Introduction to Python Development

CHAPTER 10.3
Building the CLI: Handling Arguments and Flags

Our project has a few different discrete parts and we're going to build them in isolation before eventually connecting them together. In this lesson, we'll go about building the CLI that handles the user interaction and gets the variables that we need.

Documentation For This Video
The argparse package (https://docs.python.org/3/library/argparse.html)
The argparse.ArgumentParser class
(https://docs.python.org/3/library/argparse.html#argparse.ArgumentParser)
The argparse.Action class (https://docs.python.org/3/library/argparse.html#argparse.Action)

Creating Our pgbackup Package
Up to this point pgbackup exists as a project, but not a package. We'll be creating quite a few different modules and want them to exist within the pgbackup package. To create a package we need to have a directory with our package name that includes a __init__.py file. Let's create that now within src:

```
(pgbackup) $ mkdir src/pgbackup
(pgbackup) $ touch src/pgbackup/__init__.py
```
Now we can create the rest of our modules within the src/pgbackup directory.

The argparse Package
When it comes to creating CLIs in Python, the standard library comes with a great package: the argparse package. Let's create a new Python file that will hold onto our CLI related logic at src/pgbackup/cli.py. In this file, we're going to write a function that will configure and return a parse that we can use when we finally run our tool.

Here's our initial implementation:

src/pgbackup/cli.py

```
from argparse import ArgumentParser

def create_parser():
    parser = ArgumentParser(description="""
    Back up PostgreSQL databases locally or to AWS S3.
    """)
    return parser
```
The ArgumentParser class is going to be the backbone of our user interface. It makes it easy for us to define the arguments and options available from our CLI.

Handling Arguments and Flags
If we remember back to how we stated our tool would work in the documentation (README.md), we can see that we need a few things:

```
$ pgbackup postgres://bob@example.com:5432/db_one --driver s3 backups
```
or

```
$ pgbackup postgres://bob@example.com:5432/db_one --driver local /var/local/db_one/backups
```
We need:

A positional argument for the url of the database we're connecting to.
A 2 part flag that gives us the driver name, and a context specific value of either the S3 bucket name or the local path to back up the database.
The positional argument will be easy to define, but the optional argument is actually more complicated than we can handle with an ArgumentParser out of the box, but the argparse.Action class will allow us to create a custom flag handler class. Here's what our implementation will look like to handle the arguments and flags:

~/projects/pgbackup/src/pgbackup/cli.py

```
from argparse import Action, ArgumentParser

class DriverAction(Action):
    def __call__(self, parser, namespace, values, option_string=None):
        driver, destination = values
        namespace.driver = driver.lower()
        namespace.destination = destination

def create_parser():
    parser = ArgumentParser(description="""
    Back up PostgreSQL databases locally or to AWS S3.
    """)
    parser.add_argument("url", help="URL of database to backup")
    parser.add_argument("--driver",
            help="how & where to store backup",
            nargs=2,
            action=DriverAction,
            required=True)
    return parser
```

By defining a custom "action" we're able to specify that the --driver flag will actually populate 2 attributes on our arguments namespace when we're finished parsing the arguments.

Manually Testing the Parser
As we've done many times before, we're going to pull our code into the REPL so that we can test it out to see if it works. Ideally, we would write some automated tests that could verify that the function works as expected, but that's a little beyond what we're trying to learn right now.

Let's see our parser in action:

```
(pgbackup) $ $ python -i src/pgbackup/cli.py
>>> parser = create_parser()
>>> args = parser.parse_args(['https://some_url', '--driver', 's3', 'bucket_name'])
>>> args.url
'https://some_url'
>>> args.driver
's3'
>>> args.destination
'bucket_name'
>>> parser.parse_args()
usage: cli.py [-h] --driver DRIVER DRIVER url
cli.py: error: the following arguments are required: url, --driver
```

The create_parser function will return our pre-configured parser and then to use it we'll call the parse_args method. If we don't pass anything to the parse_args method then it will automatically parse the arguments in sys.argv. To the parser out with expected arguments though, we can pass in a list of the the tokens that would come from stdin.

By using add_argument with a --driver we were specifying that this is a flag. The url argument is positional because we didn't add dashes. With this implemented we're ready to move onto another section of the tool.

Before we conclude, let's not forget to commit:

```
(pgbackup) $ git add --all .
(pgbackup) $ git commit -m 'Create pgbackup package and cli module'
```