getopt — Command Line Option Parsing

Purpose: Command line option parsing

The `getopt` module is the original command line option parser that supports the conventions established by the Unix function `getopt`. It parses an argument sequence, such as `sys.argv` and returns a sequence of tuples containing (option, argument) pairs and a sequence of non-option arguments.

Supported option syntax includes short and long form options:

```
-a
-bval
-b val
--noarg
--witharg=val
--witharg val
```

Note

`getopt` is not deprecated, but `argparse` is more actively maintained and should be used for new development.

Function Arguments

The `getopt()` function takes three arguments:

- The first parameter is the sequence of arguments to be parsed. This usually comes from `sys.argv[1:]` (ignoring the program name in `sys.arg[0]`).
- The second argument is the option definition string for single character options. If one of the options requires an argument, its letter is followed by a colon.
- The third argument, if used, should be a sequence of the long-style option names. Long style options can be more than a single character, such as `--noarg` or `--witharg`. The option names in the sequence should not include the "`--`" prefix. If any long option requires an argument, its name should have a suffix of "=".

Short and long form options can be combined in a single call.

Short Form Options

This example program accepts three options. The `-a` is a simple flag, while `-b` and `-c` require an argument. The option definition string is `"ab:c:"`.

```
# getopt_short.py

import getopt

opts, args = getopt.getopt(['-a', '-bval', '-c', 'val'], 'ab:c:')

for opt in opts:
    print(opt)
```

The program passes a list of simulated option values to `getopt()` to show the way they are processed.

```
$ python3 getopt_short.py

('-a', '')
('-b', 'val')
('-c', 'val')
```

Long Form Options

For a program that takes two options, `--noarg` and `--witharg`, the long-argument sequence should be `[ 'noarg',`

'witharg='].

```
# getopt_long.py

import getopt

opts, args = getopt.getopt(
    ['--noarg',
     '--witharg', 'val',
     '--witharg2=another'],
    '',
    ['noarg', 'witharg=', 'witharg2='],
)
for opt in opts:
    print(opt)
```

Since this sample program does not take any short form options, the second argument to `getopt()` is an empty string.

```
$ python3 getopt_long.py

('--noarg', '')
('--witharg', 'val')
('--witharg2', 'another')
```

A Complete Example

This example is a more complete program that takes five options: `-o`, `-v`, `--output`, `--verbose`, and `--version`. The `-o`, `--output`, and `--version` options each require an argument.

```
# getopt_example.py
```

```
import getopt
import sys

version = '1.0'
verbose = False
output_filename = 'default.out'

print('ARGV      :', sys.argv[1:])

try:
    options, remainder = getopt.getopt(
        sys.argv[1:],
        'o:v',
        ['output=',
         'verbose',
         'version=',
         ])
except getopt.GetoptError as err:
    print('ERROR:', err)
    sys.exit(1)

print('OPTIONS   :', options)

for opt, arg in options:
    if opt in ('-o', '--output'):
        output_filename = arg
    elif opt in ('-v', '--verbose'):
        verbose = True
    elif opt == '--version':
        version = arg

print('VERSION   :', version)
print('VERBOSE   :', verbose)
print('OUTPUT    :', output_filename)
print('REMAINING :', remainder)
```

The program can be called in a variety of ways. When it is called without any arguments at all, the default settings are used.

```
$ python3 getopt_example.py

ARGV      : []
OPTIONS   : []
VERSION   : 1.0
VERBOSE   : False
OUTPUT    : default.out
REMAINING : []
```

A single letter option can be a separated from its argument by whitespace.

```
$ python3 getopt_example.py -o foo

ARGV      : ['-o', 'foo']
OPTIONS   : [('-o', 'foo')]
VERSION   : 1.0
VERBOSE   : False
OUTPUT    : foo
REMAINING : []
```

Or the option and value can be combined into a single argument.

```
$ python3 getopt_example.py -ofoo

ARGV      : ['-ofoo']
OPTIONS   : [('-o', 'foo')]
VERSION   : 1.0
VERBOSE   : False
OUTPUT    : foo
REMAINING : []
```

A long form option can similarly be separate from the value.

```
$ python3 getopt_example.py --output foo

ARGV      : ['--output', 'foo']
OPTIONS   : [('--output', 'foo')]
VERSION   : 1.0
VERBOSE   : False
OUTPUT    : foo
REMAINING : []
```

When a long option is combined with its value, the option name and value should be separated by a single =.

```
$ python3 getopt_example.py --output=foo

ARGV      : ['--output=foo']
OPTIONS   : [('--output', 'foo')]
VERSION   : 1.0
VERBOSE   : False
OUTPUT    : foo
REMAINING : []
```

Abbreviating Long Form Options

The long form option does not have to be spelled out entirely on the command line, as long as a unique prefix is provided.

```
$ python3 getopt_example.py --o foo

ARGV      : ['--o', 'foo']
OPTIONS   : [('--output', 'foo')]
VERSION   : 1.0
VERBOSE   : False
OUTPUT    : foo
REMAINING : []
```

If a unique prefix is not provided, an exception is raised.

```
$ python3 getopt_example.py --ver 2.0

ARGV      : ['--ver', '2.0']
ERROR: option --ver not a unique prefix
```

GNU-style Option Parsing

Normally, option processing stops as soon as the first non-option argument is encountered.

```
$ python3 getopt_example.py -v not_an_option --output foo

ARGV      : ['-v', 'not_an_option', '--output', 'foo']
OPTIONS   : [('-v', '')]
VERSION   : 1.0
VERBOSE   : True
OUTPUT    : default.out
REMAINING : ['not_an_option', '--output', 'foo']
```

To mix option and non-option arguments on the command line in any order, use `gnu_getopt()` instead.

```
# getopt_gnu.py

import getopt
import sys

version = '1.0'
verbose = False
output_filename = 'default.out'

print('ARGV      :', sys.argv[1:])

try:
    options, remainder = getopt.gnu_getopt(
        sys.argv[1:],
        'o:v',
        ['output=',
         'verbose',
         'version=',
         ])
except getopt.GetoptError as err:
    print('ERROR:', err)
    sys.exit(1)

print('OPTIONS   :', options)

for opt, arg in options:
    if opt in ('-o', '--output'):
        output_filename = arg
    elif opt in ('-v', '--verbose'):
        verbose = True
    elif opt == '--version':
        version = arg

print('VERSION   :', version)
print('VERBOSE   :', verbose)
print('OUTPUT    :', output_filename)
print('REMAINING :', remainder)
```

After changing the call in the previous example, the difference becomes clear.

```
$ python3 getopt_gnu.py -v not_an_option --output foo

ARGV      : ['-v', 'not_an_option', '--output', 'foo']
OPTIONS   : [('-v', ''), ('--output', 'foo')]
VERSION   : 1.0
VERBOSE   : True
OUTPUT    : foo
REMAINING : ['not an option']
```

Ending Argument Processing

If `getopt()` encounters "`--`" in the input arguments, it stops processing the remaining arguments as options. This feature can be used to pass argument values that look like options, such as filenames that start with a dash ("`-`").

```
$ python3 getopt_example.py -v -- --output foo

ARGV      : ['-v', '--', '--output', 'foo']
OPTIONS   : [('-v', '')]
VERSION   : 1.0
VERBOSE   : True
OUTPUT    : default.out
REMAINING : ['--output', 'foo']
```

See also

- Standard library documentation for getopt
- argparse – The argparse module replaces getopt for newer applications.

argparse — Command-Line Option and Argument Parsing

readline — The GNU readline Library

Quick Links

Function Arguments
Short Form Options
Long Form Options
A Complete Example
Abbreviating Long Form Options
GNU-style Option Parsing
Ending Argument Processing

This page was last updated 2016-12-30.

Navigation

argparse — Command-Line Option and Argument Parsing
readline — The GNU readline Library

The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?

This Site

Module Index
Index

Other Writing

- Blog
- The Python Standard Library By Example