**ArgParse**

This module makes it easy to write user friendly command line interfaces.

The program defines what arguments it requires, and [argparse](https://docs.python.org/3/library/argparse.html" \l "module-argparse" \o "argparse: Command-line option and argument parsing library.) will figure out how to parse those out of [sys.argv](https://docs.python.org/3/library/sys.html" \l "sys.argv" \o "sys.argv).

When user give invalid arguments argparse automatically generates error messages.

Argparse Tutorial

What is it?

Parser for command-line options, arguments and subcommands

Why use it?

The argparse module makes it easy to write user-friendly command-line interfaces.

How does it do that?

The program defines what arguments it requires, and argparse will figure out how

to parse those out of sys.argv.

The argparse module also automatically generates help and usage messages and

issues errors when users give the program invalid arguments.

Concept

When you run the "ls" command without any options, it will default displaying the

contents of the current directory

If you run "ls" on a different directory that you currently are in, you would type

"ls directory\_name". The "directory\_name" is a "positional argument", which means

that the program know what to do with the value.

To get more information about a file we can use the "-l" switch.

The "-l" is knowns as an "optional argument"

If you want to display the help text of the ls command, you would type "ls --help"

Argparse

To start using the argparse module, we first have to import it.

import argparse

parser = argparse.ArgumentParser()

parser.parse\_args()

Run the code

Run the code with the --help option (Running the script without any options

results in nothing displayed to stdout)

python program.py --help (or python program.py -h)

usage: program.py [-h]

optional arguments:

-h, --help show this help message and exit

As seen above, even though we didnt specify any help arguments in our

script, its still giving us a nice help message.

This is the only option we get for free.

Positional arguments

In our "ls" example above, we made use of the positional arguments

"ls directory\_name".

Whenever we want to specify which command-line options the program will accept,

we use the "add\_argument()" method.

parser.add\_argument("echo") # naming it "echo"

args = parser.parse\_args() # returns data from the options specified (echo)

print(args.echo)

If we now run the code, we can see it requires us to specify an option

$ python program.py

usage: program.py [-h] echo

program.py: error: too few arguments

When we specify the echo option it will display "echo"

$ python program.py echo

echo

#Using the --help option

$ python program.py --help

usage: program.py [-h] echo

positional arguments:

echo

optional arguments:

-h, --help show this help message and exit

Extending the help text

To get more help about our positional argument (echo), we have to change our script.

import argparse

parser = argparse.ArgumentParser()

parser.add\_argument("echo", help="echo the string you use here")

args = parser.parse\_args()

print(args.echo)

Result in this:

$ python program.py --help

usage: program.py [-h] echo

positional arguments:

echo echo the string you use here

optional arguments:

-h, --help show this help message and exit

Note:

Argparse treats the options we give as a string, but we can change that.

Running the code with the type set to Integer

This code will treat the input as an integer.

import argparse

parser = argparse.ArgumentParser()

parser.add\_argument("square", help="display a square of a given number",

type=int)

args = parser.parse\_args()

print(args.square\*\*2)

If we run the program with the --help option, we can see:

$ python program.py -h

usage: program.py [-h] square

positional arguments:

square display a square of a given number

optional arguments:

-h, --help show this help message and exit

Run the program

From the help text, we can see that if we give the program a number, it will

give us the square back.

Cool, lets try it out:

$ python program.py 4

16

$ python program.py 10

100

If we would use a string instead of a number, the program will return an error

$ python program.py four

usage: program.py [-h] square

program.py: error: argument square: invalid int value: 'four'

Optional arguments

In our "ls" example above, we made use of the optional argument "-l" to get more

information about a file.

The program below will display something when --verbosity is specified and display

nothing when not.

import argparse

parser = argparse.ArgumentParser()

parser.add\_argument("--verbose", help="increase output verbosity",

action="store\_true")

args = parser.parse\_args()

if args.verbose:

print("verbosity turned on")

An optional argument (or option) is (by default) given None as a value when its

not being used.

Using the --verbosity option, only two values are actually useful, True or False.

The keyword "action" is being given the value "store\_true" which means that if

the option is specifed, then assign the value "True" to args.verbose

Not specifying the option implies False.

If we run the program with the --help option, we can see:

$ python program.py -h

usage: program.py [-h] [--verbose]

optional arguments:

-h, --help show this help message and exit

--verbose increase output verbosity

Run the program using the --verbose option

$ python program.py --verbose

verbosity turned on

Short options

Using short versions of the options is as easy as:

parser.add\_argument("-v", "--verbose", help="increase output verbosity",

action="store\_true")

The help text will updated with the short version.

By default required = False if we make it True then if we not pass any argument it shows an error

ArgumentParser objects

*class*argparse.**ArgumentParser**(*prog=None*, *usage=None*, *description=None*, *epilog=None*, *parents=[]*, *formatter\_class=argparse.HelpFormatter*, *prefix\_chars='-'*, *fromfile\_prefix\_chars=None*, *argument\_default=None*, *conflict\_handler='error'*, *add\_help=True*, *allow\_abbrev=True*)

Create a new [ArgumentParser](https://docs.python.org/3/library/argparse.html" \l "argparse.ArgumentParser" \o "argparse.ArgumentParser) object. All parameters should be passed as keyword arguments. Each parameter has its own more detailed description below, but in short they are:

* [prog](https://docs.python.org/3/library/argparse.html#prog) - The name of the program (default: sys.argv[0])
* [usage](https://docs.python.org/3/library/argparse.html#usage) - The string describing the program usage (default: generated from arguments added to parser)
* [description](https://docs.python.org/3/library/argparse.html#description) - Text to display before the argument help (default: none)
* [epilog](https://docs.python.org/3/library/argparse.html#epilog) - Text to display after the argument help (default: none)
* [parents](https://docs.python.org/3/library/argparse.html#parents) - A list of [ArgumentParser](https://docs.python.org/3/library/argparse.html" \l "argparse.ArgumentParser" \o "argparse.ArgumentParser) objects whose arguments should also be included
* [formatter\_class](https://docs.python.org/3/library/argparse.html#formatter-class) - A class for customizing the help output
* [prefix\_chars](https://docs.python.org/3/library/argparse.html#prefix-chars) - The set of characters that prefix optional arguments (default: ‘-‘)
* [fromfile\_prefix\_chars](https://docs.python.org/3/library/argparse.html#fromfile-prefix-chars) - The set of characters that prefix files from which additional arguments should be read (default: None)
* [argument\_default](https://docs.python.org/3/library/argparse.html#argument-default) - The global default value for arguments (default: None)
* [conflict\_handler](https://docs.python.org/3/library/argparse.html#conflict-handler) - The strategy for resolving conflicting optionals (usually unnecessary)
* [add\_help](https://docs.python.org/3/library/argparse.html#add-help) - Add a -h/--help option to the parser (default: True)
* [allow\_abbrev](https://docs.python.org/3/library/argparse.html#allow-abbrev) - Allows long options to be abbreviated if the abbreviation is unambiguous. (default: True)

The add\_argument() method

ArgumentParser.**add\_argument**(*name or flags...*[, *action*][, *nargs*][, *const*][, *default*][, *type*][, *choices*][, *required*][, *help*][, *metavar*][, *dest*])

Define how a single command-line argument should be parsed. Each parameter has its own more detailed description below, but in short they are:

* [name or flags](https://docs.python.org/3/library/argparse.html#name-or-flags) - Either a name or a list of option strings, e.g. foo or -f, --foo.
* [action](https://docs.python.org/3/library/argparse.html#action) - The basic type of action to be taken when this argument is encountered at the command line.
* [nargs](https://docs.python.org/3/library/argparse.html#nargs) - The number of command-line arguments that should be consumed.
* [const](https://docs.python.org/3/library/argparse.html#const) - A constant value required by some [action](https://docs.python.org/3/library/argparse.html#action) and [nargs](https://docs.python.org/3/library/argparse.html" \l "nargs) selections.
* [default](https://docs.python.org/3/library/argparse.html#default) - The value produced if the argument is absent from the command line.
* [type](https://docs.python.org/3/library/argparse.html#type) - The type to which the command-line argument should be converted.
* [choices](https://docs.python.org/3/library/argparse.html#choices) - A container of the allowable values for the argument.
* [required](https://docs.python.org/3/library/argparse.html#required) - Whether or not the command-line option may be omitted (optionals only).
* [help](https://docs.python.org/3/library/argparse.html#help) - A brief description of what the argument does.
* [metavar](https://docs.python.org/3/library/argparse.html#metavar) - A name for the argument in usage messages.
* [dest](https://docs.python.org/3/library/argparse.html#dest) - The name of the attribute to be added to the object returned by [parse\_args()](https://docs.python.org/3/library/argparse.html" \l "argparse.ArgumentParser.parse_args" \o "argparse.ArgumentParser.parse_args).

In nargs=’+’ means One or more, ‘\*’ means zero or more,’?’ means zero or one.