22 – Command line arguments

Bálint Aradi

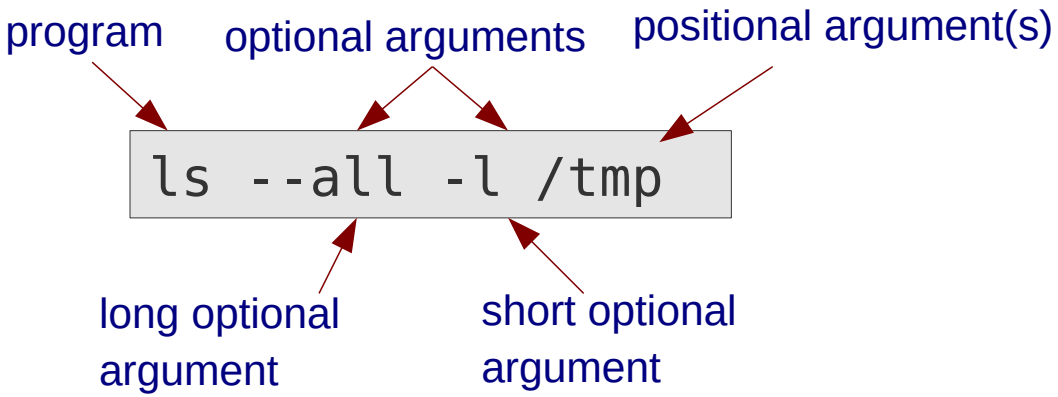
Scientific Programming in Python (2024)

https://atticlectures.net/scipro/python-2024/

CC BY SA

Command line arguments

- Command line programs usually accept various arguments which control their behaviour



Positional arguments (“what should the program act on”)

- Last arguments on the command line (usually)
- Order might matter
- Number of required positional arguments might be fixed (0, 1, 2) or arbitrary

Optional arguments (“how should the program behave”)

- Start with singe dash (short form) or double dash (long form)
- Always optional (program must work without any optional arguments)

Simple argparse example

- The command line arguments can be parsed in Python with the **ArgumentParser**

test.py

```
#!/usr/bin/env python3
import argparse


_DESCRIPTION = 'Test script demonstrating argparse'


parser = argparse.ArgumentParser(description=_DESCRIPTION)
msg = 'directory  (default: .)'
parser.add_argument('-d', '--directory', default='.', help=msg)
msg = 'arbitrary integer number'
parser.add_argument('number', type=int, metavar='NUM', help=msg)
args = parser.parse_args()
print(f"Directory: {args.directory}")
print(f"Number: {args.number:d}")
```

Use “python” on Windows
and “python3” on Linux

Testing the example

- On Linux/macOS: Make the Python-script executable: `chmod +x test.py`
- Passing the **-h** option shows a help page with detailed information about the script

```
./test.py -h
usage: test.py [-h] [-d DIRECTORY] NUM

Test script demonstrating argparse

positional arguments:
  NUM                   arbitrary integer number

optional arguments:
  -h, --help            show this help message and exit
  -d DIRECTORY, --directory DIRECTORY      directory (default: .)
```

Testing the example

- One positional argument only

```
./test.py 2
Directory: .
Number: 2
```

- One optional argument (in short form) and one positional argument

```
./test.py -d /tmp 2
Directory: /tmp
Number: 2
```

- One optional argument (in long form) and one positional one

```
./test.py --directory /tmp 2
Directory: /tmp
Number: 2
```

Testing the example

- Invoking without required positional argument

```
./test.py
usage: test.py [-h] [-d DIRECTORY] NUM
test.py: error: the following arguments are required: NUM
```

- Invoking with required positional argument of the wrong type

```
./test.py 3.2
usage: test.py [-h] [-d DIRECTORY] NUM
test.py: error: argument NUM: invalid int value: '3.2'
```

- Invoking with an invalid optional argument

```
./test.py -a 2
usage: test.py [-h] [-d DIRECTORY] NUM
test.py: error: unrecognized arguments: -a
```

Building an argument parser step by step

```
parser = argparse.ArgumentParser(description=_DESCRIPTION)
```

- Creates an argument parser
- An optional description has been provided, which will be used as the general description in the help.

```
msg = 'directory  (default: .)'
parser.add_argument('-d', '--directory', default='.', help=msg)
```

- Adds an optional argument with short form (**-d**) and long form (**--directory**) to the parser.
- As argument type is not explicitely defined, it assumes string by default.
- If the optional argument has not been specified at the command line, the given default value (**.**) will be used
- The option description is provided with the **help=** argument

Building an argument parser step by step

```
msg = 'arbitrary integer number'
parser.add_argument('number', type=int, metavar='NUM', help=msg)
```

- Adds a mandatory positional argument, named **number**
- Argument type should be an **integer**
- In the help **NUM** should be used as argument meta variable (shown in the help)
- Help message for the argument is provided

Parsing the command line

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

- Parse the command line arguments
- Returns the result as a **Namespace** object, which can be queried

```
print(f"Directory: {args.directory}")
print(f"Number: {args.number:d}")
```

- The parsed values can be accessed in the returned **Namespace** object using the dot notation.
- The name after the dot is the long name of the argument

See the Argparse Tutorial for more examples and further details