DabBiew Documentation

Release 0.1

Aaron Tumulak

Nov 18, 2017

Contents

I am playing around with Sphinx. Auto generation of documentation is . Since this makes the source code really long, I recommend finding something to fold docstrings for your editor.

Contents:

CHAPTER 1

DabBiew module reference

dabbiew.dabbiew.**debug**(*stdscr*)
Undo curses setup and enter ipdb debug mode.

https://stackoverflow.com/a/2949419/5101335

Parameters **stdscr** (*curses.window*) – window object to reset

dabbiew.dabbiew.**format_line**(*text*, *width*)
Pad or truncate text to fit width.

Text is left justified if there is sufficient room. Otherwise, text is truncated and ellipsis (\u2026) is appended.

```
>>> format_line('lorem ipsum', 16)
'lorem ipsum     '
>>> format_line('lorem ipsum', 6)
'lore\xe2\x80\xa6 '
>>> format_line('lorem ipsum', 2)
'\xe2\x80\xa6 '
>>> format_line('lorem ipsum', 1)
' '
```

Parameters

- **text** (*any type convertible to unicode*) – contents of cell
- **width** (*int*) – width of cell

Returns encoded unicode string formatted to fit in width

Return type str

dabbiew.dabbiew.**screen**(*start*, *end*, *cum_extents*, *offset*)
Generate column widths or row heights from screen start to end positions.

Indexing for start and end is analogous to python ranges. Start is first screen position that gets drawn. End does not get drawn. Returned tuples correspond to elements that are inside screen box.

```
>>> args = (5, 10, [0, 3, 6, 9, 12, 15], 0)
>>> [(col, width, cursor) for col, width, cursor in screen(*args)]
[(1, 1, 0), (2, 3, 1), (3, 1, 4)]
>>> args = (5, 10, [0, 3, 6, 9, 12, 15], 2)
>>> [(col, width, cursor) for col, width, cursor in screen(*args)]
[(1, 1, 2), (2, 3, 3), (3, 1, 6)]
```

Parameters

- **start** (*int*) – screen position start
- **end** (*int*) – screen position end
- **cum_extents** (*numpy.ndarray*) – cumulative sum of column widths or row heights
- **offset** (*int*) – shifts cursor position returned by fixed amount

Returns index of element, extent of element, position of element on screen

Return type int, int, int

dabbiew.dabbiew.**origin**(*current*, *start*, *end*, *cum_extents*, *screen*, *moving*)

Determine new origin for screen view if necessary.

The part of the DataFrame displayed on screen is conceptually a box which has the same dimensions as the screen and hovers over the contents of the DataFrame. The origin of the relative coordinate system of the box is calculated here.

```
>>> origin(0, 0, 0, [0, 4, 8, 12], 7, True)
0
>>> origin(4, 0, 2, [0, 4, 8, 12], 7, True)
5
>>> origin(5, 1, 1, [0, 4, 8, 12], 7, False)
4
```

Parameters

- **current** (*int*) – current origin of a given axis
- **start** (*int*) – leftmost column index or topmost row index selected
- **end** (*int*) – rightmost column index or bottommost row index selected
- **cum_extents** (*numpy.ndarray*) – cumulative sum of column widths or row heights
- **screen** (*int*) – total extent of a given axis
- **moving** – flag if current action is advancing

Type bool

Returns new origin

Return type int

dabbiew.dabbiew.**draw**(*stdscr*, *df*, *frozen_y*, *frozen_x*, *unfrozen_y*, *unfrozen_x*, *origin_y*, *origin_x*, *left*, *right*, *top*, *bottom*, *found_row*, *found_col*, *cum_widths*, *cum_heights*, *moving_right*, *moving_down*, *resizing*)

Refresh display with updated view.

Running line profiler shows this is the slowest part. Will optimize later.

```
>>> draw(curses.initscr(),
...      pd.DataFrame([[1, 2, 3], [4, 5, 6], [7, 8, 9]]),
...      1, 8, 10, 10,
...      0, 0, 0, 1, 0, 1, 0, 1,
...      np.append(np.array([0]), np.full(3, 3).cumsum()),
...      np.append(np.array([0]), np.full(3, 1).cumsum()),
...      False, False, True)
(0, 0)
```

Parameters

- **stdscr** (*curses.window*) – window object to update
- **df** (*pandas.DataFrame*) – underlying data to present
- **frozen_y** (*int*) – initial row offset before view box contents are shown
- **frozen_x** (*int*) – initial column offset before view box contents are shown
- **unfrozen_y** (*int*) – number of rows dedicated to contents of view box
- **unfrozen_x** (*int*) – number of columns dedicated to contents of view box
- **origin_y** (*int*) – y coordinate of bottommost part of view box
- **origin_x** (*int*) – x coordinate of leftmost part of view box
- **left** (*int*) – leftmost column of selection
- **right** – rightmost column of selection
- **top** (*int*) – topmost row of selection
- **bottom** (*int*) – bottommost row of selection
- **found_row** (*int*) – row containing current search match
- **found_col** (*int*) – column containing current search match
- **cum_widths** (*numpy.ndarray*) – cumulative sum of column widths
- **cum_heights** (*numpy.ndarray*) – cumulative sum of row heights
- **moving_right** (*bool*) – flag if current action is moving right
- **moving_down** (*bool*) – flag if current action is moving down
- **resizing** (*bool*) – flag if the selection is currently being resized

dabbiew.dabbiew.**advance**(*start*, *end*, *resizing*, *boundary*, *amount*)

Move down or right.

```
>>> advance(0, 0, True, 3, 1)
(0, 1, True)
>>> advance(0, 1, False, 3, 1)
(1, 2, True)
>>> advance(1, 2, True, 3, 1)
(1, 2, True)
>>> advance(1, 2, True, 3, 1)
(1, 2, True)
```

Parameters

- **start** (*int*) – leftmost column or topmost row

- **end** (*int*) – rightmost column or bottommost row
- **resizing** (*bool*) – flag if the selection is currently being resized
- **boundary** (*int*) – total number of columns or rows
- **amount** (*int*) – number of columns or rows to advance

dabbiew.dabbiew.**retreat**(*start*, *end*, *resizing*, *boundary*, *amount*)

Move up or left.

```
>>> retreat(1, 2, True, None, 1)
(1, 1, False)
>>> retreat(1, 1, True, None, 1)
(1, 1, False)
>>> retreat(1, 1, False, None, 1)
(0, 0, False)
>>> retreat(0, 0, False, None, 1)
(0, 0, False)
```

Parameters

- **start** (*int*) – leftmost column or topmost row
- **end** (*int*) – rightmost column or bottommost row
- **resizing** (*bool*) – flag if the selection is currently being resized
- **boundary** (*int*) – total number of columns or rows (unused in retreat)
- **amount** (*int*) – number of columns or rows to retreat

dabbiew.dabbiew.**number_in**(*keystroke_history*)

Returns number previous keystrokes have a number.

```
>>> number_in(deque(['s', 'p', 'a', 'm']))
1
>>> number_in(deque(['8', 's', 'p', 'a', 'm']))
1
>>> number_in(deque(['8', 's', 'p', 'a', 'm', '9']))
9
>>> number_in(deque(['8', 's', 'p', 'a', 'm', '9', '0']))
90
```

Parameters **keystroke_history** (*collections.deque*) – contains the last few keystrokes

Returns number inferred from keystroke history

Return type int

dabbiew.dabbiew.**expand_cumsum**(*start*, *end*, *cum_extents*, *amount*)

Increase each extent by a given amount and update cumulative extents.

```
>>> expand_cumsum(1, 2, np.array([0, 4, 8, 12, 16, 20]), 2)
array([ 0,  4, 10, 16, 20, 24])
```

Parameters

- **start** (*int*) – leftmost column or topmost row of range to expand
- **end** (*int*) – rightmost column or bottommost row of range to expand

- **cum_extents** (*numpy.ndarray*) – cumulative sum of column widths or row heights
- **amount** (*int*) – amount to increment each row or column in range

dabbiew.dabbiew.**contract_cumsum**(*start*, *end*, *cum_extents*, *amount*, *minimum_extent=2*)

Decrease each extent by a given amount and update cumulative extents.

```
>>> contract_cumsum(1, 2, np.array([0, 4, 8, 12, 16, 20]), 2)
array([ 0,  4,  6,  8, 12, 16])
>>> contract_cumsum(1, 2, np.array([0, 3, 6, 9, 12, 15]), 2)
array([ 0,  3,  6,  9, 12, 15])
```

Parameters

- **start** (*int*) – leftmost column or topmost row of range to contract
- **end** (*int*) – rightmost column or bottommost row of range to contract
- **cum_extents** (*numpy.ndarray*) – cumulative sum of column widths or row heights
- **amount** (*int*) – amount to decrement each row or column in range

dabbiew.dabbiew.**command_validator**(*keystroke*)

Change default keymappings.

Keybindings from more common ASCII control codes are remapped to emacs-type keybindings accepted by curses Textbox objects.

```
>>> command_validator(127) # backspace -> control-H
8
>>> command_validator(27) # escape -> control-G
7
>>> command_validator(ord('a'))
97
```

Parameters **keystroke** (*int*) – input read from curses Textbox

Returns remapped keystroke

Return type int

dabbiew.dabbiew.**show_prompt**(*stdscr*, *prompt*, *row*, *width*, *keystrokes=None*, *delay=0.0*)

Display a prompt for a command on the bottom of the screen.

```
>>> show_prompt(curses.initscr(), '>', 0, 10, delay=0.1,
...             keystrokes=(ord(k) for k in 'spam\rham'))
u'spam'
```

Parameters

- **stdscr** (*curses.window*) – window object to update
- **prompt** (*str*) – string to display before command prompt
- **row** (*int*) – y position on screen to draw
- **width** (*int*) – x width of prompt input field
- **keystrokes** (*generator*) – optional set of predetermined keystrokes (noninteractive)
- **delay** (*float*) – time to wait after each keystroke is rendered

Returns string read from prompt

Return type str

dabbiew.dabbiew.**next_match**(*df*, *string*, *row*, *col*)

Forward sweep columns then rows for entry containing string match.

```
>>> next_match(pd.DataFrame([['a', 'b', 'c'], ['d', 'e', 'f']]), 'e', 0, 0)
(1, 1)
>>> next_match(pd.DataFrame([['a', 'b', 'c'], ['d', 'e', 'f']]), 'e', 0, 2)
(1, 1)
>>> next_match(pd.DataFrame([['a', 'b', 'c'], ['d', 'e', 'f']]), 'e', 1, 2)
(1, 1)
>>> next_match(pd.DataFrame([['a', 'b', 'c'], ['d', 'e', 'f']]), 'g', 1, 2) is␣
↪None
True
```

Parameters

- **df** (*pandas.DataFrame*) – underlying data to present
- **string** (*str*) – string to match
- **row** (*int*) – search starting row
- **col** (*int*) – search starting col

Returns next matching row and column

Return type int, int

dabbiew.dabbiew.**prev_match**(*df*, *string*, *row*, *col*)

Reverse sweep columns then rows for entry containing string match.

```
>>> prev_match(pd.DataFrame([['a', 'b', 'c'], ['d', 'e', 'f']]), 'e', 1, 2)
(1, 1)
>>> prev_match(pd.DataFrame([['a', 'b', 'c'], ['d', 'e', 'f']]), 'e', 1, 0)
(1, 1)
>>> prev_match(pd.DataFrame([['a', 'b', 'c'], ['d', 'e', 'f']]), 'e', 0, 0)
(1, 1)
>>> prev_match(pd.DataFrame([['a', 'b', 'c'], ['d', 'e', 'f']]), 'g', 0, 0) is␣
↪None
True
```

Parameters

- **df** (*pandas.DataFrame*) – underlying data to present
- **string** (*str*) – string to match
- **row** (*int*) – search starting row
- **col** (*int*) – search starting col

Returns previous matching row and column

Return type int, int

dabbiew.dabbiew.**jump**(*left*, *right*, *top*, *bottom*, *rows*, *cols*, *to_row*, *to_col*, *resizing*)

Jump current selection to new position.

```
>>> jump(0, 0, 0, 0, 10, 10, 5, 5, False)
(5, 5, 5, 5, True, True)
```

Parameters

- **left** (*int*) – leftmost column of selection
- **right** – rightmost column of selection
- **top** (*int*) – topmost row of selection
- **bottom** (*int*) – bottommost row of selection
- **rows** (*int*) – total number of rows
- **cols** (*int*) – total number of columns
- **to_row** (*int*) – destination row for bottomright selection
- **to_col** (*int*) – destination column for bottomright selection
- **resizing** (*bool*) – flag if the selection is currently being resized

Returns new selection boundaries

Return type int, int, int, int, bool, bool

dabbiew.dabbiew.**eval_command**(*stdscr*, *df*, *command*, *row*, *left*, *right*, *top*, *bottom*, *keystrokes*)

Call method on DataFrame selection.

If the selection is just a single cell, the call is made to the entire DataFrame inplace. If the selection is more than one cell, then this function creates a new DataFrame and makes a nested call to the main function run().

Parameters

- **stdscr** (*curses.window*) – window object to update
- **df** (*pandas.DataFrame*) – underlying data to present
- **command** (*str*) – DataFrame method to call
- **row** (*int*) – y position on screen to draw input box
- **left** (*int*) – leftmost column of selection
- **right** – rightmost column of selection
- **top** (*int*) – topmost row of selection
- **bottom** (*int*) – bottommost row of selection
- **keystrokes** (*generator yielding int*) – keystrokes to use in autopilot.

dabbiew.dabbiew.**run**(*stdscr*, *df*, *keystrokes=None*)

Main loop; set state of window and wait for keystrokes.

```
>>> run(curses.initscr(),
...     pd.DataFrame([['a' ,'b', 'c'], [1, 2, 3], [4.0, 5.0, 6.0]]),
...     keystrokes=iter(ord(c) for c in 'vljhk.,><tyty[]GG$/c\rnp^
↪ggjvllv:sum()\rq:fail()\r:sort_values(1)\rsSx}q')) is None
True
```

Parameters

- **stdscr** (*curses.window*) – window object to update

- **df** (*pandas.DataFrame*) – underlying data to present
- **keystrokes** (*generator yielding int*) – keystrokes to use in autopilot.

dabbiew.dabbiew.**to_dataframe**(*filepath*)

Infer file type and load as DataFrame.

Parameters **filepath** (*str*) – path to file containing data

Returns loaded DataFrame

Return type pandas.DataFrame

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index