

# PARSEARG: TURNS ARGPARSE ON ITS HEAD, THE DECLARATIVE WAY

THOMAS P. HARTE

## CONTENTS

1. Quickstart	1
1.1. Overview	1
1.2. Usage	1
2. Overview	3
3. Let's build the interface for a TO-DO app	4
4. How parsearg works	7
4.1. The A-tree	7
4.2. The A-A tree	8

## 1. QUICKSTART

**1.1. Overview.** `parsearg` is a Python package for writing command-line interfaces (“CLI”) that augments (rather than replaces) the standard Python module for writing CLIs, `argparse`. There is nothing wrong with `argparse`: It’s fine in terms of the *functionality* that it provides, but it can be clunky to use, especially when a program’s structure has subcommands, or nested subcommands (*i.e.* subcommands that have subcommands). Moreover, because of the imperative nature of `argparse`, it makes it hard to understand how a program’s interface is structured (*viz.* the program’s “view”).

`parsearg` puts a layer on top of `argparse` that makes writing a CLI easy: You declare your view (*i.e.* the CLI), with a dict so that the view is a data structure (*i.e.* pure configuration). The data structure declares the *intent* of the CLI and you no longer have to instruct `argparse` on how to put the CLI together: `parsearg` does that for you. In this respect, `parsearg` turns `argparse` on its head, in the sense that it replaces imperative instructions with declarative data.

**1.2. Usage.** Suppose we wish to create a program called `quickstart-todos.py` to manage the TO-DOs of a set of different users. We want to have subprograms of `quickstart-todos.py`; for example, we may want to create a user (`python quickstart-todos.py create user`, say), or we may want to create a TO-DO for a particular user (`python quickstart-todos.py create todo`, say). We might also want to add optional parameters to each subprogram such as the user’s email and phone number, or the TO-DO’s due date. An invocation of the program’s CLI might look like the following:

```
1 python quickstart-todos.py create user Bob --email=bob@email.com --phone=+1-212-555-1234
2 python quickstart-todos.py create todo Bob 'taxes' --due-date=2021-05-17
```

With `argparse`, the subprogram `create` would necessitate fiddling with subparsers. With `parsearg`, the CLI for the above is declared with a dict and `parsearg.parser.ParseArg` supplants the normal use of `argparse.ArgumentParser`. Moreover, the callback associated with each subcommand is explicitly linked to its declaration.

```

1 import sys
2 from parsearg import ParseArg
3
4 def create_user(args):
5     print(f'created user: {args.name!r} (email: {args.email}, phone: {args.phone})')
6
7 def create_todo(args):
8     print(f'created TO-DO for user {args.user!r}: {args.title} (due: {args.due_date})')
9
10 view = {
11     'create|user': {
12         'callback': create_user,
13         'name': {'help': 'create user name', 'action': 'store'},
14         '-e|--email': {'help': "create user's email address", 'action': 'store', 'default': ''},
15         '-p|--phone': {'help': "create user's phone number", 'action': 'store', 'default': ''},
16     },
17     'create|todo': {
18         'callback': create_todo,
19         'user': {'help': 'user name', 'action': 'store'},
20         'title': {'help': 'title of TO-DO', 'action': 'store'},
21         '-d|--due-date': {'help': 'due date for the TO-DO', 'action': 'store', 'default': None},
22     },
23 }
24
25 def main(args):
26     # ParseArg takes the place of argparse.ArgumentParser
27     parser = ParseArg(d=view)
28
29     # parser.parse_args returns an argparse.Namespace
30     ns = parser.parse_args(args)
31
32     # ns.callback contains the function in the 'callback' key of 'view'
33     result = ns.callback(ns)
34
35 if __name__ == "__main__":
36     args = sys.argv[1:] if len(sys.argv) > 1 else []
37
38     main(' '.join(args))

```

A fully-worked version of the TO-DO example is presented in the docs. The output of the above is:

```
1 python quickstart-todos.py create user Bob --email=bob@email.com --phone=212-555-1234
```

```
created user: 'Bob' (email: bob@email.com, phone: 212-555-1234)
```

```
1 python quickstart-todos.py create todo Bob 'taxes' --due-date=2021-05-17
```

```
created TO-DO for user 'Bob': taxes (due: 2021-05-17)
```

Because parsearg is built on top of argparse, all the usual features are available, such as the extensive help features (essentially making the CLI self-documenting):

```
1 python quickstart-todos.py --help
```

```
usage: quickstart-todos.py [-h] {create} ...

positional arguments:
  {create}

optional arguments:
  -h, --help  show this help message and exit
```

```
1 python quickstart-todos.py create --help
```

```
usage: quickstart-todos.py create [-h] {user,todo} ...

positional arguments:
  {user,todo}

optional arguments:
  -h, --help  show this help message and exit
```

```
1 python quickstart-todos.py create user --help
```

```
usage: quickstart-todos.py create user [-h] [-e EMAIL] [-p PHONE] name

positional arguments:
  name                create user name

optional arguments:
  -h, --help          show this help message and exit
  -e EMAIL, --email EMAIL
                     create user's email address
  -p PHONE, --phone PHONE
                     create user's phone number
```

```
1 python quickstart-todos.py create todo --help
```

```
usage: quickstart-todos.py create todo [-h] [-d DUE_DATE] user title

positional arguments:
  user                user name
  title               title of TO-DO

optional arguments:
  -h, --help          show this help message and exit
  -d DUE_DATE, --due-date DUE_DATE
                     due date for the TO-DO
```

## 2. OVERVIEW

The “standard” Python module for writing command-line interfaces (“CLI”) is `argparse`. It is standard in so far as it is one of the batteries that comes included with the Python distribution, so no special installation is required. Probably because `argparse` is a bit clunky to use, many other (non-standard) packages have been developed for creating CLIs. Why “clunky”? Putting together a CLI with `argparse` alone is nothing if not an exercise in imperative programming, and this has three very negative consequences:

- (1) It obfuscates the intention of the CLI design;
- (2) It is prone to errors;
- (3) It discourages CLI design in the first instance; it makes debugging a CLI design very difficult; and it makes refactoring or re-configuring the CLI design overly burdensome.

In spite of this clunkiness, `argparse` has everything we need in terms of functionality. `parsearg`, then, is nothing more than a layer over `argparse` that exposes the `argparse` functionality via a `dict`. The `dict` is the View component of the [Model-View-Controller](#) (“MVC”) design pattern. The `dict` embeds callbacks from the Controller component, thereby achieving a clean separation of duties, which is what the MVC pattern calls for. By separating the View component into a `dict`, the CLI design can be expressed in a declarative way: `parsearg` manifests the *intention* of the CLI design without having to specify how that design is implemented in terms of `argparse`’s parsers and subparsers (`parsearg` does that for you).

Other packages—such as `click` and `plac`—effectively decorate functions that are part of the Controller with functionality from the View. Unfortunately, while this may expose the functionality of `argparse` in a more friendly way via the packages’ decorators, it dissipates the elements of the View across the Controller and in so doing it makes the CLI design difficult to grasp.

The `parsearg` philosophy is that `argparse` is already good enough in terms of the functionality that it provides, but that it just needs a little nudge in terms of how it's used. Arguments to be added to a CLI with `argparse` can be clearly specified as data, as can the callbacks that consume these arguments. `parsearg` takes advantage of this by specifying everything (in the View component of MVC) as a `dict`, from which `parsearg` then generates a parser (or set of nested parsers) using `argparse`. The Controller is then free to use the generated parser.

The `parsearg` approach is declarative because it manifests the CLI design in a data structure: a `dict`, which is one of Python's built-in data structures. The keys of this `dict` form a flattened tree of the CLI's subcommands. Keys like `A|B|C` are easy to specify and neatly summarize the nested hierarchy of subcommands: `A -> B -> C`. They are also easy to change. The magic, such as it is, of `parsearg` is that it unflattens this flattened tree into a tree of `argparse` parsers. `parsearg` requires nothing special: It works with Python out of the box, and therefore uses what's already available without introducing dependencies.

Simple? The following examples should help.

### 3. LET'S BUILD THE INTERFACE FOR A TO-DO APP

The `examples` folder in the source distribution contains a TO-DO app. The app

- (1) illustrates a sufficiently realistic, but not overly complex, problem;
- (2) illustrates operation of the MVC pattern in the wild;
- (3) shows how `parsearg` neatly segments the View component of MVC with a `dict`.

Let's start with the outer layer of the onion. How do we interact with `todos.py`? First, we can create some users in a User table with the `create user` subcommand of `todos.py`. Note that we do not (yet) have a phone number for user Dick, nor do we have an email address for user Harry:

```
1 python todos.py create user Tom -e tom@email.com -p 212-555-1234
2 python todos.py create user Dick -e dick@email.com
3 python todos.py create user Harry -p 212-123-5555
```

```
'create user Tom -e tom@email.com -p 212-555-1234':
-----
SUCCESS
'create user Dick -e dick@email.com':
-----
SUCCESS
'create user Harry -p 212-123-5555':
-----
SUCCESS
```

Second, create some TO-DOs in the Todo table with the `create todo` subcommand of `todos.py`.

```
1 python todos.py create todo Tom title1 -c description1 -d 2020-11-30
2 python todos.py create todo Tom title2 -c description2 --due-date=2020-12-31
3 python todos.py create todo Harry todo-1 --description=Christmas-party -d 2020-11-30
4 python todos.py create todo Harry todo-2 --description=New-Year-party
```

```
'create todo Tom title1 -c description1 -d 2020-11-30':
-----
SUCCESS
'create todo Tom title2 -c description2 --due-date=2020-12-31':
-----
SUCCESS
'create todo Harry todo-1 --description=Christmas-party -d 2020-11-30':
-----
SUCCESS
'create todo Harry todo-2 --description=New-Year-party':
-----
SUCCESS
```

Let's make some changes to the records entered so far. We can add an email address for user Harry (using the `update user email` subcommands) and a phone number for user Dick (using the `update user phone` subcommands):

```
1 python todos.py update user email Harry harry@email.com
2 python todos.py update user phone Dick 203-555-1212
```

```
'update user email Harry harry@email.com':
-----
      SUCCESS
'update user phone Dick 203-555-1212':
-----
      SUCCESS
```

Now update two of the TO-DOs, changing the title (using the `update todo title` subcommands) and the description (using the `update todo description` subcommands) in the fourth TO-DO:

```
1 python todos.py update todo title 4 most-important
2 python todos.py update todo description 4 2021-party
```

```
'update todo title 4 most-important':
-----
      SUCCESS
'update todo description 4 2021-party':
-----
      SUCCESS
```

```
1 python todos.py show users
2 python todos.py show todos
```

```
'show users':
-----
      SUCCESS
'show todos':
-----
      SUCCESS
```

The result of these commands is that the two tables (User and Todo) are populated in a [SQLite](#) database:

```
1 sqlite3 todo.db 'select * from User;'
```

```
Tom|tom@email.com|212-555-1234|2021-06-12 18:52:58
Dick|dick@email.com|203-555-1212|2021-06-12 18:52:58
Harry|harry@email.com|212-123-5555|2021-06-12 18:52:58
```

```
1 sqlite3 todo.db 'select * from Todo;'
```

```
25|title1|description1|2020-11-30|2021-06-12 18:52:58|Tom
26|title2|description2|2020-12-31|2021-06-12 18:52:58|Tom
27|todo-1|Christmas-party|2020-11-30|2021-06-12 18:52:58|Harry
28|todo-2|New-Year-party|None|2021-06-12 18:52:58|Harry
```

Let's look at the Python code for [todos.py](#).

The View component is entirely contained within a single dict, *viz.* `view`, which has been formatted here for clarity using `parsearg.utils.show`:

```
1 from todos import view
2 from parsearg.utils import show
3
4 show(view)
```

```

'purge|users':
  'callback':
    <function purge_users at 0x7f47f9487af0>
'purge|todos':
  'callback':
    <function purge_todos at 0x7f47f942b820>
'show|users':
  'callback':
    <function show_users at 0x7f47f942b8b0>
'show|todos':
  'callback':
    <function show_todos at 0x7f47f942b940>
'create|user':
  'callback':
    <function create_user at 0x7f47f942b9d0>
  'name':
    {'help': 'create user name', 'action': 'store'}
  '-e|--email':
    {'help': 'create user's email address', 'action': 'store', 'default': ''}
  '-p|--phone':
    {'help': 'create user's phone number', 'action': 'store', 'default': ''}
'create|todo':
  'callback':
    <function create_todo at 0x7f47f942ba60>
  'user':
    {'help': 'user name', 'action': 'store'}
  'title':
    {'help': 'title of to-do', 'action': 'store'}
  '-c|--description':
    {'help': 'description of to-do', 'action': 'store', 'default': ''}
  '-d|--due-date':
    {'help': 'due date for the to-do', 'action': 'store', 'default': None}
'update|user|email':
  'callback':
    <function update_user_email at 0x7f47f942baf0>
  'name':
    {'help': 'user name', 'action': 'store'}
  'email':
    {'help': 'user email', 'action': 'store'}
'update|user|phone':
  'callback':
    <function update_user_phone at 0x7f47f942bb80>
  'name':
    {'help': 'user name', 'action': 'store'}
  'phone':
    {'help': 'user phone', 'action': 'store'}
'update|todo|title':
  'callback':
    <function update_todo_title at 0x7f47f942bc10>
  'id':
    {'help': 'ID of to-do', 'action': 'store'}
  'title':
    {'help': 'title of to-do', 'action': 'store'}
'update|todo|description':
  'callback':
    <function update_todo_description at 0x7f47f942bca0>
  'id':
    {'help': 'ID of to-do', 'action': 'store'}
  'description':
    {'help': 'description of to-do', 'action': 'store'}

```

We can generate a tree view of the CLI design specified by the above dict, namely view, as follows:

```

1 from parsearg import ParseArg
2
3 print(
4     ParseArg(d=view, root_name='TODO').tree.show(quiet=True)
5 )

```

```

TODO
  update
    user
      phone

```

```

    email
  todo
    description
    title
  purge
    todos
    users
  create
    user
    todo
  show
    todos
    users

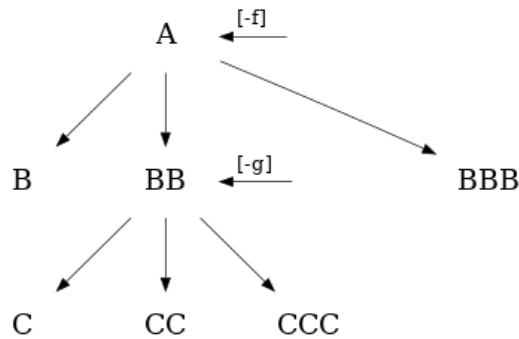
```

#### 4. HOW PARSEARG WORKS

It is easier to explain how `parsearg` works with a simpler abstraction than the above example `parsearg.examples.todos.py`. Here, we will consider nested parsers as trees. We introduce two trees for this purpose:

- (1) The “A tree”, and
- (2) The “A-AA tree”.

**4.1. The A-tree.** The “A tree” has three levels. As each node of the tree must necessarily occupy a positional argument of the command line, `[-f]` and `[-g]` are correspondingly *optional arguments* that attach to the nodes (A and BB, respectively, in the below diagram).



Let’s look at the Python code for [a.py](#).

Consider the dict that represents the View component:

```

1 from a import view
2
3 show(view)

```

```

'A':
  'callback':
    <function make_callback.<locals>.func at 0x7f47f94501f0>
  '-c':
    {'help': 'A [optional pi]', 'action': 'store_const', 'const': 3.141593}
  '-v|--verbose':
    {'help': 'A verbosity', 'action': 'store_true'}
'A|B':
  'callback':
    <function make_callback.<locals>.func at 0x7f47f936ef70>
  '-c':
    {'help': 'A B [optional pi]', 'action': 'store_const', 'const': 3.141593}
  '-v|--verbose':
    {'help': 'A B verbosity', 'action': 'store_true'}
'A|BB':
  'callback':

```

```

<function make_callback.<locals>.func at 0x7f47f93d7040>
'-c':
{'help': 'A BB [optional pi]', 'action': 'store_const', 'const': 3.141593}
'-v|--verbose':
{'help': 'A BB verbosity', 'action': 'store_true'}
'A|BB|C':
'callback':
<function make_callback.<locals>.func at 0x7f47f93d70d0>
'-c':
{'help': 'A BB C [optional pi]', 'action': 'store_const', 'const': 3.141593}
'-v|--verbose':
{'help': 'A BB C verbosity', 'action': 'store_true'}
'A|BB|CC':
'callback':
<function make_callback.<locals>.func at 0x7f47f93d7160>
'-c':
{'help': 'A BB CC [optional pi]', 'action': 'store_const', 'const': 3.141593}
'-v|--verbose':
{'help': 'A BB CC verbosity', 'action': 'store_true'}
'A|BB|CCC':
'callback':
<function make_callback.<locals>.func at 0x7f47f93d71f0>
'-c':
{'help': 'A BB CCC [optional pi]', 'action': 'store_const', 'const': 3.141593}
'-v|--verbose':
{'help': 'A BB CCC verbosity', 'action': 'store_true'}
'A|BBB':
'callback':
<function make_callback.<locals>.func at 0x7f47f93d7280>
'-c':
{'help': 'A BBB [optional pi]', 'action': 'store_const', 'const': 3.141593}
'-v|--verbose':
{'help': 'A BBB verbosity', 'action': 'store_true'}

```

and then the tree that the parsed dict is represented by:

```

1 print(
2     ParseArg(d=view, root_name='TODO').tree.show(quiet=True)
3 )

```

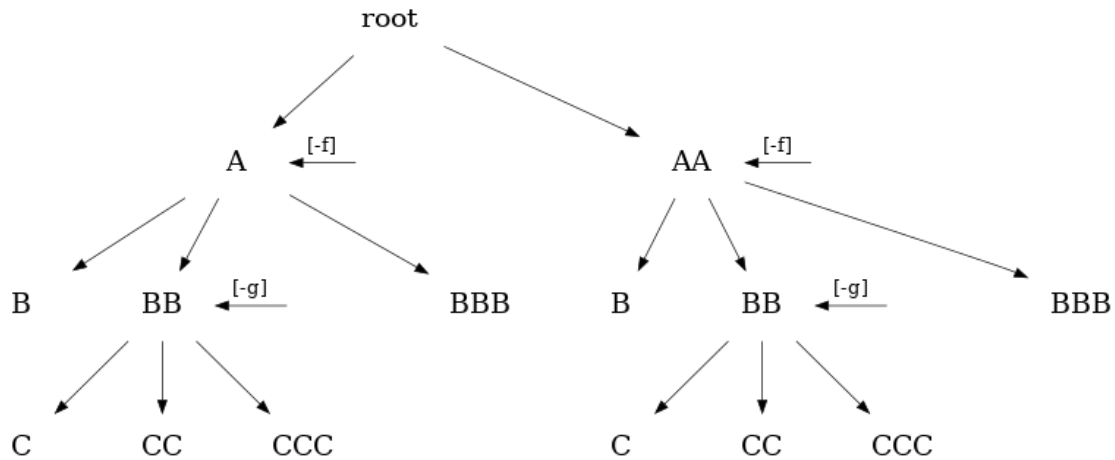
```

TODO
  A
    B
    BB
      C
      CCC
      CC
    BBB

```

4.2. **The A-A tree.** The A-AA tree simply extends the A tree one level:





Let's look at the Python code for [a\\_aa.py](#).

Consider the dict that represents the View component:

```

1 from a_aa import view
2
3 show(view)

```

```

'A':
  'callback':
    <function make_callback.<locals>.func at 0x7f47f9377e50>
  '-c':
    {'help': 'A [optional pi]', 'action': 'store_const', 'const': 3.141593}
  '-v|--verbose':
    {'help': 'A verbosity', 'action': 'store_true'}
'A|B':
  'callback':
    <function make_callback.<locals>.func at 0x7f47f9377ee0>
  '-c':
    {'help': 'A B [optional pi]', 'action': 'store_const', 'const': 3.141593}
  '-v|--verbose':
    {'help': 'A B verbosity', 'action': 'store_true'}
'A|BB':
  'callback':
    <function make_callback.<locals>.func at 0x7f47f9377dc0>
  '-c':
    {'help': 'A BB [optional pi]', 'action': 'store_const', 'const': 3.141593}
  '-v|--verbose':
    {'help': 'A BB erbosity', 'action': 'store_true'}
'A|BB|C':
  'callback':
    <function make_callback.<locals>.func at 0x7f47f9377040>
  '-c':
    {'help': 'A BB C [optional pi]', 'action': 'store_const', 'const': 3.141593}
  '-v|--verbose':
    {'help': 'A BB C verbosity', 'action': 'store_true'}
'A|BB|CC':
  'callback':
    <function make_callback.<locals>.func at 0x7f47f93770d0>
  '-c':
    {'help': 'A BB CC [optional pi]', 'action': 'store_const', 'const': 3.141593}
  '-v|--verbose':
    {'help': 'A BB CC verbosity', 'action': 'store_true'}
'A|BB|CCC':
  'callback':
    <function make_callback.<locals>.func at 0x7f47f9377160>
  '-c':
    {'help': 'A BB CCC [optional pi]', 'action': 'store_const', 'const': 3.141593}
  '-v|--verbose':

```

```

    {'help': 'A BB CCC verbosity', 'action': 'store_true'}
'A|BBB':
  'callback':
    <function make_callback.<locals>.func at 0x7f47f93771f0>
  '-c':
    {'help': 'A BBB [optional pi]', 'action': 'store_const', 'const': 3.141593}
  '-v|--verbose':
    {'help': 'A BBB verbosity', 'action': 'store_true'}
'AA':
  'callback':
    <function make_callback.<locals>.func at 0x7f47f9377280>
  '-c':
    {'help': 'AA [optional pi]', 'action': 'store_const', 'const': 3.141593}
  '-v|--verbose':
    {'help': 'AA verbosity', 'action': 'store_true'}
'AA|B':
  'callback':
    <function make_callback.<locals>.func at 0x7f47f9377310>
  '-c':
    {'help': 'AA B [optional pi]', 'action': 'store_const', 'const': 3.141593}
  '-v|--verbose':
    {'help': 'AA B verbosity', 'action': 'store_true'}
'AA|BB':
  'callback':
    <function make_callback.<locals>.func at 0x7f47f9377430>
  '-c':
    {'help': 'AA BB [optional pi]', 'action': 'store_const', 'const': 3.141593}
  '-v|--verbose':
    {'help': 'AA BB verbosity', 'action': 'store_true'}
'AA|BB|C':
  'callback':
    <function make_callback.<locals>.func at 0x7f47f93773a0>
  '-c':
    {'help': 'AA BB C [optional pi]', 'action': 'store_const', 'const': 3.141593}
  '-v|--verbose':
    {'help': 'AA BB C verbosity', 'action': 'store_true'}
'AA|BB|CC':
  'callback':
    <function make_callback.<locals>.func at 0x7f47f93775e0>
  '-c':
    {'help': 'AA BB CC [optional pi]', 'action': 'store_const', 'const': 3.141593}
  '-v|--verbose':
    {'help': 'AA BB CC verbosity', 'action': 'store_true'}
'AA|BB|CCC':
  'callback':
    <function make_callback.<locals>.func at 0x7f47f9377700>
  '-c':
    {'help': 'AA BB CCC [optional pi]', 'action': 'store_const', 'const': 3.141593}
  '-v|--verbose':
    {'help': 'AA BB CCC verbosity', 'action': 'store_true'}
'AA|BBB':
  'callback':
    <function make_callback.<locals>.func at 0x7f47f9377790>
  '-c':
    {'help': 'AA BBB [optional pi]', 'action': 'store_const', 'const': 3.141593}
  '-v|--verbose':
    {'help': 'AA BBB verbosity', 'action': 'store_true'}

```

and then the tree that the parsed dict is represented by:

```

1 print(
2     ParseArg(d=view, root_name='TODO').tree.show(quiet=True)
3 )

```

```

TODO
  AA
    B
    BB
      C
      CCC
      CC
    BBB
  A

```

```

B
BB
  C
  CCC
  CC
BBB
    
```

We can now run the A-tree and the A-AA-tree examples, respectively:

```

1 from a import main
2 main()
    
```

```

NODE :: 'A':
-----
usage: ipython A [-h] [-c] [-v] {B,BB,BBB} ...

positional arguments:
  {B,BB,BBB}

optional arguments:
  -h, --help      show this help message and exit
  -c              A [optional pi]
  -v, --verbose   A verbosity

'A':
-----
args: {'c': None, 'verbose': False}
<Mock name='mock.A' id='139947101347168'>
'A -v':
-----
args: {'c': None, 'verbose': True}
<Mock name='mock.A' id='139947101347168'>
'A -c':
-----
args: {'c': 3.141593, 'verbose': False}
<Mock name='mock.A' id='139947101347168'>
'A -v -c':
-----
args: {'c': 3.141593, 'verbose': True}
<Mock name='mock.A' id='139947101347168'>

NODE :: 'A B':
-----
usage: ipython A B [-h] [-c] [-v]

optional arguments:
  -h, --help      show this help message and exit
  -c              A B [optional pi]
  -v, --verbose   A B verbosity

'A B':
-----
args: {'c': None, 'verbose': False}
<Mock name='mock.A_B' id='139947101462384'>
'A B -v':
-----
args: {'c': None, 'verbose': True}
<Mock name='mock.A_B' id='139947101462384'>
'A B -c':
-----
args: {'c': 3.141593, 'verbose': False}
<Mock name='mock.A_B' id='139947101462384'>
'A B -v -c':
-----
args: {'c': 3.141593, 'verbose': True}
<Mock name='mock.A_B' id='139947101462384'>

NODE :: 'A BB':
-----
    
```

```

usage: ipython A BB [-h] [-c] [-v] {C,CCC,CC} ...

positional arguments:
  {C,CCC,CC}

optional arguments:
  -h, --help            show this help message and exit
  -c                    A BB [optional pi]
  -v, --verbose         A BB erbosity

'A BB':
-----
args: {'c': None, 'verbose': False}
<Mock name='mock.A_BB' id='139947100865632'>
'A BB -v':
-----
args: {'c': None, 'verbose': True}
<Mock name='mock.A_BB' id='139947100865632'>
'A BB -c':
-----
args: {'c': 3.141593, 'verbose': False}
<Mock name='mock.A_BB' id='139947100865632'>
'A BB -v -c':
-----
args: {'c': 3.141593, 'verbose': True}
<Mock name='mock.A_BB' id='139947100865632'>

NODE :: 'A BB C':
-----
usage: ipython A BB C [-h] [-c] [-v]

optional arguments:
  -h, --help            show this help message and exit
  -c                    A BB C [optional pi]
  -v, --verbose         A BB C verbosity

'A BB C':
-----
args: {'c': None, 'verbose': False}
<Mock name='mock.A_BB_C' id='139947100922976'>
'A BB C -v':
-----
args: {'c': None, 'verbose': True}
<Mock name='mock.A_BB_C' id='139947100922976'>
'A BB C -c':
-----
args: {'c': 3.141593, 'verbose': False}
<Mock name='mock.A_BB_C' id='139947100922976'>
'A BB C -v -c':
-----
args: {'c': 3.141593, 'verbose': True}
<Mock name='mock.A_BB_C' id='139947100922976'>

NODE :: 'A BB CC':
-----
usage: ipython A BB CC [-h] [-c] [-v]

optional arguments:
  -h, --help            show this help message and exit
  -c                    A BB CC [optional pi]
  -v, --verbose         A BB CC verbosity

'A BB CC':
-----
args: {'c': None, 'verbose': False}
<Mock name='mock.A_BB_CC' id='139947100924128'>
'A BB CC -v':
-----
args: {'c': None, 'verbose': True}
<Mock name='mock.A_BB_CC' id='139947100924128'>
'A BB CC -c':
-----

```

```

    args: {'c': 3.141593, 'verbose': False}
    <Mock name='mock.A_BB_CC' id='139947100924128'>
'A BB CC -v -c':
-----
    args: {'c': 3.141593, 'verbose': True}
    <Mock name='mock.A_BB_CC' id='139947100924128'>

NODE :: 'A BB CCC':
-----
usage: ipython A BB CCC [-h] [-c] [-v]

optional arguments:
  -h, --help      show this help message and exit
  -c              A BB CCC [optional pi]
  -v, --verbose   A BB CCC verbosity

'A BB CCC':
-----
    args: {'c': None, 'verbose': False}
    <Mock name='mock.A_BB_CCC' id='139947100925424'>
'A BB CCC -v':
-----
    args: {'c': None, 'verbose': True}
    <Mock name='mock.A_BB_CCC' id='139947100925424'>
'A BB CCC -c':
-----
    args: {'c': 3.141593, 'verbose': False}
    <Mock name='mock.A_BB_CCC' id='139947100925424'>
'A BB CCC -v -c':
-----
    args: {'c': 3.141593, 'verbose': True}
    <Mock name='mock.A_BB_CCC' id='139947100925424'>

```

```

1 from a_aa import main
2 main()

```

```

NODE :: 'A':
-----
usage: ipython A [-h] [-c] [-v] {B,BB,BBB} ...

positional arguments:
  {B,BB,BBB}

optional arguments:
  -h, --help      show this help message and exit
  -c              A [optional pi]
  -v, --verbose   A verbosity

'A':
----
    args: {'c': None, 'verbose': False}
    <Mock name='mock.A' id='139947112579024'>
'A -v':
-----
    args: {'c': None, 'verbose': True}
    <Mock name='mock.A' id='139947112579024'>
'A -c':
-----
    args: {'c': 3.141593, 'verbose': False}
    <Mock name='mock.A' id='139947112579024'>
'A -v -c':
-----
    args: {'c': 3.141593, 'verbose': True}
    <Mock name='mock.A' id='139947112579024'>

NODE :: 'A B':
-----
usage: ipython A B [-h] [-c] [-v]

```

```

optional arguments:
-h, --help      show this help message and exit
-c              A B [optional pi]
-v, --verbose  A B verbosity

'A B':
-----
args: {'c': None, 'verbose': False}
<Mock name='mock.A_B' id='139947100555488'>
'A B -v':
-----
args: {'c': None, 'verbose': True}
<Mock name='mock.A_B' id='139947100555488'>
'A B -c':
-----
args: {'c': 3.141593, 'verbose': False}
<Mock name='mock.A_B' id='139947100555488'>
'A B -v -c':
-----
args: {'c': 3.141593, 'verbose': True}
<Mock name='mock.A_B' id='139947100555488'>

NODE :: 'A BB':
-----
usage: ipython A BB [-h] [-c] [-v] {C,CCC,CC} ...

positional arguments:
{C,CCC,CC}

optional arguments:
-h, --help      show this help message and exit
-c              A BB [optional pi]
-v, --verbose  A BB verbosity

'A BB':
-----
args: {'c': None, 'verbose': False}
<Mock name='mock.A_BB' id='139947100557120'>
'A BB -v':
-----
args: {'c': None, 'verbose': True}
<Mock name='mock.A_BB' id='139947100557120'>
'A BB -c':
-----
args: {'c': 3.141593, 'verbose': False}
<Mock name='mock.A_BB' id='139947100557120'>
'A BB -v -c':
-----
args: {'c': 3.141593, 'verbose': True}
<Mock name='mock.A_BB' id='139947100557120'>

NODE :: 'A BB C':
-----
usage: ipython A BB C [-h] [-c] [-v]

optional arguments:
-h, --help      show this help message and exit
-c              A BB C [optional pi]
-v, --verbose  A BB C verbosity

'A BB C':
-----
args: {'c': None, 'verbose': False}
<Mock name='mock.A_BB_C' id='139947100555200'>
'A BB C -v':
-----
args: {'c': None, 'verbose': True}
<Mock name='mock.A_BB_C' id='139947100555200'>
'A BB C -c':
-----
args: {'c': 3.141593, 'verbose': False}
<Mock name='mock.A_BB_C' id='139947100555200'>

```

```

'A BB C -v -c':
-----
  args: {'c': 3.141593, 'verbose': True}
  <Mock name='mock.A_BB_C' id='139947100555200'>

NODE :: 'A BB CC':
-----
usage: ipython A BB CC [-h] [-c] [-v]

optional arguments:
  -h, --help      show this help message and exit
  -c              A BB CC [optional pi]
  -v, --verbose   A BB CC verbosity

'A BB CC':
-----
  args: {'c': None, 'verbose': False}
  <Mock name='mock.A_BB_CC' id='139947100553856'>
'A BB CC -v':
-----
  args: {'c': None, 'verbose': True}
  <Mock name='mock.A_BB_CC' id='139947100553856'>
'A BB CC -c':
-----
  args: {'c': 3.141593, 'verbose': False}
  <Mock name='mock.A_BB_CC' id='139947100553856'>
'A BB CC -v -c':
-----
  args: {'c': 3.141593, 'verbose': True}
  <Mock name='mock.A_BB_CC' id='139947100553856'>

NODE :: 'A BB CCC':
-----
usage: ipython A BB CCC [-h] [-c] [-v]

optional arguments:
  -h, --help      show this help message and exit
  -c              A BB CCC [optional pi]
  -v, --verbose   A BB CCC verbosity

'A BB CCC':
-----
  args: {'c': None, 'verbose': False}
  <Mock name='mock.A_BB_CCC' id='139947100553424'>
'A BB CCC -v':
-----
  args: {'c': None, 'verbose': True}
  <Mock name='mock.A_BB_CCC' id='139947100553424'>
'A BB CCC -c':
-----
  args: {'c': 3.141593, 'verbose': False}
  <Mock name='mock.A_BB_CCC' id='139947100553424'>
'A BB CCC -v -c':
-----
  args: {'c': 3.141593, 'verbose': True}
  <Mock name='mock.A_BB_CCC' id='139947100553424'>

NODE :: 'AA':
-----
usage: ipython AA [-h] [-c] [-v] {B,BB,BBB} ...

positional arguments:
  {B,BB,BBB}

optional arguments:
  -h, --help      show this help message and exit
  -c              AA [optional pi]
  -v, --verbose   AA verbosity

'AA':
-----

```

```

        args: {'c': None, 'verbose': False}
        <Mock name='mock.AA' id='139947100554336'>
'AA -v':
-----
        args: {'c': None, 'verbose': True}
        <Mock name='mock.AA' id='139947100554336'>
'AA -c':
-----
        args: {'c': 3.141593, 'verbose': False}
        <Mock name='mock.AA' id='139947100554336'>
'AA -v -c':
-----
        args: {'c': 3.141593, 'verbose': True}
        <Mock name='mock.AA' id='139947100554336'>

NODE :: 'AA B':
-----
usage: ipython AA B [-h] [-c] [-v]

optional arguments:
  -h, --help            show this help message and exit
  -c                    AA B [optional pi]
  -v, --verbose         AA B verbosity

'AA B':
-----
        args: {'c': None, 'verbose': False}
        <Mock name='mock.AA_B' id='139947100554384'>
'AA B -v':
-----
        args: {'c': None, 'verbose': True}
        <Mock name='mock.AA_B' id='139947100554384'>
'AA B -c':
-----
        args: {'c': 3.141593, 'verbose': False}
        <Mock name='mock.AA_B' id='139947100554384'>
'AA B -v -c':
-----
        args: {'c': 3.141593, 'verbose': True}
        <Mock name='mock.AA_B' id='139947100554384'>

NODE :: 'AA BB':
-----
usage: ipython AA BB [-h] [-c] [-v] {C,CCC,CC} ...

positional arguments:
  {C,CCC,CC}

optional arguments:
  -h, --help            show this help message and exit
  -c                    AA BB [optional pi]
  -v, --verbose         AA BB verbosity

'AA BB':
-----
        args: {'c': None, 'verbose': False}
        <Mock name='mock.AA_BB' id='139947100555776'>
'AA BB -v':
-----
        args: {'c': None, 'verbose': True}
        <Mock name='mock.AA_BB' id='139947100555776'>
'AA BB -c':
-----
        args: {'c': 3.141593, 'verbose': False}
        <Mock name='mock.AA_BB' id='139947100555776'>
'AA BB -v -c':
-----
        args: {'c': 3.141593, 'verbose': True}
        <Mock name='mock.AA_BB' id='139947100555776'>

NODE :: 'AA BB C':
-----

```



```

usage: ipython AA BB C [-h] [-c] [-v]

optional arguments:
  -h, --help      show this help message and exit
  -c              AA BB C [optional pi]
  -v, --verbose   AA BB C verbosity

'AA BB C':
-----
args: {'c': None, 'verbose': False}
<Mock name='mock.AA_BB_C' id='139947100556592'>
'AA BB C -v':
-----
args: {'c': None, 'verbose': True}
<Mock name='mock.AA_BB_C' id='139947100556592'>
'AA BB C -c':
-----
args: {'c': 3.141593, 'verbose': False}
<Mock name='mock.AA_BB_C' id='139947100556592'>
'AA BB C -v -c':
-----
args: {'c': 3.141593, 'verbose': True}
<Mock name='mock.AA_BB_C' id='139947100556592'>

NODE :: 'AA BB CC':
-----
usage: ipython AA BB CC [-h] [-c] [-v]

optional arguments:
  -h, --help      show this help message and exit
  -c              AA BB CC [optional pi]
  -v, --verbose   AA BB CC verbosity

'AA BB CC':
-----
args: {'c': None, 'verbose': False}
<Mock name='mock.AA_BB_CC' id='139947100557024'>
'AA BB CC -v':
-----
args: {'c': None, 'verbose': True}
<Mock name='mock.AA_BB_CC' id='139947100557024'>
'AA BB CC -c':
-----
args: {'c': 3.141593, 'verbose': False}
<Mock name='mock.AA_BB_CC' id='139947100557024'>
'AA BB CC -v -c':
-----
args: {'c': 3.141593, 'verbose': True}
<Mock name='mock.AA_BB_CC' id='139947100557024'>

NODE :: 'AA BB CCC':
-----
usage: ipython AA BB CCC [-h] [-c] [-v]

optional arguments:
  -h, --help      show this help message and exit
  -c              AA BB CCC [optional pi]
  -v, --verbose   AA BB CCC verbosity

'AA BB CCC':
-----
args: {'c': None, 'verbose': False}
<Mock name='mock.AA_BB_CCC' id='139947100556064'>
'AA BB CCC -v':
-----
args: {'c': None, 'verbose': True}
<Mock name='mock.AA_BB_CCC' id='139947100556064'>
'AA BB CCC -c':
-----
args: {'c': 3.141593, 'verbose': False}
<Mock name='mock.AA_BB_CCC' id='139947100556064'>
'AA BB CCC -v -c':
-----

```

```
-----  
args: {'c': 3.141593, 'verbose': True}  
<Mock name='mock.AA_BB_CCC' id='139947100556064'>
```