Python Command Line Argument Parsing Positional and Optional Arguments

Fan Wang

2020-12-19

Contents

1 Python Command-line Arguments Parsing

Go to the **RMD**, **PDF**, or **HTML** version of this file. Go back to Python Code Examples Repository (bookdown site) or the pyfan Package (API).

```
import pprint
import argparse
```

1.1 Positional and Optional Arguments

Provide a positional and an optional argument. Position arguments are provided positions when the module is called, without prefixed by which parameter this is. Optional argument requires parameter specification.

```
# Start parser for arguments
parser = argparse.ArgumentParser()

# Positional argument 1st, will be stored as int
parser.add_argument('esrtype', type=int, help='positional argument 1st')
# Positional argument 2nd, will be stored as string
```

```
## _StoreAction(option_strings=[], dest='esrtype', nargs=None, const=None, default=None, type=<class 'in
```

```
parser.add_argument('speckey', type=str, help='positional argument 2nd')

# Optional argument
```

```
## _StoreAction(option_strings=[], dest='speckey', nargs=None, const=None, default=None, type=<class 'st
```

```
parser.add_argument('-A', type=str, default='opt_arg_A_default_str_val')

# Call with positional argument specified
# Note that one is bracketed, will be interpreted as int
```

```
## _StoreAction(option_strings=['-A'], dest='A', nargs=None, const=None, default='opt_arg_A_default_str
print(f"Must specify posi. arg: {parser.parse_args(['1', 'mpoly_1=esti_tinytst_mpoly_13=3=3'])=}")
# Call with two positional arguments and one optional
# Note that the first positional argument becomes int, second beomce str
```

```
## Must specify posi. arg: parser.parse_args(['1', 'mpoly_1=esti_tinytst_mpoly_13=3=3'])=Namespace(A='o
print(f"With opt arg: {parser.parse_args(['1', '2', '-A', 'abc'])=}")
```

```
## With opt arg: parser.parse_args(['1', '2', '-A', 'abc'])=Namespace(A='abc', esrtype=1, speckey='2')
```

1.2 Short and Long Parameter Name Specifications

Test below a boolean parameter that will be true or false. The default value is False. The parameter is called *boolparam* with short name *abc*. There is a variety of ways of setting the parameter to true.

```
# Start parser for arguments
parser = argparse.ArgumentParser()

# short name for the first parameter is a, full name is abc, boolean parameter
parser.add_argument('-abc', '--boolparam', action="store_true", default=False)

# default is false but turn on option so true
```

```
## _StoreTrueAction(option_strings=['-abc', '--boolparam'], dest='boolparam', nargs=0, const=True, defau
print(f"default false: {parser.parse_args()=}")
```

```
## default false: parser.parse_args()=Namespace(boolparam=False)
print(f"default false, set to true, short all: {parser.parse_args(['-abc'])=}")
```

```
## default false, set to true, short all: parser.parse_args(['-abc'])=Namespace(boolparam=True)
print(f"default false, set to true, short part ab for abc: {parser.parse_args(['-ab'])=}")
```

```
## default false, set to true, short part ab for abc: parser.parse_args(['-ab'])=Namespace(boolparam=Tru
print(f"default false, set to true, short part a for abc: {parser.parse_args(['-a'])=}")
```

```
## default false, set to true, short part a for abc: parser.parse_args(['-a'])=Namespace(boolparam=True
print(f"default false, set to true, full param: {parser.parse_args(['--boolparam'])=}")
```

```
## default false, set to true, full param: parser.parse_args(['--boolparam'])=Namespace(boolparam=True)
print(f"default false, set to true, full param: {parser.parse_args(['--boolparam'])=}")
```

```
## default false, set to true, full param: parser.parse_args(['--boolparam'])=Namespace(boolparam=True)
```

1.3 A List of Allowed Values

There is a parameter, only some specific values are allowed. Also provide help for each allowed option. Note added *argparse.RawTextHelpFormatter* to parse the next lines in help.

```
# Start parse
parser = argparse.ArgumentParser(description='Run ESR cmd', formatter_class=argparse.RawTextHelpFormatt

# A required positional argument parameter tht is int and can take eight possible values
parser.add_argument('esrtype', type=int,
```

```
                    choices=[1, 2, 3, 4, 5, 6, 7, 8],
                    help='1. Simulate at N sets of parameter combinations\n'
                         '2. Polynomial approximation surface based on (1) '
                         'for each outcome of interest, find best\n'
                         '3. Estimation at N sets of starting points with (2) as objective function\n'
                         '4. Gather results frorm (3), find M best.\n'
                         '5. Simulate (estimate once) at the top M best results from (4) actual model,
                         'compare objective to approximated from (3)\n'
                         '6. Gather results from (5), re-rank best of the M best from (4)\n'
                         '7. Estimate at the top M best results from (4) actual model, '
                         '(4) M best are M best seeds\n'
                         '8. Gather results from (7), re-rank best of the final results from the M best
# Print defaults
```

```
## _StoreAction(option_strings=[], dest='esrtype', nargs=None, const=None, default=None, type=<class 'in
```

```
print(f"provide 1 for the value of the positional argument: {parser.parse_args(['1'])=}")
```

```
## provide 1 for the value of the positional argument: parser.parse_args(['1'])=Namespace(esrtype=1)
```

1.4 Boolean, Integer, String and list Parameters

How to handle parameters of different types, boolean, integer, string and list. For these four types, the same way to specify short and long parameter names. How to set the parameter types, and how to set default values for each type.

```
# Start parser for arguments
parser = argparse.ArgumentParser()

# Single letter string parameters
# Note dest name over-rides full name
parser.add_argument('-cta', '--cttaaaaa', dest="combo_type_a", default='e',
                    type=str)
# Multiple letters and integers
# Note without dest full name is dest
```

```
## _StoreAction(option_strings=['-cta', '--cttaaaaa'], dest='combo_type_a', nargs=None, const=None, defa
```

```
parser.add_argument('-ctb', '--combo_type_b', default='20201025',
                    type=str)
# Multiple letters and integers
# Note without dest and full name short name is parameter name
```

```
## _StoreAction(option_strings=['-ctb', '--combo_type_b'], dest='combo_type_b', nargs=None, const=None,
```

```
parser.add_argument('-ctc', default=['list_tKap_mlt_ce1a2'],
                    nargs='+', type=str)

# Print defaults
```

```
## _StoreAction(option_strings=['-ctc'], dest='ctc', nargs='+', const=None, default=['list_tKap_mlt_ce1a
```

```
print(f"default false: {parser.parse_args()=}")
# change parameters
```

```
## default false: parser.parse_args()=Namespace(combo_type_a='e', combo_type_b='20201025', ctc=['list_tK
```

```
print(f"default false: {parser.parse_args(['-ctb', '20201111'])=}")
```

```
## default false: parser.parse_args(['-ctb', '20201111'])=Namespace(combo_type_a='e', combo_type_b='202
```

see variable-argument-lists.

1.5 Parse multiple parameter types

Provide several types of parameters to a function, so that the function can be called easily container call to execute estimation. The types of parameters includes:

1. A list including parameter information
2. A string including estimation/computational controls
3. Additional parameters

```
# Start parser for arguments
parser = argparse.ArgumentParser()

# First (and only) positional argument for esrtype:
parser.add_argument('esrtype', type=int, help='positional argument')

# Optional argument
```

```
## _StoreAction(option_strings=[], dest='esrtype', nargs=None, const=None, default=None, type=<class 'in
```

```
parser.add_argument('-s', dest='speckey', type=str,
                    default='ng_s_t=esti_tinytst_thin_1=3=3',
                    help="compute and esti keys and omments")

# abc and e of comb_type
```

```
## _StoreAction(option_strings=['-s'], dest='speckey', nargs=None, const=None, default='ng_s_t=esti_tiny
```

```
parser.add_argument('-cta', dest="combo_type_a", default='e', type=str)
```

```
## _StoreAction(option_strings=['-cta'], dest='combo_type_a', nargs=None, const=None, default='e', type=
```

```
parser.add_argument('-ctb', dest="combo_type_b", default='20201025', type=str)
```

```
## _StoreAction(option_strings=['-ctb'], dest='combo_type_b', nargs=None, const=None, default='20201025
```

```
parser.add_argument('-ctc', dest="combo_type_c", default=['list_tKap_mlt_ce1a2'], nargs='+', type=str)
```

```
## _StoreAction(option_strings=['-ctc'], dest='combo_type_c', nargs='+', const=None, default=['list_tKap
```

```
parser.add_argument('-cte1', dest="combo_type_e1", default=None, type=str)
```

```
## _StoreAction(option_strings=['-cte1'], dest='combo_type_e1', nargs=None, const=None, default=None, ty
```

```
parser.add_argument('-cte2', dest="combo_type_e2", default='mpoly_1=esti_tinytst_mpoly_13=3=3', type=str

# other parameters
```

```
## _StoreAction(option_strings=['-cte2'], dest='combo_type_e2', nargs=None, const=None, default='mpoly_1
```

```
parser.add_argument('-f', dest="save_directory_main", default='esti')

# Default, must specify erstype
```

```
## _StoreAction(option_strings=['-f'], dest='save_directory_main', nargs=None, const=None, default='esti
```

```
print(f"default false: {parser.parse_args(['1'])=}")
# Print with the nargs+ arguments
# specified two elements, abc, and efg for nargs ctc, becomes a string list
```

```
## default false: parser.parse_args(['1'])=Namespace(combo_type_a='e', combo_type_b='20201025', combo_ty
```

```
print(f"default false: {parser.parse_args(['1', '-ctc', 'abc', 'efg'])=}")
# one input for ctc, still generates a list
```

```
## default false: parser.parse_args(['1', '-ctc', 'abc', 'efg'])=Namespace(combo_type_a='e', combo_type
```

```
print(f"default false: {parser.parse_args(['1', '-ctc', 'abc'])=}")
```

```
## default false: parser.parse_args(['1', '-ctc', 'abc'])=Namespace(combo_type_a='e', combo_type_b='202
```