, type in:

```
conda env update -f environment.yml -n your_env_name
```

If you would like to get rid of the entire environment, merely type in:

```
conda remove --name your_env_name --all
```

The '**— all**' flag is to remove all packages from the environment and is necessary to completely clean the environment.

Conclusion

As a quick summary, this guide covered how to create your virtual environment by specifying packages on the command line as well as from within a **YAML** file, how to get into and out of the virtual environment, how to update it, and how to remove it once you no longer need it.

At this point, you should know enough to independently configure your environment as you see fit and have enough context for why you should create a virtual environment. While this guide didn't go in depth on the differences between **virtualenv**, **venv**, and **conda** virtual environments, I've provided a few links below to get you started.

Please let me know if you have any questions or suggestions on how to make this better.

. . .

https://jakevdp.github.io/blog/2016/08/25/conda-myths-and-misconceptions/
https://realpython.com/python-virtual-environments-a-primer/
https://medium.freecodecamp.org/why-you-need-python-environments-and-how-to-manage-them-with-conda-85f155f4353c

Python Data Science