What does argparse do?

argparse is a Python module that allows a user to pass "arguments" into a program when they run it from the command line. For example, imagine a simple script that takes an input file. Rather than specify the filepath deterministically in the program or prompt the user to enter it at runtime, we can set up something like this:

```
$ python myscript.py --input_file myfile.txt
```

This way, the user specifies the input filepath at runtime. This means that the program can be written to use different input files without the user having to change the actual code.

argparse is also useful as a documentation tool because it provides a simple interface for providing the user with help messages for running the script.

How do we use argparse to take in arguments?

Let's consider the example above, where we needed to import the name of an input file from a command line argument. To do this, we import the module first:

```
import argparse
```

Next, we can call the relevant function in the module and define the arguments that we want the program to be able to support:

```
parser = argparse.ArgumentParser()
parser.add_argument('-i', '--input_file', help="name of the input file")
```

Here we've provided a **help** message to help the user understand how to use the input_file argument. **More on this later**. We've also included a **shortcut version** of the "input_file" argument which can be invoked by adding just "-i" to the command line at runtime. In terms of functionality, both "-i" and "--input_file" will do the same thing.

Finally, we have to "process" the arguments so they can be used in the program:

```
args = parser.parse_args()
```

This **parses** the arguments and stores them so that they can be called later in the code. When we need to call the arguments, they are stored as **attributes** of the parser. In this case, they'll be stored as **attributes** of the args variable we just created for the parser above. Here, we take the parsed argument, add it to a variable name, and print it:

```
input_file = args.input_file
print(input_file)
```

Then, if we run the program from the command line with our argument defined, we'll get the printed output:

```
$ python argparse_example.py -i hello_there
hello_there
```

Of course, for the actual functionality of consuming an input file, we would use the imported argument differently. Here is some similar syntax from my API call program where I use **argparse** to pull the name of my input file:

```
parser = argparse.ArgumentParser()
parser.add_argument('-i', '--input_file', help="path to a newline-delimited list of years",
    required=True)
args = parser.parse_args()
input_file = args.input_file
with open (input_file, "r") as year_file:
    year_list = year_file.read().splitlines()
return year_list
```

Here, I've also specified the "required=true" option because my script cannot run without an input file. If we try to run without specifying -i, **argparse** will provide a helpful error message!

```
$ python apicall.py
usage: apicall.py [-h] -i INPUT_FILE
apicall.py: error: argument -i/--input_file is required
```

Some **argparse** options that you might want to consider using:
help= (Stores a help message that describes what the argument does)
required=True (Requires this argument for the program to run)
dest= (Specify the name of the attribute under which this argument value will be stored)
nargs= (Specifies a number of required values for this argument. Required to support more than 1 value. * is a wildcard allowing any number of values.)
type= (Specifies an argument type such as int, str, etc.)

default= (Allows the user to define a default value for the argument if the user doesn't pass anything in)
choices=['option1', 'option2'] (Defines a limited set of valid values for this argument, "option1" and "option2" in this case.)
action="store_true" (Stores the value of the argument as the boolean value "True" rather than the argument text itself)
action="store_const" (Stores the value under a specified constant value rather than the argument text itself)
action="append" (Allows the user to specify the argument multiple times. The individual values are stored as a list)

How can I use argparse to display help messages for a user?

argparse includes built-in functionality to help you display program usage instructions to your user. If **argparse** is imported into a program, you can run it from the command line with the "--help" or "-h" arguments:

```
$ python argparse_example.py --help
$ python argparse_example.py -h
```

Running either of these will display the help text that was included when defining the program's arguments. For example, if we run them for our argparse_example program from earlier, we'll see the following:

```
$ python argparse_example.py -h
usage: argparse_example.py [-h] [-i INPUT_FILE]

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT_FILE, --input_file INPUT_FILE
                        name of the input file
```

This shows us all the possible arguments that can be given to the script along with their shortcuts.

Challenge: If you haven't done so already, rework an existing script that requires user input to use argparse instead. Remember to anticipate which user input is required and which is optional. Also, keep in mind that you might have to validate the input passed with an argument!