Options

Jamie Saxon

Introduction to Programming for Public Policy

October 16, 2016

Arguments

What Are Arguments

We have already used arguments to:

- perform functions on different inputs.
- modify the behavior of programs (e.g., grep -i or sort -r).

Now let's modify the behavior of our scripts!

The idea is that we want the same code to be able to do many things, without rewriting it.

The argparse module

Unapproachable **documentation** but a good **tutorial**.

- Try running the simplest example, with '-h':

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("number", help="A number!")
args = parser.parse_args()

print(args)
```

The argparse module

ex/options_intro.py

Unapproachable **documentation** but a good **tutorial**.

- Try running the simplest example, with '-h':

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("number", help="A number!")
args = parser.parse_args()

print(args)
```

Add a requred argument:

- `parser.add_argument("number", help="A number!")`

The argparse module

Unapproachable **documentation** but a good **tutorial**.

- Try running the simplest example, with '-h':

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("number", help="A number!")
args = parser.parse_args()

print(args)
```

Add a requred argument:

- `parser.add_argument("number", help="A number!")`

Add an optional argument (dashes, or 'required'):

- `parser.add_argument("--extras", default = "yay!")`

Controling the Types

Please follow along with these.

Controling the Types

Please follow along with these.

By default, all arguments are strings; 'cast' them on the fly with 'type.'

- `parser.add_argument("number", type = float, help="A number!")`

`type` takes a function; lambda functions, str.lower, etc. work too.

Controling the Types

Please follow along with these.

By default, all arguments are strings; 'cast' them on the fly with 'type.'

- `parser.add_argument("number", type = float, help="A number!")`

`type` takes a function; lambda functions, str.lower, etc. work too.

Specifying default values is easy:

- `parser.add_argument("number", type = float, default = 3)`

Controling the Types

Please follow along with these.

By default, all arguments are strings; 'cast' them on the fly with 'type.'

- `parser.add_argument("number", type = float, help="A number!")`

`type` takes a function; lambda functions, str.lower, etc. work too.

Specifying default values is easy:

- `parser.add_argument("number", type = float, default = 3)`

Or an `action="store_true"` (default is the opposite):

- `parser.add_argument("--store", action="store_true")`

Using the Arguments

- After running `args = parser.parse_args()`, the arguments become accessible as variables, via the long-form versions of the argument names, `args.var_name`.
- We can then pass these variables into a function or class, to run our script with varying behavior.

Let's experiment with `options_example.py`.