Published in Towards Data Science

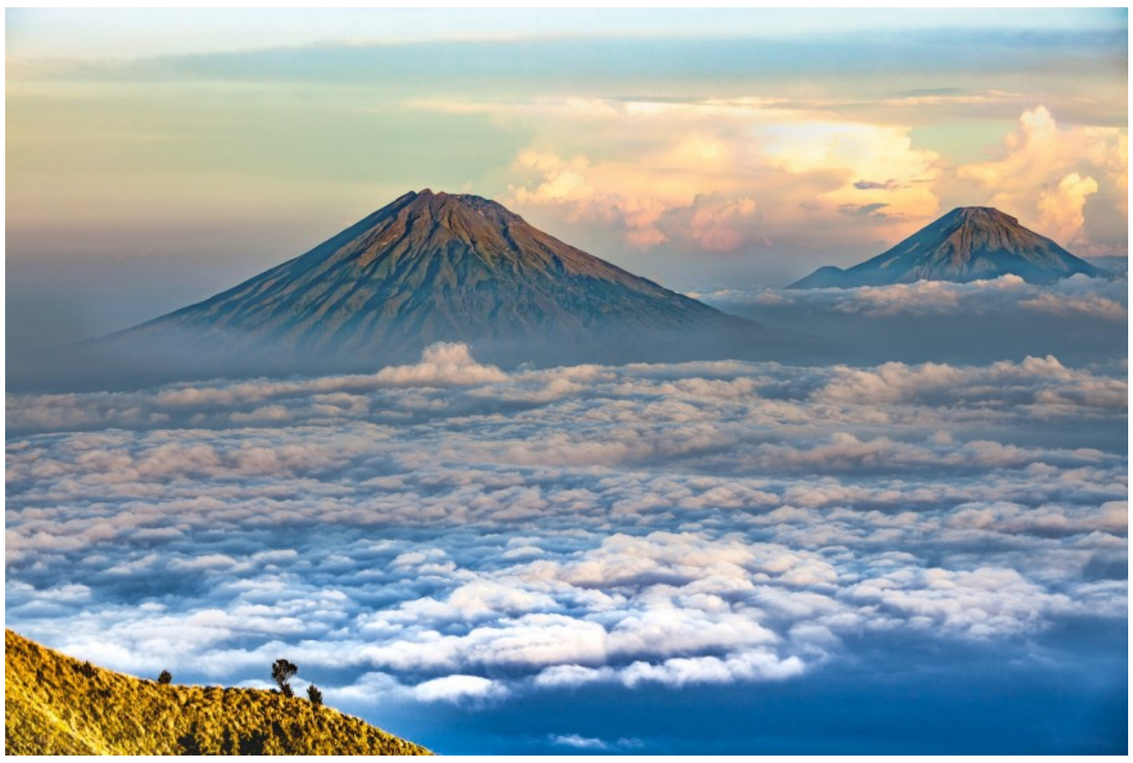
Jeff Hale

Feb 4, 2019 · 6 min read ★ · Listen

Learn Enough Python to be Useful: argparse

How to Get Command Line Arguments Into Your Scripts

If you plan to be a software developer with Python, you'll want to be able to use `argparse` for your scripting needs. If you're a data scientist, you'll likely find yourself needing to port your code from a Jupyter Notebook to a reproducible script. For many newer data scientists this can be a step from a comfortable, happy place into scary land. This guide is designed to make the leap less scary.



Like a mountain through the clouds, argparse brings arguments from the command line. 😊

`argparse` is the "recommended command-line parsing module in the Python standard library." It's what you use to get command line arguments into your program.

I couldn't find a good intro guide for `argparse` when I needed one, so I wrote this article. Enjoy!

Jupyter is great

Beyond the Jupyter Notebook

The first time I saw `argparse` in a Python script for a side project I thought, "What is this voodo magic?" And quickly moved the code into a Jupyter Notebook. This move turned out to be suboptimal. 😦

I wanted to be able to run a script rather than have to step through a Jupyter Notebook. A script with `argparse` would have been much simpler to use and much less work. Unfortunately, I was in a hurry and didn't find the docs easy to grasp.

Since then, I've come to understand and enjoy `argparse`. It's indispensable.

Here's what you need to know.

Why Use argparse?

`argparse` — parse the arguments.

Using `argparse` is how you let the user of your program provide values for variables at runtime. It's a means of communication between the writer of a program and the user. That user might be your future self. 😃

Using `argparse` means the doesn't need to go into the code and make changes to the script. Giving the user the ability to enter command line arguments provides flexibility.

Python is great, too!

Example

Say you have a directory with videos in it and want to turn those videos into images using the OpenCV library. You could use `argparse` so the user can enter input and output directories. Here's what the `argparse` section of your *videos.py* file looks like:

```
# videos.py
import argparse

parser = argparse.ArgumentParser(description='Videos to images')
parser.add_argument('indir', type=str, help='Input dir for videos')
parser.add_argument('outdir', type=str, help='Output dir for image')

args = parser.parse_args()

print(args.indir)
```

This file imports the `argparse` library. Then it makes a parser object with a description. Then the variable *indir* is created using `parser.add_argument()`. The type of the variable is set to string and a help message is provided. Then the same is done for *outdir*. Next the *args* variable is set to the values of the parsed arguments.

Now the following command can be run from the command line:

```
python videos.py /videos /images
```

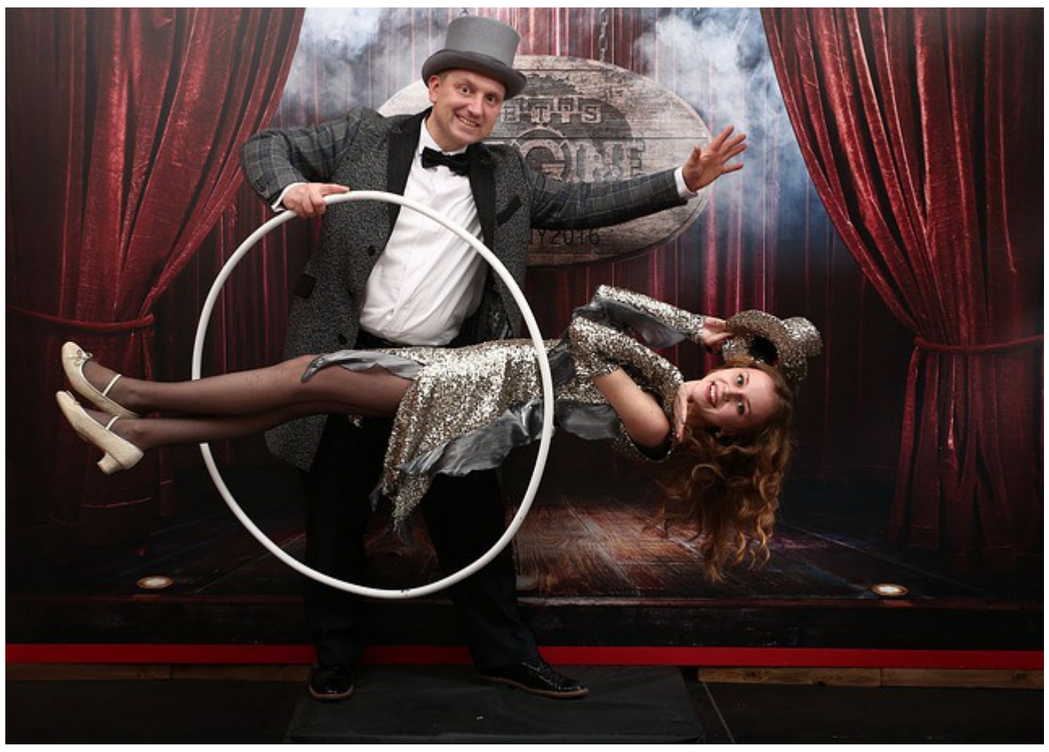
Note that quotes do not need to be placed around the values `/videos` and `/images` when you pass them.

`"/videos"` becomes the value for `args.indir` and `"/images"` becomes the value for `args.outdir`.

The output printed to the terminal is `/videos`.

We just showed that you can use the `args.indir` variable anywhere in your program. How cool is that?

You've now seen the magic of argparse!



How do they do that?

What Else Should You Know About argparse?

Positional Arguments

`parser.add_argument('indir', type=str, help='Input dir for videos')` created a *positional argument*. For positional arguments to a Python function, the order matters. The first value passed from the command line becomes the first positional argument. The second value passed becomes the second positional argument.

What happens if you exclude these positional arguments and try to run `python videos.py` ?

You'll get an error: `videos.py: error: the following arguments are required: indir, outdir` . Positional arguments are always required to be passed in the command to run the script.

Optional Arguments

What happens if you run `python videos.py --help` ?

You get the helpful information we put into our script to tell you what you need to do.

```
usage: videos.py [-h] indir outdir

Videos to images

positional arguments:
  indir       Input dir for videos
  outdir      Output dir for image

optional arguments:
  -h, --help  show this help message and exit
```

Output of `python videos.py --help`

Excellent! `help` is an example of an *optional argument*. Note that `--help` is the only optional argument you get for free, but you can make more.

Optional arguments are created just like positional arguments except that they have a `'--'` double dash at the start of their name (or a `'-'` single dash and one additional character for the short version). For example, you can create an optional argument with `parser.add_argument('-m', '--my_optional')` .

The following larger example shows how to create and reference an optional argument. Note that we specify the type `int` for an integer in this example. You could also specify other valid Python variable types.

```
# my_example.py
import argparse

parser = argparse.ArgumentParser(description='My example explanation')

parser.add_argument(
    '--my_optional',
    default=2,
    help='provide an integer (default: 2)'
)
```

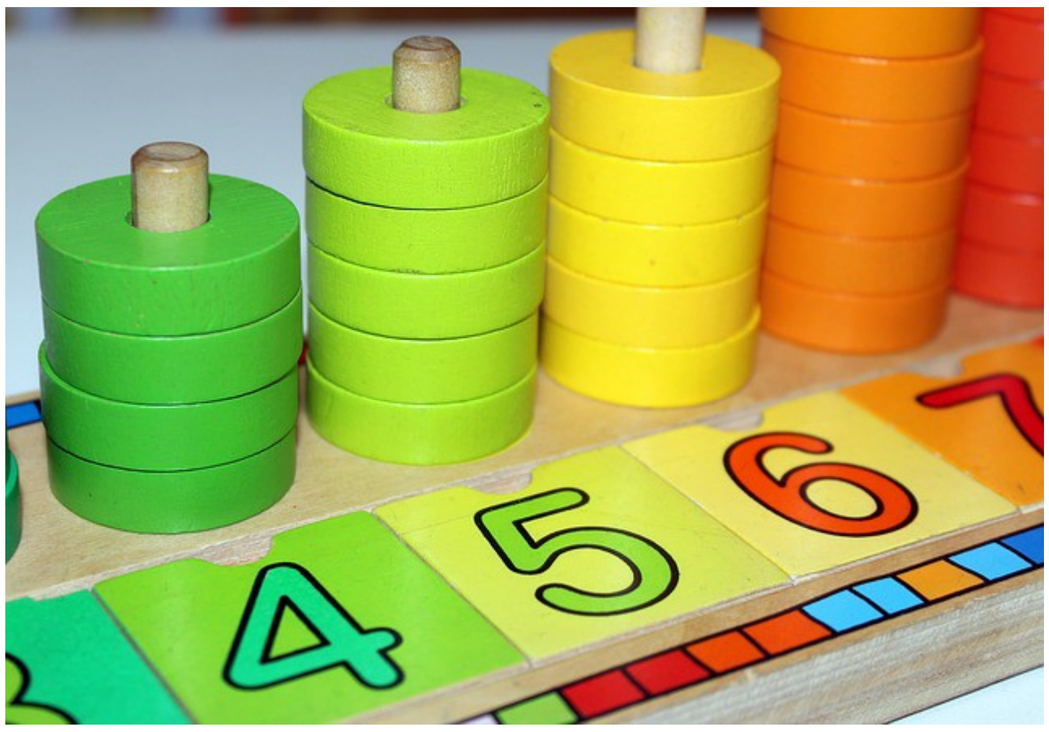
```
my_namespace = parser.parse_args()

print(my_namespace.my_optional)
```

Note that the argument specified with `'--my_optional'` becomes this namespaced variable without the dashes: `'my_namespace.my_optional'`.

Also note that the optional argument can have a default value. Here we specify a default of 2. Running `python my_example.py` outputs *2*.

The optional argument value can be set at run time from the command line like this: `python my_example.py--my_optional=3`. The program then outputs *3*.



Integers

You can do even more with `argparse`. For example, you can have arguments gathered into lists with `nargs='*'`. You can also check for ranges of values with `choices`. See the `argparse` docs for all you can do.

When Else Might I Use argparse?

You can also use `argparse` with programs running in Docker containers. If you want to pass command line arguments to your scripts when building your image you can do so with RUN. If you want to pass arguments to your script at run time you can do so with CMD or ENTRYPOINT. Learn more about Dockerfiles in my series on Docker:

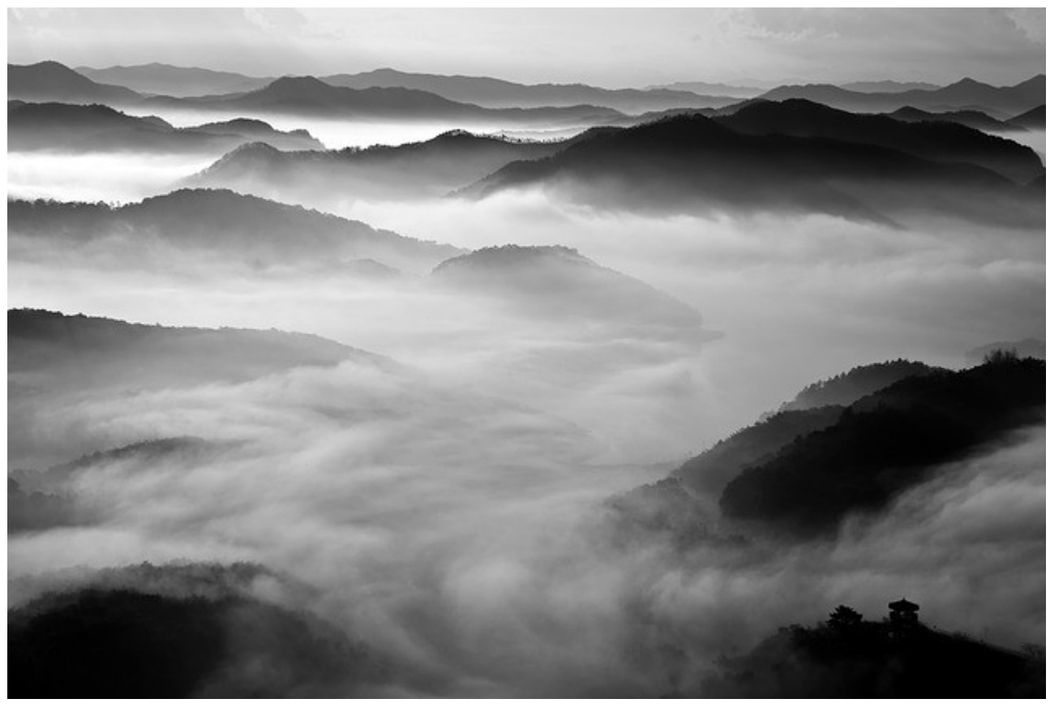
Learn Enough Docker to be Useful
Part 3: A Dozen Dandy Dockerfile Instructions
towardsdatascience.com

Wrap

Now you've seen the basics of `argparse`. You've seen how to get positional and optional arguments into your programs from the command line. You've also seen how to set default optional arguments. If you want to go deeper, check out the official docs.

Update Mar. 6, 2019 I should mention that there are a number of packages available to add command line arguments to your program. Readers have suggested several in the comments, the most popular of which I've linked to here:

- click
- fire
- docopt



More mountains through the mist

Here are a few more suggestions to help you step out of the Jupyter Notebook.

Environment variables are useful variables that get set outside a program. Here's a nice, clear intro. This article from DataCamp blog focuses on the PATH variable.

You can convert repos with Jupyter notebooks into Docker Images with Repo2Docker. Will Koehrsen wrote a good guide on the tool here.

I plan to write more articles about interacting with the file system and scripting. Follow me to make sure you don't miss them! 😃

I hope you found this intro useful. If you did, share it on your favorite forums and social media. Data scientists and programmers who don't know `argparse` will thank you!

I write about data science, cloud computing, and other tech stuff. Follow me and read more here.

Thanks for reading!👏