Migrating optparse code to argparse

Release 3.14.0rc3

Guido van Rossum and the Python development team

September 18, 2025

Python Software Foundation
Email: docs@python.org

Contents

The `argparse` module offers several higher level features not natively provided by the `optparse` module, including:

- Handling positional arguments.
- Supporting subcommands.
- Allowing alternative option prefixes like `+` and `/`.
- Handling zero-or-more and one-or-more style arguments.
- Producing more informative usage messages.
- Providing a much simpler interface for custom `type` and `action`.

Originally, the `argparse` module attempted to maintain compatibility with `optparse`. However, the fundamental design differences between supporting declarative command line option processing (while leaving positional argument processing to application code), and supporting both named options and positional arguments in the declarative interface mean that the API has diverged from that of `optparse` over time.

As described in choosing-an-argument-parser, applications that are currently using `optparse` and are happy with the way it works can just continue to use `optparse`.

Application developers that are considering migrating should also review the list of intrinsic behavioural differences described in that section before deciding whether or not migration is desirable.

For applications that do choose to migrate from `optparse` to `argparse`, the following suggestions should be helpful:

- Replace all `optparse.OptionParser.add_option()` calls with `ArgumentParser.add_argument()` calls.
- Replace `(options, args) = parser.parse_args()` with `args = parser.parse_args()` and add additional `ArgumentParser.add_argument()` calls for the positional arguments. Keep in mind that what was previously called `options`, now in the `argparse` context is called `args`.
- Replace `optparse.OptionParser.disable_interspersed_args()` by using `parse_intermixed_args()` instead of `parse_args()`.
- Replace callback actions and the `callback_*` keyword arguments with `type` or `action` arguments.
- Replace string names for `type` keyword arguments with the corresponding type objects (e.g. int, float, complex, etc).
- Replace `optparse.Values` with `Namespace` and `optparse.OptionError` and `optparse.OptionValueError` with `ArgumentError`.

- Replace strings with implicit arguments such as `%default` or `%prog` with the standard Python syntax to use dictionaries to format strings, that is, `%(default)s` and `%(prog)s`.
- Replace the OptionParser constructor `version` argument with a call to `parser.add_argument('--version', action='version', version='<the version>')`.