monotable Documentation

Release 3.1.0

Mark Taylor

May 15, 2020

Contents

monotable version 3.1.0.

ASCII table with per column format specs, multi-line content, formatting directives, column width control.

- Licensed under Apache 2.0.
- Python versions 3.5-3.8, and pypy3 default versions supported by Travis-CI.
- No required dependencies.

Contents:

CHAPTER 1

Introduction, Installation

Monotable is a Python library that generates an ASCII table from tabular cell data that looks *pretty* in a monospaced font.

Monotable offers formatting *directives* to reduce messy table pre-formatting code. You can set directives for each column. You can also write and plug in your own format function directives.

Here is a list of some of the things Monotable does:

- Allows multi-line title, heading, and cell strings.
- Supports column oriented cell data.
- Generate a table with borders.
- Directives to limit column width and text wrap.
- Add horizontal and vertical rules.
- Join ASCII tables horizontally.
- Is *thoroughly* documented and tested.

1.1 Installation

```
pip install monotable
```

CHAPTER 2

Examples

2.1 Per column format specifications

Specify format string for each column.

```
from monotable import mono
headings = ['comma', 'percent']
formats = [',', '.1%']
cells = [[123456789, 0.33], [2345678, 0.995]]
print(mono(
    headings, formats, cells, title="',' and '%' formats."))
```

```
',' and '%' formats.
--------------------
      comma  percent
--------------------
123,456,789    33.0%
  2,345,678    99.5%
--------------------
```

- List **formats** supplies the format strings containing the formatting instructions. They are assigned to columns from left to right.
- Here the format strings are just format specifications.
- For each cell in the column, it and the format specification is passed to the built-in function **format(value, format_spec)**.
- To write a format_spec, consult Python's Format Specification Mini-Language.
- The cells are organized as a list of rows where each row is a list of cells.

2.2 zero and none format directives

Improve the appearance of zero values. Show the meaning of cell type None.

```
from datetime import datetime
from monotable import mono

headings = [
    'hour',
    '24 hour\ntemp\nchange',
    'wind\nspeed',
    'precip.\n(inches)'
    ]

formats = [
    '%H',
    '(zero=same)+.0f',
    '(zero=calm;none=offline).0f',
    '(zero=).2f',
    ]

h7 = datetime(2019, 2, 28, 7, 0, 0)
h8 = datetime(2019, 2, 28, 8, 0, 0)
h9 = datetime(2019, 2, 28, 9, 0, 0)

cells = [
    [h7, -2.3,   11, 3.4],
    [h8,  0.1,    0, 0.0],
    [h9,    5, None, 0.6734]
    ]

print(mono(
    headings, formats, cells, title='=Formatting directives.'))
```

```
     Formatting directives.
--------------------------------
      24 hour
         temp     wind   precip.
hour   change    speed  (inches)
--------------------------------
07         -2       11      3.40
08       same     calm
09         +5  offline      0.67
--------------------------------
```

- The `'%H'` format specification is passed by built-in function **format()** to datetime.__format__().
- The `'(zero=same)+.0f'` format string is split into two parts.
  - `(zero=same)` selects the zero directive with the value `same`.
  - `+.0f` is passed to the format function as format_spec.
- The zero format directive applies when the cell is a Number and the formatted text contains no non-zero digits. The characters after zero= are the formatted text for the cell.
- Format directives are enclosed by `(` and `)`.
- Separate multiple format directives with `;`.

- The none format directive formats the cell value None as the characters after none=.

2.3 parentheses format directive

Enclose negative numbers with parentheses. The 1's digit remains in the same column.

```
from monotable import mono

headings = ['Description', 'Amount']
formats = ['', '(zero=n/a;parentheses),']

cells = [
    ['receivables', 51],
    ['other assets', 9050],
    ['gifts', 0],
    ['pending payments',  -75],
    ['other liabilities', -623]
    ]

print(mono(
    headings, formats, cells, title='parentheses directive.'))
```

```
  parentheses directive.
-------------------------
Description        Amount
-------------------------
receivables            51
other assets        9,050
gifts                 n/a
pending payments      (75)
other liabilities    (623)
-------------------------
```

2.4 Format function directives

Format function directives select the format function used for the column. These are useful for scaling numbers and showing truth values.

```
from monotable import mono

headings = [
    'units of\nthousands',
    'bool to\nyes/no'
    ]

formats = [
    '(thousands).1f',
    '(boolean)yes,no'
    ]

cells = [
    [35200, True],
    [1660, False]
```

(continues on next page)

(continued from previous page)

```
    ]

print(mono(
    headings, formats, cells, title='Format function directives.'))
```

```
Format function directives.
------------------
 units of  bool to
thousands   yes/no
------------------
     35.2      yes
      1.7       no
------------------
```

- Note the format function directives thousands and boolean.
- '(thousands)' divides the cell value by 1000.0 and then calls **format**().
- '(boolean)yes,no' formats the cells that test True as 'yes' and False as 'no'.
- You can substitute any text you want for 'yes,no' for example 'on,off'.
- You can also write and plug in an unlimited number of custom format function directives.
- monotable's format function directives are implemented in the file plugin.py.

2.5 Column oriented input with vertical rule column

```
from monotable import monocol, VR_COL

column0 = ('award', '', ['nominated', 'won'])
column1 = ('bool to\nyes/no', '(boolean)yes,no', [True, False])
columns = [column0, VR_COL, column1]

print(monocol(columns,
    title='Columns with\nvertical rule.'))
```

```
    Columns with
   vertical rule.
-------------------
          | bool to
award     |  yes/no
-------------------
nominated |     yes
won       |      no
-------------------
```

- VR_COL in the second column renders the vertical bars.
- The title is center aligned.

2.6 Horizontal and vertical rules in a row oriented table

The cell row **monotable.HR_ROW** will be replaced with a heading guideline.

The text between columns can be changed with the format directive lsep. lsep specifies the separator between this column and the left side neighbor column.

By default the column separator is two spaces. In this example lsep in the second column is changed to `' | '`. This creates an effect approximating a vertical rule.

The last row only has one element. **monotable** extends short heading, formats, and cell rows with the empty string value. Extra format directive strings are silently ignored.

```
from monotable import mono, HR_ROW

headings = ['col-0', 'col-1']
formats = ['', '(lsep= | )']

cells = [['time', '12:45'],
         ['place', 'home'],
         HR_ROW,               # put a heading guideline here
         ['sound', 'bell'],
         ['volume']]           # short row is extended with empty string

print(mono(headings, formats, cells))
```

```
--------------
col-0  | col-1
--------------
time   | 12:45
place  | home
--------------
sound  | bell
volume |
--------------
```

Documentation on Read the Docs

More Examples

CHAPTER 3

List of format directives

Read about all the format directive syntax in the full Documentation. Follow the Format directives link in the Quick Links section.

none=ccc render cell type None as characters ccc.

zero=ccc render numeric cell that formats to zero to characters ccc.

parentheses remove minus sign and enclose negative cell value in parentheses.

lsep=ccc Characters ccc separate this column and the column to the left.

rsep=ccc Characters ccc separate this column and the column to the right.

width=N sets maximum width of column to N characters, content is truncated

width=N;wrap sets maximum width of column to N characters, content is text wrapped

width=N;fixed Pads or truncates content to N characters.

width=N;fixed;wrap Pads or text wraps content to N characters.

CHAPTER 4

List of format function directives

boolean test cell truth value and substitute caller's strings for True, False. The format_spec is ttt,fff where characters ttt are rendered for True and the characters fff are rendered for False.

function-name selects user defined function function-name. User can plug in an unlimited number of format functions.

thousands millions billions trillions divide cell value by 1000.0 (1.0e6, 1.0e9, 1,0e12).

milli micro nano pico multiply cell value by 1000.0 (1.0e6, 1.0e9, 1,0e12).

kibi mebi gibi tebi divide cell value by 1024. (1024**2, 1024**3, 1024**4).

mformat format cells that are mappings by selecting keys with the format_spec.

pformat cell is formatted by python printf-style percent operator '%'.

sformat format cell with str.format().

tformat format cell using string.Template.substitute().

CHAPTER 5

Auto-alignment and how to override it

Monotable auto-aligns the title, headings, and each column.

Auto-alignment is overridden by using one of `'<'`, `'^'`, `'>'` prefixes on a heading string, format directive string, or title.

Read more about auto-alignment in “Quick Links” section in the full Documentation. Follow the link *Auto-alignment*.

CHAPTER 6

Links to License, Docs, Repos, Issues, PYPI page

- License: Apache 2.0
- Full Documentation on Read the Docs
- Repository
- Issue Tracker
- Python Package Index/monotable
- Master branch build status, coverage, testing

CHAPTER 7

What monotable does not do

- Produce terminal graphics characters. Try PYPI terminaltables.
- Handle CJK wide characters.
- Handle ANSI escape terminal color sequences. Try PYPI terminaltables.
- Produce arbitrary markup source text. Try PYPI tabulate instead. However calling mono() or monocol() with keyword argument bordered=True produces valid reStructuredText grid table and simple table markup is possible.

Monotable does make the output of its formatting and alignment engine available in list form. Please look for the function **MonoTable.row_strings()** in the API documentation.

CHAPTER 8

Recent Changes

3.1.0 - 2020-05-15

- Add py.typed designation. Add to setup() zip_safe=False.

3.0.1 - 2020-05-10

- Remove Python 2.7 compatibility.
- Move typing comments into function annotations. Rework typing.
- Add monotable.join_strings().

2.1.0 - 2019-02-25

- Add module level convenience functions mono(), monocol() and constants HR_ROW, VR_COL.
- Add formatting directives none, zero, parentheses, lsep, and rsep.
- Reorder/rework docs examples and other sections.
- Change what (boolean) prints when malformed format spec.
- Drop Python 3.3 and 3.4 classifiers. Drop Python 3.4 tests from Travis CI.

2.0.1 - 2018-05-12

- Bugfix- MonoTableCellError on str below float in a column.
- Bugfix- Incorrect format spec reported in MonoTableCellError.

2.0.0 - 2017-06-16

- Changed the API: headings and formats parameters are now passed to table(), bordered_table().
- Added to class MonoTable 2 member functions that take table data organized as columns.
- Added convenience functions to module monotable.table. They call class MonoTable public member functions.
- Added 13 new plugin format functions and the corresponding format options: boolean, thousands, millions, billions, trillions, milli, micro, nano, pico, kibi, mebi, gibi, tebi.
- Removed 'from MonoTable import' statements from __init__.py.

1.0.2 - 2017-04-06

- Bug fix, incorrect cell auto-alignment when mixed types in a column.
- Bug fix, format_none_as cell ignoring column format string's align_spec.
- Remove and re-add files to git index so stored with LFs.
- Add complexity inspections to CI.
- Refactor 2 functions to reduce McCabe complexity.
- Code inspection fixes. Docs and comments fixed.

1.0.1 - 2017-03-26

- MANIFEST.in and doc fixes.

More . . .

If you are not already there, please continue reading More Examples in the Documentation on Read the Docs.

CHAPTER 9

Contributing and Developing

Please see Contributing.

CHAPTER 10

More Examples

10.1 Join tables together side by side

The function **monotable.join_strings()** lays out multi-line strings side by side. In this example table1, table2, and table3 are multi-line strings representing rendered ASCII tables. Note how the vertical alignment within each table is maintained in the result.

```
from monotable import join_strings

table1 = '\n'.join([
    "',' and '%' formats.",
    "--------------------",
    "      comma  percent",
    "--------------------",
    "123,456,789    33.0%",
    "  2,345,678    99.5%",
    "--------------------"
])

table2 = '\n'.join([
    "     Formatting directives.",
    "---------------------------------",
    "      24 hour",
    "         temp     wind   precip.",
    "hour   change    speed  (inches)",
    "---------------------------------",
    "07         -2       11      3.40",
    "08       same     calm",
    "09         +5  offline      0.67",
    "---------------------------------"
])

table3 = '\n'.join([
    "  parentheses directive.",
```

(continues on next page)

(continued from previous page)

```
    "-------------------------",
    "Description        Amount",
    "-------------------------",
    "receivables           51",
    "other assets       9,050",
    "gifts                n/a",
    "pending payments     (75)",
    "other liabilities   (623)",
    "-------------------------",
])

tables = [table1, table2, table3]
print(join_strings(
    tables,
    rsep='   '))   # 3 spaces between each table
```

```
',' and '%' formats.        Formatting directives.          parentheses directive.
--------------------   --------------------------------   -------------------------
      comma  percent          24 hour                      Description        Amount
--------------------              temp     wind   precip.   -------------------------
123,456,789    33.0%   hour    change    speed  (inches)   receivables            51
  2,345,678    99.5%   --------------------------------   other assets        9,050
--------------------   07          -2       11      3.40   gifts                 n/a
                       08        same     calm             pending payments      (75)
                       09          +5  offline      0.67   other liabilities    (623)
                       --------------------------------   -------------------------
```

10.2 User defined format function

Set a user defined format function for the 3rd column.

The user defined format function directive is plugged in to the table by passing a mapping to **mono()** or **monocol()** as keyword only argument format_func_map. The mapping contains the name of the format function as the key and function object as the value.

The keys in the mapping become format directive function names.

```
from monotable import mono

# User defined format function.
def fulfill_menu_request(value, spec):
    _, _ = value, spec          # avoid unused variable nag
    return 'Spam!'              # ignore both args

my_functions = {'fulfill_menu_request': fulfill_menu_request}

headings = ['Id Number', 'Duties', 'Meal\nPreference']
formats = ['', '', '(fulfill_menu_request)']

cells = [[1, 'President and CEO', 'steak'],
         [2, 'Raise capital', 'eggs'],
         [3, 'Oversee day to day operations', 'toast']]

print(mono(
```

(continues on next page)

(continued from previous page)

```
    headings, formats, cells,
    title='>User defined format function.',
    format_func_map=my_functions))
```

```
                      User defined format function.
-------------------------------------------------
                                          Meal
Id Number  Duties                        Preference
-------------------------------------------------
        1  President and CEO              Spam!
        2  Raise capital                  Spam!
        3  Oversee day to day operations  Spam!
-------------------------------------------------
```

- The user defined format function **fulfill_menu_request()** ignores the arguments and returns the string 'Spam!'.
- Keys in the dictionary **my_functions** become directive names,
- The Duties column auto-aligns to the left since the cells are strings.
- The headings auto-align to the alignment of the cell in the first row.
- The title starts with an `'>'` align_spec_char which right aligns the title over the table.

10.3 Change or omit the guidelines

```
from monotable import mono

headings = ['purchased\nparrot\nheart rate', 'life\nstate']

# > is needed to right align None cell since it auto-aligns to left.
# monotable uses empty string to format the second column.
formats = ['>(none=rest).0f']
cells = [[0, 'demised'],
         [0.0, 'passed on'],
         [None, 'is no more'],
         [-1],
         [0, 'ceased to be']]

print(mono(
    headings, formats, cells,
    title='Complaint\n(registered)',

    # top guideline is equals, heading is period, bottom is omitted.
    guideline_chars='=. '))
```

```
       Complaint
      (registered)
========================
 purchased
    parrot  life
heart rate  state
........................
         0  demised
         0  passed on
```

(continues on next page)

(continued from previous page)

```
      rest  is no more
        -1
         0  ceased to be
```

10.4 Limit column width

Here we employ the format directive **(width=15)** to limit the width of the second column to 15 characters or less. The **more_marker** '...' shows where text was omitted.

The width=N format directive applies only to the cells, not the heading.

```
from monotable import mono

headings = ['Id Number', 'Duties', 'Start Date']
formats = ['', '(width=15)']
cells = [[1, 'President and CEO', '06/02/2016'],
         [2, 'Raise capital', '06/10/2016'],
         [3, 'Oversee day to day operations', '06/21/2016']]

print(mono(headings, formats, cells,
        title='Limit center column to 15 characters.'))
```

```
Limit center column to 15 characters.
--------------------------------------
Id Number  Duties           Start Date
--------------------------------------
        1  President an...  06/02/2016
        2  Raise capital    06/10/2016
        3  Oversee day ...  06/21/2016
--------------------------------------
```

10.5 Wrap a column and limit cell height

The second column is wrapped to a maximum width of 12 characters.

Here we customize an instance of class MonoTable in order to change the class variable max_cell_height. We call MonoTable's table() method.

```
from monotable import MonoTable

headings = ['Id Number', 'Duties', 'Start Date']
formats = ['', '(width=12;wrap)']
t3 = MonoTable()
t3.max_cell_height = 2              # override class var

cells = [[1, 'President and CEO', '06/02/2016'],
         [2, 'Raise capital', '06/10/2016'],
         [3, 'Oversee day to day operations', '06/21/2016']]

title = ('Wrap center column to a maximum of 12 characters.\n'
         'Limit cell height to 2 lines')
```

(continues on next page)

(continued from previous page)

```
print(t3.table(headings, formats, cells, title=title))
```

```
Wrap center column to a maximum of 12 characters.
Limit cell height to 2 lines
---------------------------------
Id Number  Duties       Start Date
---------------------------------
        1  President    06/02/2016
           and CEO
        2  Raise        06/10/2016
           capital
        3  Oversee day  06/21/2016
           to day  ...
---------------------------------
```

- Limiting the maximum cell height to 2 lines affects the Duties cell in the bottom row. The **more_marker** '...' is placed at the end of the cell to indicate text was omitted.
- The default **max_cell_height** is None which means unlimited.
- **max_cell_height** is applied to every cell in the table.
- Changing **max_cell_height** to 1 assures there will be no multi-line cells in the table.
- The second column ended up wrapping to 11 characters wide, one character less than the format directive (width=12;wrap) specified. This behaviour is a side affect of using Python textwrap to implement the format directive.

10.6 Fix column width

Add **;fixed** after **(width=11)** to fix the column width. The formatted text will be padded or truncated to the exact width.

fixed can also be used with **wrap** like this: **(width=N;fixed;wrap)**.

```
from monotable import mono

headings = ['left\ncol', 'mid\ncol', 'right\ncol']
formats = ['', '^(width=11;fixed)']
cells = [['A',   1, 'x'],
         ['B', 222, 'y'],
         ['C',   3, 'z']]

title = 'Middle column is fixed width.'

print(mono(headings, formats, cells, title=title))
```

```
Middle column is fixed width.
------------------------
left      mid      right
col       col      col
------------------------
A          1       x
B        222       y
```

(continues on next page)

(continued from previous page)

```
C          3       z
------------------------
```

- The align_spec_prefix '^' of the formats[1] center justifies the column.

10.7 Selecting keys from a dictionary and table borders

This example sets the format function of the second column. A format string has the form:

```
[align_spec][directives][format_spec]
```

align_spec is one of the characters '<', '^', '>' to override auto-alignment. align_spec is not used in this example.

directives is one or more format directives enclosed by `'('` and `')'` separated by `';'`. In the second column the directive is `(mformat)`. mformat selects the function **monotable.plugin.mformat()** as the format function.

This example also shows formatted cells with newlines.

```
from monotable import mono

headings = ['int', 'Formatted by mformat()']
formats = ['',
    '(mformat)name= {name}\nage= {age:.1f}\ncolor= {favorite_color}']
cells = [[2345, dict(name='Row Zero',
                     age=888.000,
                     favorite_color='blue')],

         [6789, dict(name='Row One',
                     age=999.111,
                     favorite_color='No! Red!')]]

print(mono(headings, formats, cells,
           title='mformat() Formatting.',
           bordered=True))
```

```
      mformat() Formatting.
+------+------------------------+
|  int | Formatted by mformat() |
+======+========================+
| 2345 | name= Row Zero         |
|      | age= 888.0             |
|      | color= blue            |
+------+------------------------+
| 6789 | name= Row One          |
|      | age= 999.1             |
|      | color= No! Red!        |
+------+------------------------+
```

10.8 Selecting attributes or elements

Here one attribute of a cell object is selected for formatting in the first column. The second column selects the element indexed by [1] from a sequence.

```
from monotable import mono

headings = ['x\nattrib.', '[1]\nindex']
formats = ['(sformat){.x}', '(sformat){[1]}']

class MyCell:
    def __init__(self, x, y):
        self.x = x
        self.y = y

cells = [[MyCell(1, 91), ['a', 'bb']],
         [MyCell(2, 92), ['c', 'dd']]]

print(mono(headings, formats, cells,
           title='<Select attribute/index.'))
```

```
Select attribute/index.
--------------
x        [1]
attrib.  index
--------------
1        bb
2        dd
--------------
```

- Set the format directive to '(sformat)' to select **monotable.plugin.sformat**() as the format function. It is an adapter to string.format().
- The format_spec `'{.x}'` selects the attribute named 'x' of the cell.
- The format_spec `'{[1]}'` selects the element at index 1 of the cell.
- Note that a cell passed to str.format() satisfies only the first replacement field of the Python Format String Syntax. You can only use one replacement field with the sformat format directive.
- Note that the first column auto-aligns to the left. This is because auto-align senses the cell type which is class MyCell. Only cells that inherit from numbers.Number are auto-aligned to the right. MyCell does not inherit from numbers.Number.
- You can override auto-alignment on the first column by adding the align_spec `'>'` at the start of the format string.
- Since the heading auto-aligns to the alignment of the cell in the first row, you can also override auto-alignment on the first heading to keep it left aligned.

```
# Continues previous example.
headings = ['<x\nattrib.', '[1]\nindex']
formats = ['>(sformat){.x}', '(sformat){[1]}']
print(mono(headings, formats, cells,
           title='<Select attribute/index.'))
```

```
Select attribute/index.
--------------
x        [1]
attrib.  index
--------------
      1  bb
      2  dd
--------------
```

10.9 Make a reStructuredText Simple Table

The **separated_guidelines** and **guideline_chars** class variables can be overridden to produce reStructuredText Simple Table markup.

```
from monotable import MonoTable

class SeparatedMonoTable(MonoTable):
    separated_guidelines = True
    guideline_chars = '==='

headings = ['directive name', 'format function', 'description']
t4 = SeparatedMonoTable()

cells = [['mformat', 'monotable.plugin.mformat', 'mapping with str.format()'],
         ['pformat', 'monotable.plugin.pformat', 'printf style'],
         ['sformat', 'monotable.plugin.sformat', 'str.format()'],
         ['tformat', 'monotable.plugin.tformat', 'string.Template()'],
         ['function-name', '--', 'user defined function']]

print(t4.table(headings, [], cells))
```

```
=============  ========================  ========================
directive name  format function           description
=============  ========================  ========================
mformat         monotable.plugin.mformat  mapping with str.format()
pformat         monotable.plugin.pformat  printf style
sformat         monotable.plugin.sformat  str.format()
tformat         monotable.plugin.tformat  string.Template()
function-name   --                        user defined function
=============  ========================  ========================
```

Which looks like this when rendered.

directive name	format function	description
mformat	monotable.plugin.mformat	mapping with str.format()
pformat	monotable.plugin.pformat	printf style
sformat	monotable.plugin.sformat	str.format()
tformat	monotable.plugin.tformat	string.Template()
function-name	–	user defined function

10.10 String template substitution

The format directive tformat is used to select keys from a dictionary. It is implemented by an adapter to Python standard library string.Template.substitute().

```
from monotable import MonoTable

headings = ['an\nint', 'Formatted by\nstr.Template()']
formats = ['', '(tformat)name= $name\nage= $age\ncolor= $favorite_color']
cells = [[2345,
          dict(name='Row Zero', age=888, favorite_color='blue')],
         [6789,
```

(continues on next page)

(continued from previous page)

```
          dict(name='Row One', age=999, favorite_color='No......')]]

print(mono(headings, formats, cells,
           title='A multi-line\nTitle.', bordered=True))
```

```
       A multi-line
          Title.
+------+-----------------+
|   an | Formatted by    |
|  int | str.Template()  |
+======+=================+
| 2345 | name= Row Zero  |
|      | age= 888        |
|      | color= blue     |
+------+-----------------+
| 6789 | name= Row One   |
|      | age= 999        |
|      | color= No...... |
+------+-----------------+
```

- The title auto-aligns to center justification.
- Title auto-alignment is overridden by placing an align_spec char at the beginning of the title string.

10.11 Tiled table of four tables

See **test_tile_four_tables_together()** near the bottom of pytest cases of examples.

CHAPTER 11

Quick Links

11.1 Format directives

Format directives

11.2 Auto-alignment rules

Auto-alignment

11.3 Format directive string syntax

Format directive string syntax

11.4 Title string syntax

Title string syntax

CHAPTER 12

API

monotable.HR_ROW Row containing a horizontal rule to use as a row in cellgrid.

monotable.VR_COL Vertical rule column for use as a column_tuple with monocol().

monotable.MonoTable Class to create an aligned and formatted text table from a grid of cells.

12.1 Functions

The convenience functions below are the quickest way to generate an Ascii table. They create and configure a temporary instance of the class MonoTable and call a member function.

`monotable.`**`mono`**(*headings: Iterable[str] = (), formats: Iterable[str] = (), cellgrid: Iterable[Iterable[Any]] = ((),), title: str = '', *, bordered: bool = False, format_func_map: Optional[Mapping[str, Callable[[Any, str], str]]] = None, guideline_chars: str = '—', indent: str = ''*) → str

Generate ASCII table from cellgrid.

Parameters

- **`headings`** – Iterable of strings for each column heading.
- **`formats`** – Iterable of format strings of the form `[align_spec][directives][format_spec]`. Please see *Format directive string syntax*.
- **`cellgrid`** – representing table cells.
- **`title`** – `[align_spec][wrap_spec]string`. Text to be aligned and printed above the text table. Please see *Title string syntax*.

Keyword Arguments

- **`bordered`** – True means generate table with ASCII cell border characters.
- **`format_func_map`** – Dictionary of format functions keyed by name. name, when used as a format directive in a format string, selects the corresponding function from the dictionary.

If a key is one of the included format directive function names like 'boolean', 'mformat', etc. the included format directive function is hidden.

This value overrides *format_func_map* in the MonoTable instance that generates the table.

- **guideline_chars** – String of 0 to 3 characters to specify top, heading, and bottom guideline appearance. This value overrides *guideline_chars* in the MonoTable instance that generates the table.
- **indent** – String added to the beginning of each line in the text table.

Returns The text table as a single string.

Raises MonoTableCellError

monotable.**monocol**(*column_tuples: Sequence[Tuple[str, str, Sequence[T_co]]] = (), title: str = '', *, bordered: bool = False, format_func_map: Optional[Mapping[str, Callable[[Any, str], str]]] = None, guideline_chars: str = '—', indent: str = ''*) → str

Generate ASCII table from column tuples.

Parameters

- **column_tuples** – List of tuple of (heading string, format string, iterable of cell objects).

  The heading string syntax is described here *table()* under the parameter headings. The column tuple has a single heading string.

  The format directive string syntax is described here *Format directive string syntax*. The column tuple has a single format string.

  Iterable of cell objects represent the cells in the column.

- **title** – [align_spec][wrap_spec]string. Text to be aligned and printed above the text table. Please see *Title string syntax*.

Keyword Arguments

- **bordered** – True means generate table with ASCII cell border characters.
- **format_func_map** – Dictionary of format functions keyed by name. name, when used as a format directive in a format string, selects the corresponding function from the dictionary.

  If a key is one of the included format directive function names like 'boolean', 'mformat', etc. the included format directive function is hidden.

  This value overrides *format_func_map* in the MonoTable instance that generates the table.

- **guideline_chars** – String of 0 to 3 characters to specify top, heading, and bottom guideline appearance. This value overrides *guideline_chars* in the MonoTable instance that generates the table.
- **indent** – String added to the beginning of each line in the text table.

Returns The text table as a single string.

Raises MonoTableCellError

monotable.**join_strings**(*multi_line_strings: List[str], *, title: str = '', rsep: str = ' ', valign: int = 10*) → str

Join side-by-side multi-line strings preserving vertical alignment.

Parameters **multi_line_strings** – List of strings.

Keyword Arguments

- **title** – Text to be aligned and printed above the joined strings. Please see *Title string syntax*.
- **rsep** – Text placed between each line of the multi-line strings on the right hand side. It is not applied to the right-most multi-line string.
- **valign** – Alignment used for vertical justification of multi-line strings when the number of lines in the strings differ. Callers should use one of TOP, CENTER_TOP, CENTER_BOTTOM, or BOTTOM defined in monotable.alignment.

12.2 Legacy Functions

Since v2.1.0 these functions are superseded by mono() and monocol() above. The details of the function parameters are provided in the class MonoTable docstrings located by following the links below.

monotable.table.**table**(*headings: Iterable[str] = ()*, *formats: Iterable[str] = ()*, *cellgrid: Iterable[Iterable[Any]] = ((),)*, *title: str = ''*) → str

Wrapper to *monotable.table.MonoTable.table()*.

monotable.table.**bordered_table**(*headings: Iterable[str] = ()*, *formats: Iterable[str] = ()*, *cellgrid: Iterable[Iterable[Any]] = ((),)*, *title: str = ''*) → str

Wrapper to *monotable.table.MonoTable.bordered_table()*.

monotable.table.**cotable**(*column_tuples: Sequence[Tuple[str, str, Sequence[T_co]]] = ()*, *title: str = ''*) → str

Wrapper to *monotable.table.MonoTable.cotable()*.

monotable.table.**cobordered_table**(*column_tuples: Sequence[Tuple[str, str, Sequence[T_co]]] = ()*, *title: str = ''*) → str

Wrapper to *monotable.table.MonoTable.cobordered_table()*.

Note: The prefix **co** in cotable and cobordered_table stands for column oriented.

CHAPTER 13

Subpackage API

The information here is useful for configuring and extending the MonoTable class. The most convenient way to use monotable is described by *API*.

13.1 Class MonoTable

Links:

__init__() table() bordered_table() row_strings() cotable() cobordered_table() Title string syntax, *Format directive string syntax*, *Format directives*, *Auto-alignment*

class monotable.table.**MonoTable**(*indent=''*)

Create an aligned and formatted text table from a grid of cells.

Call *table()* passing a sequence of heading strings, a sequence of format strings, a sequence of sequence of cells, and a title string. The return value is a string ready for printing.

Call *cotable()* passing a sequence of tuples of (heading, format, list of cells in the column); one tuple for each column; and a title string. The prefix co stands for column oriented.

Call *bordered_table()* with the same arguments as table() to format a table with character borders.

Call *cobordered_table()* with the same arguments as cotable() to format a table with character borders.

Call *row_strings()* passing a sequence of heading strings, a sequence of format strings, a sequence of sequence of cells to return a tuple of lists of formatted headings and list of list of cells.

Heading strings, title strings, and formatted cells may contain newlines.

Cells that inherit from numbers.Number are *Auto-Aligned* to the right and all other cell types are auto aligned to the left.

Definitions:

Align_spec One of the characters '<', '^', '>'. The characters indicate left, center, and right justification. To configure, override the class variable *align_spec_chars*.

Heading String [align_spec]string

Format String [align_spec][directives][format_spec]

Title String [align_spec][wrap_spec]string

- The format directive string syntax is described by the *table()* argument **formats** below.

Auto-alignment:

- Heading alignment is determined by this decision order:
  1. align_spec if present in the heading string.
  2. align_spec if present in the format string.
  3. Auto-Alignment for the cell in the first row.
- Cell alignment is determined by this decision order:
  1. align_spec if present in the format string.
  2. Auto-Alignment for the cell.
- The title is auto-aligned to center.

Note The align_spec prefix may be omitted, but is required if the rest of the string starts with one of the align_spec_chars. Or the user can put in an empty directives for example '()'.

Note align_spec scanning/parsing can be disabled by setting the class variable *align_spec_chars* to the empty string.

There is one format string for each column of cells.

Formatting, by default, is done by <built-in function format>.

- The user can specify a different default format function for the table by overriding the class variable *format_func*.
- In directives the user can specify a format function for a column which takes precedence over the table default format function. They are listed here: *Format directives*.
- Any user defined function in the dict assigned to the class variable *format_func_map* may be selected by putting its key as a formatting directive.

Here is the data flow through the formatting engine:

```
                     text              MonoBlock
cell -> format_func ---> (parentheses) ------> width control,
        format_spec      (zero=)          |    justification
                                          |      (width=)
cell is None ------------(none=)----------+      (max)
                                                 (wrap)
                                                 (fixed)

* format directives are shown enclosed by ().
* format_func may be selected by a format function directive.
* user can plug in new format function directives.
```

- If cell is None an empty string is formatted. Configure for a column by using the *none=* format directive in directives. Configure for the whole table by overriding class variable *format_none_as*.

- If a cell is type float, and format_spec is an empty string, and the format function is <built-in function format>, the cell is formatted using class variable *default_float_format_spec*.

directives can contain the format directives *lsep=* and *rsep=* which sets the separator string placed before/after the column.

If wrap_spec_char is present in table title the title is text wrapped to the width of the table. To change the wrap_spec_char or disable wrap_spec_char scanning, see the class variable *wrap_spec_char*. Title is auto-aligned to center by default.

A format error produces an exception object which identifies the offending cell. Format error handling is configurable by overriding class variable *format_exc_callback*.

All the class variables can be overridden in ether a subclass or on an instance. For the complete list please see section *MonoTable Class Variables*.

MonoTable.**__init__**(*indent=''*)

Parameters **indent** (*str*) – String added to the beginning of each line in the text table.

MonoTable.**table**(*headings: Iterable[str] = ()*, *formats: Iterable[str] = ()*, *cellgrid: Iterable[Iterable[Any]] = ((),)*, *title: str = ''*) → str

Format printable text table. It is pretty in monospaced font.

Parameters

- **headings** – Iterable of strings for each column heading.
- **formats** – Iterable of format strings of the form `[align_spec][directives][format_spec]`. *Format directive string syntax*
- **cellgrid** – representing table cells.
- **title** – `[align_spec][wrap_spec]string`. Text to be aligned and printed above the text table. *Title string syntax*

Returns The text table as a single string.

Raises *MonoTableCellError*

Here is an example of non-bordered text table:

```
Int, Float, String                          <-- title
------------------------------------        <-- top guideline
      Int        Float  String              <-- heading row
------------------------------------        <-- heading guideline
123456789     3.141593  Hello World         <-- cell row
        2  2718.281828  another string      <-- cell row
------------------------------------        <-- bottom guideline

         ^^           ^^
          |            |
       sep/rsep     sep/rsep
```

This example has 6 sep strings of 2 spaces each. The seps are placed between columns in the heading line and between the columns in each of the 2 cell rows.

Title string syntax:

`[align_spec][wrap_spec]string`

align_spec One of the characters '<', '^', '>' to override auto-alignment.

wrap_spec Character '=' to indicate the title should be text wrapped to the width of the table.

Format directive string syntax:

`[align_spec][directives][format_spec].`

align_spec One of the characters '<', '^', '>' to override auto-alignment.

directives One or more of format directives enclosed by '(' and ')' separated by ';'.

format_spec String passed to the format function.

Format directives:

The format string directives described here apply to all MonoTable methods and monotable convenience functions that take format strings.

- At most, one format function directive is allowed.
- Each directive is allowed once.
- Spacing before '=' is always ignored.
- Spacing after '=' is significant for none, zero, lsep, and rsep.

width=N Truncate each formatted cell text line(s) to width N and insert the class variable *more_marker* if text was omitted.

fixed Used with width=N formats text to exactly width = N columns.

wrap Used with width=N does textwrap of formatted cell to width = N.

lsep=ccc Characters after 'lsep=' are the column separator on the left hand side of the column. Use lsep with care because it will silently overwrite the rsep specified for the column to the left. It has no effect if specified on the left-most column.

rsep=ccc Characters after 'rsep=' are the column separator on the right hand side of the column. It has no effect if specified on the right-most column.

sep=ccc Same as rsep. sep is an alias for rsep since version 2.1.0. New code should use rsep=ccc since the meaning is more explicit.

Format function directives:

boolean Convert boolean cell truth value to one of two strings supplied by the format_spec. Implemented by *monotable.plugin.boolean()*

none=ccc Specifies the formatted text for None cell value.

zero=ccc For cells that are numbers, when all digits in formatted text are zero replace the formatted text with ccc. `0.00e00 -> ccc`.

parentheses For cells that are numbers, when formatted text starts with a minus sign, remove the minus sign and add enclosing parentheses. `-100.1 -> (100.1)`. Maintains alignment with numbers not enclosed by parentheses.

thousands Select format function that divides by 10.0e3 before applying the format_spec. *monotable.plugin.thousands()*

millions Select format function that divides by 10.0e6 before applying the format_spec. *monotable.plugin.millions()*

billions Select format function that divides by 10.0e9 before applying the format_spec. *monotable.plugin.billions()*

trillions Select format function that divides by 10.0e12 before applying the format_spec. *monotable.plugin.trillions()*

milli Select format function that multiplies by 10.0e3 before applying the format_spec. *monotable.plugin.milli()*

micro Select format function that multiplies by 10.0e6 before applying the format_spec. *monotable.plugin.micro()*

nano Select format function that multiplies by 10.0e9 tbefore applying the format_spec. *monotable.plugin.nano()*

pico Select format function that multiplies by 10.0e12 before applying the format_spec. *monotable.plugin.pico()*

kibi Select format function that divides by 1024 before applying the format_spec. *monotable.plugin.kibi()*

mebi Select format function that divides by 1024^2 before applying the format_spec. *monotable.plugin.mebi()*

gibi Select format function that divides by 1024^3 before applying the format_spec. *monotable.plugin.gibi()*

tebi Select format function that divides by 1024^4 before applying the format_spec. *monotable.plugin.tebi()*

mformat Select format function that selects values from a dictionary. *monotable.plugin.mformat()*

pformat Select format function adapter to percent operator %. *monotable.plugin.pformat()*

sformat Select format function adapter to str.format(). *monotable.plugin.sformat()*

tformat Select format function adapter to string.Template.substitute(). *monotable.plugin.tformat()*

function-name Select format function with key function-name in the dictionary supplied by class variable *format_func_map*.

MonoTable.**bordered_table**(*headings: Iterable[str] = ()*, *formats: Iterable[str] = ()*, *cellgrid: Iterable[Iterable[Any]] = ((),)*, *title: str = ''*) → str

Format printable text table with individual cell borders.

Parameters

- **headings** – Iterable of strings for each column heading.
- **formats** – Iterable of format strings of the form `[align_spec][directives][format_spec]`. *Format directive string syntax*
- **cellgrid** – representing table cells.
- **title** – `[align_spec][wrap_spec]string`. Text to be aligned and printed above the text table. *Title string syntax*

Returns The text table as a single string.

Raises *MonoTableCellError*

Here is an example of bordered text table:

```
+----------------------+------------------+  <-- top guideline
| format string        | format string    |  <-- heading row
| "%Y-%m-%d--%I:%M:%S" | "week-%U-day-%j" |
+======================+==================+  <-- heading guideline
```

(continues on next page)

(continued from previous page)

```
| 2016-01-10--07:35:18 | week-02-day-010  |  <-- cell row
+----------------------+------------------+  <-- bottom guideline
```

MonoTable.**row_strings**(*headings: Iterable[str] = ()*, *formats: Iterable[str] = ()*, *cellgrid: Iterable[Iterable[Any]] = (())*, *strip: bool = False*) → List[List[str]]

Format and justify table. Return rows of the strings.

Parameters

- **headings** – Iterable of strings for each column heading.
- **formats** – Iterable of format strings of the form `[align_spec][directives][format_spec]`. *Format directive string syntax*
- **cellgrid** – representing table cells.
- **strip** – If True remove leading and trailing spaces.

Returns First row is headings, following rows are formatted cell strings.

Return type list of rows of string

Raises *MonoTableCellError*

When strip=False, each heading and all cells in each column are justified and are the same length.

MonoTable.**cotable**(*column_tuples: Sequence[Tuple[str, str, Sequence[T_co]]] = ()*, *title: str = ''*) → str

Format printable text table from tuples describing columns.

Parameters

- **column_tuples** – List of tuple of (heading string, format string, iterable of cell objects).

  The heading string syntax is described here *table()* under the parameter headings. The column tuple has a single heading string.

  The format directive string syntax is described here *table()* under the parameter formats. The column tuple has a single format string.

  Iterable of cell objects represent the cells in the column.

- **title** – `[align_spec][wrap_spec]string`. Text to be aligned and printed above the text table. *Title string syntax*

Returns The text table as a single string.

Raises *MonoTableCellError*

MonoTable.**cobordered_table**(*column_tuples: Sequence[Tuple[str, str, Sequence[T_co]]] = ()*, *title: str = ''*) → str

Format printable bordered text table from tuples describing columns.

Parameters

- **column_tuples** – List of tuple of (heading string, format string, iterable of cell objects).

  The heading string syntax is described here *table()* under the parameter headings. The column tuple has a single heading string.

  The format directive string syntax is described here *table()* under the parameter formats. The column tuple has a single format string.

  Iterable of cell objects represent the cells in the column.

- **title** – [align_spec][wrap_spec]string. Text to be aligned and printed above the text table. *Title string syntax*

Returns The text table as a single string.

Raises *MonoTableCellError*

13.2 MonoTable Class Variables

Links to class variable below:
format_func *format_exc_callback*
default_float_format_spec *format_none_as*
sep *separated_guidelines* *guideline_chars*
format_func_map
more_marker *align_spec_chars* *wrap_spec_char*
option_spec_delimiters
heading_valign *cell_valign* *max_cell_height*
border_chars *hmargin* *vmargin*

MonoTable.**format_func = <built-in function format>**
User defined function with signature of <built-in function format>.

This function is selected for cell formatting except when a column format string directives specifies a format function.

These *Format Functions* can be used here.

When overriding in a subclass definition use the result of Python built in function staticmethod() like this:

```
>>> import monotable
>>> def your_user_defined_format_function(value, format_spec):
...     pass
>>> class SubclassMonoTable(monotable.MonoTable):
...     format_func = staticmethod(your_user_defined_format_function)
>>> tbl = SubclassMonoTable()
>>>
>>> # When overriding on an instance do not use staticmethod like this:
>>>
>>> tbl = monotable.MonoTable()
>>> tbl.format_func = your_user_defined_format_function
```

Reading Python functions and methods Docs Here helps explain when to use built in function staticmethod().

MonoTable.**format_exc_callback = <function raise_it>**
Function called when format_func raises an exception.

The function takes the argument MonoTableCellError and (if returning) returns the string to be returned by format_func.

These *Format Function Error Callbacks* can be used here.

Please see advice at *format_func* about when to use staticmethod().

MonoTable.**default_float_format_spec = '.6f'**
Default format specification for float type.

Applies only to cells that satisfy all of:

- cell value is type float
- format function is <built-in function format>
- format_spec is the empty string

If the cell is not type float or a different format function is set, this will not apply.

This sets the precision to align the decimal points in a column of floats. This is useful when a column contains floats and strings. The presence of strings prevents the use of a float format_spec for the column.

Disable this feature by setting to the empty string. This feature applies to the entire table.

MonoTable.**format_none_as = ''**
Value placed in table for cell of type None.

MonoTable.**sep = ' '**
String that separates columns in non-bordered tables.

sep after a column can be overridden by the format string directive rsep.

MonoTable.**separated_guidelines = False**
When True guidelines will have the sep characters between columns.

This looks good when the seps are all spaces. When False the guideline character is repeated for the width of the table. seps refers to characters placed between heading and cell columns in the table.

MonoTable.**guideline_chars = '---'**
String of 0 to 3 characters to specify guideline appearance.

The first character is used for the top guideline. The second and third characters are used for the heading guideline and bottom guideline. If the character is a space the guideline is omitted. The empty string suppresses all guidelines.

MonoTable.**format_func_map = None**
Adds format functions selectable by a format directive.

Dictionary of format functions keyed by name. name, when used as a format directive in a format string, selects the corresponding function from the dictionary. If a key is one of the included format directive function names like 'boolean', 'mformat', etc. the included format directive function is hidden.

MonoTable.**more_marker = '...'**
Inserted to indicate text has been omitted.

MonoTable.**align_spec_chars = '<^>'**
Three characters to indicate justification.

Applies to align_spec scanning in heading, title, and format directive strings. The first character indicates left justification. The second and third characters indicate center and right justification. Setting align_spec_chars to "" disables align_spec scanning.

MonoTable.**wrap_spec_char = '='**
Single character to indicate the title should be text wrapped.

Wrapping is done to the width of the table. Setting wrap_spec_char to "" disables title text wrap.

MonoTable.**option_spec_delimiters = '(;)'**
Three characters to enclose and separate format directives.

The first and third chars enclose the options. The second char separates individual options. Setting option_spec_delimiters to "" disables format directive scanning in format strings.

MonoTable.**heading_valign = 13**
Alignment used for vertical justification of a multi-line heading.

Should be one of one of *Vertical Alignment Constants* TOP, CENTER_TOP, CENTER_BOTTOM, or BOTTOM.

`MonoTable.`**`cell_valign = 10`**
Alignment used for vertical justification of a multi-line cell.

Should be one of one of *Vertical Alignment Constants* TOP, CENTER_TOP, CENTER_BOTTOM, or BOTTOM.

`MonoTable.`**`max_cell_height = None`**
Truncates multi-line cells to this height. None means unlimited.

If characters are omitted, inserts **more_marker** at the end of the cell. Setting max_cell_height=1 will suppress multi-line cells.

`MonoTable.`**`border_chars = '--|+='`**
Characters used for borders in bordered tables.

One char each: top, bottom, sides, corner, heading guideline.

`MonoTable.`**`hmargin = 1`**
Number of blanks inserted on each side of text between borders.

Applies to bordered tables.

`MonoTable.`**`vmargin = 0`**
Number of blank lines inserted above and below formatted text.

Applies to bordered tables.

13.3 Format Functions

Links to format functions below:
`boolean()`
`sformat()` *`mformat()`* *`pformat()`* *`tformat()`*
`thousands()` *`millions()`* *`billions()`* *`trillions()`*
`milli()` *`micro()`* *`nano()`* *`pico()`*
`kibi()` *`mebi()`* *`gibi()`* *`tebi()`*

Format functions have the same signature as `<built-in function format>`.

These can be used to override the class variable *`format_func`* in a subclass or on an instance.

13.3.1 Boolean Values

`monotable.plugin.`**`boolean`**(*bool_value: bool*, *format_spec: str = 'T, F'*) → str
Format function that formats the boolean values to user's strings.

The format_spec is a string `'true-truth-value,false-truth-value'` of the true and false truth value strings joined by comma where true-truth-value is returned when bool_value evaluates to logical True. The default value for argument format_spec above is a good example. If fspec or !fspec is rendered check for an incorrect format_spec.

13.3.2 Python Formatting Function Adapters

The Python 3 documentation links below show how to write the format spec for the monotable's adapters to Python formatting functions.

function	Python 3 Documentation
sformat	Format String Syntax
mformat	same as sformat
tformat	Template Strings
pformat	printf-style

monotable.plugin.**sformat**(*value: Any*, *format_spec: str = ''*) → str
Format function adapter to str.format().

Please keep in mind that only a single replacement field can be used.

monotable.plugin.**mformat**(*mapping: Mapping[str, Any]*, *format_spec: str = ''*) → str
Format function that selects values from a dictionary.

In the format_spec use references to keyword arguments described by Python Standard Library Format String Syntax that are keys in mapping.

For d = dict(key1=value1, key2=value2, ...) A call to mformat(d, format_spec) behaves like calling: format_spec.format(key1=value1, key2=value2, ...).

Example:

```
>>> format_spec = '{key1:.2f} {key2:}!'
>>> print(format_spec.format(key1=25.9456, key2='spam'))
25.95 spam!
```

```
>>> from monotable.plugin import mformat
>>> format_spec = '{key1:.2f} {key2:}!'   # same as above
>>> d = {'key1': 25.9456, 'key2': 'spam'}
>>> print(mformat(d, format_spec))
25.95 spam!
```

The keys must be strings but need not be valid python identifiers as shown here with a key that begins with a digit and a key containing a '-'.

```
>>> from monotable.plugin import mformat
>>> format_spec = '{0key1:.2f} {key-2:}!'   # same as above
>>> d = {'0key1': 25.9456, 'key-2': 'spam'}
>>> print(mformat(d, format_spec))
25.95 spam!
```

monotable.plugin.**pformat**(*value: Any*, *format_spec: str = ''*) → str
Format function adapter to percent operator %.

The exact number % replacements in the printf-style format spec must be satisfied by items from value.

monotable.plugin.**tformat**(*value: Mapping[str, Any]*, *format_spec: str = ''*) → str
Format function adapter to string.Template.substitute().

13.3.3 Units Format Functions

These functions change the units of numeric values. They multiply or divide by a floating point number. The format_spec should be appropriate for type float.

monotable.plugin.**thousands**(*numeric_value: complex*, *format_spec: str = ''*) → str
Format function that divides by 1.0e3.

monotable.plugin.**millions**(*numeric_value: complex*, *format_spec: str = ''*) → str
Format function that divides by 1.0e6.

monotable.plugin.**billions**(*numeric_value: complex*, *format_spec: str = ''*) → str
Format function that divides by 1.0e9.

monotable.plugin.**trillions**(*numeric_value: complex*, *format_spec: str = ''*) → str
Format function that divides by 1.0e12.

monotable.plugin.**milli**(*numeric_value: complex*, *format_spec: str = ''*) → str
Format function that multiplies by 1.0e3.

monotable.plugin.**micro**(*numeric_value: complex*, *format_spec: str = ''*) → str
Format function that multiplies by 1.0e6.

monotable.plugin.**nano**(*numeric_value: complex*, *format_spec: str = ''*) → str
Format function that multiplies by 1.0e9.

monotable.plugin.**pico**(*numeric_value: complex*, *format_spec: str = ''*) → str
Format function that multiplies by 1.0e12.

monotable.plugin.**kibi**(*numeric_value: complex*, *format_spec: str = ''*) → str
Format function that divides by 1024.

monotable.plugin.**mebi**(*numeric_value: complex*, *format_spec: str = ''*) → str
Format function that divides by 1024^2.

monotable.plugin.**gibi**(*numeric_value: complex*, *format_spec: str = ''*) → str
Format function that divides by 1024^3.

monotable.plugin.**tebi**(*numeric_value: complex*, *format_spec: str = ''*) → str
Format function that divides by 1024^4.

13.3.4 References

Please refer to Wikipedia articles Unit_Prefix and Binary_Prefix.

13.4 Exception and Error Callbacks

13.4.1 Exception

exception monotable.table.**MonoTableCellError**(*row*, *column*, *format_spec=''*, *trace_text=None*)
Raised when format_func fails. Identifies the offending cell.

row
Cell grid row index of value causing format_func exception.

Type int

column
Cell grid column index of value causing format_func exception.

Type int

format_spec
Format_spec passed to format_func when exception occurred.

Type str

trace_text
Exception trace information for root cause of the exception.

Type str

name
Name of the exception shown in the string representation.

Type str

13.4.2 Format Function Error Callbacks

Format function error callbacks take an instance of monotable.table.MonoTableCellError.

These are used to override the class variable *format_exc_callback* in a subclass or on an instance.

monotable.plugin.**raise_it**(*cell_error_exception: monotable.cellerror.MonoTableCellError*) → None
Format function error callback. Exception is raised.

monotable.plugin.**print_it**(*cell_error_exception: monotable.cellerror.MonoTableCellError*) → str
Format function error callback. Prints exception. Returns '???'.

monotable.plugin.**ignore_it**(*_: monotable.cellerror.MonoTableCellError*) → str
Format function error callback. No action taken. Returns '???'.

13.5 HR, Vertical Align Constants

13.5.1 Horizontal Rule

monotable.table.**HR**
Placed in column 0 of a *row* in a cellgrid to insert a horizontal rule.

Since v2.1.0 monotable.HR_ROW can be used instead. A row that starts with a HR is omitted from the table and a heading guideline is inserted in its place.

13.5.2 Vertical Alignment Constants

Use these to specify a value for MonoTable class variable *heading_valign* or *cell_valign*

monotable.alignment.**TOP = 10**
Shift the text lines towards the top, add blank lines at the bottom.

monotable.alignment.**CENTER_TOP = 11**
Shift the text lines towards the top when odd number extra blank lines.

monotable.alignment.**CENTER_BOTTOM = 12**
Shift the text lines towards bottom when odd number extra blank lines.

monotable.alignment.**BOTTOM = 13**
Shift the text lines towards the bottom, add blank lines at the top.

CHAPTER 14

How to Configure MonoTable

14.1 Override class vars in subclass

Create a subclass of MonoTable and override one or more *MonoTable Class Variables*

```
import monotable

class SeparatedMonoTable(monotable.MonoTable):
   guideline_chars = '-=-'
   separated_guidelines = True

headings = ['an int', 'string', 'another int', 'another string']
tbl = SeparatedMonoTable()

cells = [[123, 'import', 4567, 'this']]

print(tbl.table(headings, [], cells, title='Subclass of MonoTable.'))
```

```
           Subclass of MonoTable.
------  ------  -----------  --------------
an int  string  another int  another string
======  ======  ===========  ==============
   123  import         4567  this
------  ------  -----------  --------------
```

14.2 Assign to class var names

Assign to one or more *MonoTable Class Variables* on an instance.

This creates an instance variable that overrides the class variable with an instance variable of the same name.

```
import monotable

headings = ['an int', 'string', 'another int', 'another string']
tbl = monotable.MonoTable()
tbl.guideline_chars = '-=-'
tbl.separated_guidelines = True

cells = [[123, 'import', 4567, 'this']]

print(tbl.table(headings, [], cells, title='Override on an instance.'))
```

```
          Override on an instance.
------  ------  -----------  --------------
an int  string  another int  another string
======  ======  ===========  ==============
   123  import         4567  this
------  ------  -----------  --------------
```

Note: Double check the spelling of the class variable. A misspelled variable name will be silently ignored.

These techniques work because:

- None of the instance variables assigned by __init__() depend on the value of any other class or instance variable.
- MonoTable member functions, except __init__(), do not modify any class or instance variables.

CHAPTER 15

Hints

- Text wrapping wraps text to a maximum width, but it can be less.
- Headings are not affected by width, fixed, and wrap format directives. A wider heading will take precedence.
- Auto-alignment always looks at the type of the cell. When reading keys from a cell that is a dictionary auto-alignment is determined by the type of the cell (in this case dict(), which auto-aligns to the left) and not the value of the key.
- In a format string a missing format directive end delimiter is not an error. The intended directive text will become part of the format_spec.
- The lsep format directive silently overrides the rsep format directive on the preceding column.
- When any scaling format function (thousands(), millions(), ...) is applied to a cell of type integer, the resulting value is promoted to float before it is formatted. A format spec compatible with float should be used.
- For the none, zero, lsep, rsep formatting directives a semicolon cannot be used in the =ccc part since it is interpreted as the delimiter between formatting directives. The delimiters may be changed by overriding the MonoTable class var option_spec_delimiters.
- Check spelling carefully when overriding a class variable. Misspelling will be silently ignored.
- Format directive none=ccc is all lower case.
- The file test/test_examples.py has PEP484 (mypy) type annotation comments for experimental static type checking. It can serve as a guide to solving type checking issues.
- In the code option_spec is a synonym for format directive.

CHAPTER 16

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index

Symbols

A

B

C

D

F

G

H

I

J

K

M

N

O

P

R

S

T

V

W