Using help

Help in aggregate with decl_help, the help method, and parse error reports

Stephen J. Mildenhall

2026-11-01

The package `aggregate` offers help at three levels: for the DecL language, for an object, and for a mistake. The function `decl_help()`[1] returns a page of DecL help as HTML. The method `.help(regex)`[2] lists the public methods and properties of an object whose names match a regular expression. A DecL parse error comes with a structured report[3]: where the error is, what would have been accepted there, a "did you mean" suggestion, and, for a known trap, a hint with the rule that was broken.

```
decl_help()                    # DecL at a glance: one agg taken apart slot by slot
decl_help('card')              # DecL on one page: a ten box reference card
a.help('^est_')                # public names of a that match a regex
a.help('^q$', lod='short')     # with the first lines of each docstring
e.report                       # the ErrorReport on a parse error e
```

```
import re, html
from aggregate import build, qd, decl_help, DECL_HELP_FORMS, __version__
from style import wprint, GT, refuse, figures
figures()
print(__version__)
```

```
1.60.0
```

DecL help pages

The DecL help pages are listed in `DECL_HELP_FORMS`. The first, `anatomy`, is the default and shows one `agg` statement slot by slot; the second, `card`, is a reference card of ten boxes. A page is a `str` containing one self-contained HTML `<section>`. As the last expression of a notebook cell a page is rendered as HTML, through its `_repr_html_`, and `IPython.display.HTML(page)` renders it too. Elsewhere its repr shows the form, the version, and the size, rather than the HTML.

```
print(DECL_HELP_FORMS)
page = decl_help()
page
```

```
('anatomy', 'card')
```

[1] From aggregate 1.2.0
[2] From aggregate 1.0
[3] From aggregate 1.0

`agg Book`	label	keyword, then a label you choose
`250 claims` `1000 xs 0`	exposure	**volume**: amount of business optional **coverage**: limit xs deductible
`sev lognorm 100 cv 1.5`	severity	severity curve: family, mean, cv
`occurrence net of 250 xs 750`	occurrence reinsurance	optional, before frequency
`poisson`	frequency	claim count distribution
`aggregate net of 5000 xs 30000`	aggregate reinsurance	optional, after frequency
`note{main book}`	trailer	optional note
`hints{bs=1;log2=16;}`	trailer	optional update hints

Exposure volume	**coverage (optional)**	**Severity**	**Frequency**	**Reinsurance**

`250 claims` expected count
`25000 loss` expected loss
`40000 premium at 65% lr`
`1000 exposure at 25 rate`
units × rate
	`1000 xs 0` limit xs deductible
	omit for unlimited
	`[500 1000] xs [0 0]` a mix
		`sev lognorm 100 cv 1.5` mean, cv
		`sev gamma 100 cv 1` any scipy.stats
		`sev 50 * invgamma 2.5` scale, shape
		`sev ... !` unconditional
			`poisson`
			`mixed gamma 0.3` mixing cv 0.3
			`fixed` exactly the count
				`occurrence net of 75% po 600 xs 400` partial placement
				`aggregate ceded to 250 xs 250 and 500 xs 500` limit xs attachment, lowest first
				`aggregate net of 66% po inf xs 0 rate 100% cede 30%` 66% quota share w 30% ceding commission

less common				
`[100 200 50] claims` vectorized exposure `dfreq [1 2 3] [.5 .3 .2]` listed counts and probabilities (optional, default equally likely)	`4000 xs 1000` an excess layer	`dsev [1 2 10] [.4 .4 .2]` listed `sev 40 * expon + 10` scale and shift `picks [250 1000] [20000 4500]` adjust expected loss by layer	`negbin 2` variance multiple `binomial 0.5`	`50% po 500 xs 500` partial placement `tower [0 250 500 1000]` program breakpoints `rol 5% deposit 600 reinstatements 1 at 100%`

A help page is an ordinary string, so the usual string tools work on it. The regular expression below

extracts the titles of the ten boxes on the card.

```
card = decl_help('card')
print(repr(card))
wprint(re.findall(r'<h3[^>]*>(.*?)</h3>', card))
```

```
DeclHelp('card', aggregate 1.60.0, 23 kB)
['1 · An aggregate', '2 · Exposure', '3 · Severity', '4 · Frequency', '5 ·
Reinsurance', '6 · Reinsurance terms', '7 · A portfolio', '8 · A profit and
loss', '9 · Writing rules', '10 · The library, and more']
```

Every program on a help page builds, and the library's tests check that it does. A program that a host, such as the aggregate Loss Lab, can load is stored in a `data-load` attribute. The first on the anatomy page is built below.

```
programs = [html.unescape(p) for p in re.findall(r'data-load="([^"]*)"', page)]
print(len(programs))
print(programs[0])
qd(build(programs[0]))
```

```
3
agg Book
  250 claims
  1000 xs 0
  sev lognorm 100 cv 1.5
  occurrence net of 250 xs 750
  poisson
  aggregate net of 5000 xs 30000
  note{main book}
  hints{bs=1;log2=16;}
Aggregate object: Book. Frequency distribution poisson. Severity
lognorm, [0, 1,000], bounded; 1,000 xs 0. Updated with bucket size 1 and
log2 = 16. Validation: reinsurance; subject not unreasonable; net not
unreasonable.

       Mean     SD       CV     Skew   P01 Median   P99
X
Freq    250 15.811 0.063246 0.063246
Sev  96.712  120.2   1.2428    2.926     4     55   693
Agg   24167   2408 0.099643 0.085218 18823  24107 30000
```

The help method

The help method `.help(regex)` is available on `Aggregate`, `Portfolio`, `Severity`, `Frequency`, `Distortion`, `PnL`, `Copula`, `BivariateAggregate`, `Bounds`, `GridDistribution`, and `Underwriter`. It lists the public names that match `regex`, using `re.search`, and gives each method with its signature. With no argument the regex is `'.*'`, which matches every public name. The code below counts the public names of a dice aggregate and then lists those that match `^est_`.

```
a = build('agg Dice dfreq [3] dsev [1:6]')
print(len([n for n in dir(a) if not n.startswith('_')]))
a.help('^est_')
```

```
175
Attribute: est_cv
Attribute: est_m
Attribute: est_sd
```

```
Attribute: est_sev_cv
Attribute: est_sev_m
Attribute: est_sev_sd
Attribute: est_sev_skew
Attribute: est_sev_var
Attribute: est_skew
Attribute: est_var
```

The level of detail argument `lod` sets how much of each docstring follows the name: `'terse'` (none, the default), `'short'` (the first four lines), or `'all'`. The short form for the quantile method `q` shows its signature and the first four lines of its docstring, which include the definition of the lower quantile.

```
a.help('^q$', lod='short')
```

```
Callable: q(p, kind='lower')
Return quantile function of density_df.p_total.

Definition 2.1 (Quantiles)
x(α) = qα(X) = inf{x ∈ R : P[X ≤ x] ≥ α} is the lower α-quantile of X
```

The `values` argument adds the value of each matching attribute and property. A method shows its name and signature and is never called, so a broad regex is safe. With `'short'` a DataFrame or Series is cut to five rows, and with `'all'` it is shown in full.

```
a.help('^est_(m|cv)$', values='all')
```

```
Attribute: est_cv
0.28171808490950573
Attribute: est_m
10.5
```

The module-level `build` is an `Underwriter`, and its help method gives a quick index to the recipe base: the methods that load, look up, fork, and preview recipes. The recipe base itself is described in Using recipes.

```
build.help('^(load|recipe|recipes|fork|preview)$')
```

```
Callable: fork(name=None)
Callable: load(request=None)
Callable: preview(program)
Callable: recipe(name, kind='')
Attribute: recipes
```

The `fmt` argument sets the output format. The default, `'auto'`, gives ANSI color under Jupyter and plain text elsewhere; `'text'` and `'ansi'` force one or the other, and `'html'` renders each docstring as Markdown in a notebook. Python's own `help(a.q)` prints the full docstring of a single method.

Parse error reports

A DecL program that does not parse raises a `ValueError`. Its message is a one-line summary, including any suggestion, and its `report` attribute is an `ErrorReport` containing the line and column, the offending token, the suggestions, and the tokens that would have been accepted. The underwriter also logs `report.render()` at level ERROR before raising, so the full display, with a caret under the error, appears on stderr above the output of the cell.

```
try:
    build('agg Typo 10 clams sev lognorm 10 cv 2 poisson')
except ValueError as e:
    report = e.report
    wprint(e)
print(report.line, report.column, report.got, report.suggestions)
wprint(report.expected)
```

```
DecL parse error at line 1, column 13:

  agg Typo 10 clams sev lognorm 10 cv 2 poisson
              ^^^^^

Unexpected 'clams'. Did you mean: claims? Expected: 'claims' (or 'claim'), '/', '**' or '^',
'exposure', '@', 'loss', 'premium' (or 'prem'), '*', 'years' (or 'year'; renewal horizon)

DecL parse error at line 1, column 13: Unexpected 'clams'. Did you mean: claims?
1 13 clams ['claims']
["'claims' (or 'claim')", "'/'", "'**' or '^'", "'exposure'", "'@'", "'loss'",
"'premium' (or 'prem')", "'*'", "'years' (or 'year'; renewal horizon)"]
```

Some parse errors have a token report that is accurate but unhelpful, and for a known trap of this kind the report adds a hint with the rule that was broken. In an expense clause `as` closes an expense group, so an `and` after the label cannot parse, and the hint includes both fixes. The hint is `report.hint`, and it is appended to the summary, so `str(e)` includes it.

```
program = ('xpnl E.X 10000 premium less agg B.A 6500 loss 1000 xs 0 '
           'sev lognorm 100 cv 2 mixed ig 0.25 less '
           '500 fixed expense as FE and 15% premium expense as Comm')
try:
    build(program)
except ValueError as e:
    wprint(e)
```

```
DecL parse error at line 1, column 121:

  ... mixed ig 0.25 less 500 fixed expense as FE and 15% premium expense as Comm
                                                  ^^^

Unexpected 'and'. Expected: 'exp', a hints clause (hints{...}), 'lpar', 'lsqb', a note clause
(note{...}), a number, 'peel' (walk reinsurance layers, not tiers), a tags clause (tags{...})
Hint: 'as' closes an expense group, so 'and' cannot follow the label. Drop the 'and' to leave the
groups separate (one expense leg each), or move the 'as' label after the last 'and'-joined term to
combine them into one leg.

DecL parse error at line 1, column 121: Unexpected 'and'. 'as' closes an expense
group, so 'and' cannot follow the label. Drop the 'and' to leave the groups
separate (one expense leg each), or move the 'as' label after the last
'and'-joined term to combine them into one leg.
```

Two errors the grammar cannot catch have the same report. A misspelled distribution name, such as `lognorml`, is a legal identifier to the grammar, so it is checked against the distribution families, with the closest names in `report.suggestions`. An unmatched bracket is reported at its position in the program.

```
for program in ['agg Typo 10 claims sev lognorml 10 cv 2 poisson',
                'agg Open 1 claim dsev [1 2 fixed']:
    try:
        build(program)
```

```
    except ValueError as e:
        wprint(e)
        print(e.report.line, e.report.column, e.report.got, e.report.suggestions)
```

```
DecL parse error at line 1, column 24:

  agg Typo 10 claims sev lognorml 10 cv 2 poisson
                         ^^^^^^^^

Unknown severity family 'lognorml'. Did you mean: lognorm, loguniform, powerlognorm?

DecL parse error at line 1, column 24: Unknown severity family 'lognorml'. Did
you mean: lognorm, loguniform, powerlognorm?
1 24 lognorml ['lognorm', 'loguniform', 'powerlognorm']
DecL parse error at line 1, column 23: Unmatched '['.
1 23 [ []
```

Refused help requests

An unknown help page, level of detail, or output format is refused. Each raises a `ValueError` that lists the accepted values.

```
refuse(lambda: decl_help('cheat'), lambda: a.help(lod='long'),
       lambda: a.help(fmt='pdf'))
```

```
Unknown DecL help form 'cheat'; the forms are 'anatomy', 'card'.
lod must be 'terse', 'short', or 'all'; got 'long'
fmt must be 'auto', 'text', 'ansi', or 'html'; got 'pdf'
```