purdy

Release 1.10.2

Christopher Trudeau

Nov 02, 2022

CONTENTS

During talks or screencasts I don't want to be typing code, it is too error prone and too likely to mess up my speaking flow. **Purdy** is both a set of programs and a library to display colourized code in a series of animations.

The `purdy` command takes one of a Python program, a Python REPL console file or a Bash console file. Source code is presented to the screen as if typing. For console files, the typing pauses at a prompt, waiting for interaction. Prompts are:

- >>> or `...` for Python REPL
- $ for Bash console

If the program is paused at a prompt, pressing the **right** arrow will continue. Typing animation can be skipped over by pressing the letter "s" instead. Animation can be undone by pressing the **left** arrow. More info on keys can be found in the help dialog, viewed by pressing "?".

Example Usage:

```
$ purdy code-snippet.py
```

The result looks like this:

Once the code has been displayed, further key presses are ignored. At any time you can press "q" to quit.

CHAPTER

ONE

PURDY PROGRAMS

The following programs come with the *purdy* library:

- `purdy` – Animated display that looks like a program is being typed to the screen.
- `pat` – "purdy cat", prints ANSI colourized source.
- `prat` – "purdy RTF cat", prints colourized source in RTF document format. Particularly useful for copying to a clipboard and pasting full colourized source into a document. On OS X *prat* <*filename*> | *pbcopy* will put the output directly to the clipboard.
- `subpurdy` – Full set of commands to control Purdy. Sub-commands dictate behaviour. Does a variety of code presentation. Includes ANSI, RTF, HTML output as well as the typewriter animations.

More information can be found in the Command Line Program Documentation.

CHAPTER

TWO

PURDY TUI CONTROLS

The following keys help you to control the TUI purdy programs:

- `?` – Help screen
- `<RIGHT>` – next animation step
- `<LEFT>` – previous animation step
- `s` – go to the next step, skipping any animation

For custom made code using the purdy library, the following controls will also work:

- `S` – go to the next section, skipping any animation.
- `<TAB>` – focus next window area in a multi Screen display
- `<SHIFT><TAB>` – focus previous window area in a multi Screen display

Additionally the `s`, `S`, and `<LEFT>` commands all support skipping multiple steps by specifying a number first. For example the sequence `12s` would skip past the next 12 steps.

CHAPTER

THREE

PURDY LIBRARY

The `purdy` script is fairly simple. You can create more complex animations by writing programs using the purdy library. Custom programs can have split screens, highlighting lines, slide transitions and more. More information can be found in the Library Documentation.

CHAPTER

FOUR

INSTALLATION

```
$ pip install purdy
```

CHAPTER

FIVE

SUPPORTS

Purdy has been tested with Python 3.7 through 3.11. Terminal control is done with the Urwid library. Parsing and tokenization is done through Pygments. Both libraries are execellent and I'm grateful they're publically available.

CHAPTER

SIX

DOCS & SOURCE

Docs: http://purdy.readthedocs.io/en/latest/

Source: https://github.com/cltrudeau/purdy

Version: 1.10.2

6.1 Contents

6.1.1 Command Line Program Documentation

purdy Command

Displays a highlighted version of python text to the screen as if it is being typed

```
usage: purdy [-h] [--version] [-l {con,py3,bash,dbash,node}]
             [--maxheight MAXHEIGHT] [-c] [-x16] [-d DELAY | -w WPM]
             [--variance VARIANCE]
             filename
```

Positional Arguments

filename Name of file to parse

Named Arguments

--version show program's version number and exit

-l, --lexer Possible choices: con, py3, bash, dbash, node

Name of lexer to use to parse the file. Choices are: "con" (Python 3 Console), "py3" (Python 3 Source), "bash" (Bash Console), "dbash" (Bash Console with a dollar-sign prompt), "node" (JavaScript Node.js Console). If no choice given, attempts to determine the result automatically. If it cannot detect it, it assumes Python 3.

Default: "detect"

--maxheight	Sets a maximum screen height for the TUI screen viewer. Ignored if not in TUI mode. Default: 0
-c, --continuous	Instead of prentending to type like a human, just dump the file to the screen Default: False
-x16	Force 16 colour terminal mode in case 256 is not working to the screen Default: False
-d, --delay	Amount of time between each letter when in typewriter mode. Specified in milliseconds. Defaults to 130ms
-w, --wpm	Number of words per minute that the typing speed should look like
--variance	To make the typing look more real there is a variance in the delay between keystrokes. This value, in milliseconds is how much to go over or under the delay by. Defaults to +/- 30ms

`pat` Command

This command prints ANSI colourized versions of a file, parsing the file based on a limited number of pygments lexers. 'pat' is part of the 'purdy' library. A list of supported lexers is available in the help. If no lexer is specified the library attempts to determine which lexer to use automatically.

```
usage: pat [-h] [-l {con,py3,bash,dbash,node}] [--version] [--num NUM]
           [--highlight HIGHLIGHT]
           filename
```

Positional Arguments

filename	Name of file to parse

Named Arguments

-l, --lexer	Possible choices: con, py3, bash, dbash, node Name of lexer to use to parse the file. Choices are: "con" (Python 3 Console), "py3" (Python 3 Source), "bash" (Bash Console), "dbash" (Bash Console with a dollar-sign prompt), "node" (JavaScript Node.js Console). If no choice given, attempts to determine the result automatically. If it cannot detect it, it assumes Python 3. Default: "detect"
--version	show program's version number and exit
--num	Display line numbers with code starting with the value given here Default: -1
--highlight, --hl	Highlight certain line numbers when displaying the code. Line numbers are 1-indexed. Multiple lines can be higlighted using a hyphen for range (e.g. 1-4, inclusive) or a comma separated list (e.g. 1-4,7,9 is line 1, 2, 3, 4, 7 and 9).

`prat` Command

This command prints colourized RTF version of a file, parsing the file based on a limited number of pygments lexers. 'prat' is part of the 'purdy' library. A list of supported lexers is available in the help. If no lexer is specified the library attempts to determine which lexer to user automatically.

```
usage: prat [-h] [-l {con,py3,bash,dbash,node}] [--version] [--num NUM]
            [--background BACKGROUND] [--highlight HIGHLIGHT]
            filename
```

Positional Arguments

filename Name of file to parse

Named Arguments

-l, --lexer Possible choices: con, py3, bash, dbash, node

Name of lexer to use to parse the file. Choices are: "con" (Python 3 Console), "py3" (Python 3 Source), "bash" (Bash Console), "dbash" (Bash Console with a dollar-sign prompt), "node" (JavaScript Node.js Console). If no choice given, attempts to determine the result automatically. If it cannot detect it, it assumes Python 3.

Default: "detect"

--version show program's version number and exit

--num Display line numbers with code starting with the value given here

Default: -1

--background, --bg Change the background colour of the document. When using the –highlight option, you you should also set a background colour, otherwise the background will turn white due to how RTF supports colouring. Format of the colour is like an HTML colour, e.g. #c1b455, without the leading #

--highlight, --hl Highlight certain line numbers when displaying the code. Line numbers are 1-indexed. Multiple lines can be higlighted using a hyphen for range (e.g. 1-4, inclusive) or a comma separated list (e.g. 1-4,7,9 is line 1, 2, 3, 4, 7 and 9).

`subpurdy` Command

Purdy is a library and set of command line tools for displaying code. You can write your own code to display specific content using the library, or use the subcommands of this program for preset usages. The 'purdy' command uses the Urwid library to display a colourized version of your code in the console, and is a wrapper to the subcommand of the same name.

```
usage: subpurdy [-h] [-l {con,py3,bash,dbash,node}] [--version]
                {purdy,tokens,print,html,rtf} ... filename
```

Positional Arguments

filename Name of file to parse

Named Arguments

-l, --lexer Possible choices: con, py3, bash, dbash, node

Name of lexer to use to parse the file. Choices are: "con" (Python 3 Console), "py3" (Python 3 Source), "bash" (Bash Console), "dbash" (Bash Console with a dollar-sign prompt), "node" (JavaScript Node.js Console). If no choice given, attempts to determine the result automatically. If it cannot detect it, it assumes Python 3.

Default: "detect"

--version show program's version number and exit

subcommands

command Possible choices: purdy, tokens, print, html, rtf

Sub-commands:

purdy

Display code in a interactive console window. Code is written to the screen as if it is being typed

```
subpurdy purdy [-h] [-c] [-x16] [-d DELAY | -w WPM] [--variance VARIANCE]
```

Named Arguments

-c, --continuous Instead of prentending to type like a human, just dump the file to the screen

Default: False

-x16 Force 16 colour terminal mode in case 256 is not working to the screen

Default: False

-d, --delay Amount of time between each letter when in typewriter mode. Specified in milliseconds. Defaults to 130ms

-w, --wpm Number of words per minute that the typing speed should look like

--variance To make the typing look more real there is a variance in the delay between keystrokes. This value, in milliseconds is how much to go over or under the delay by. Defaults to +/- 30ms

tokens

Prints out each line in a file with the corresponding tokens indented beneath it

```
subpurdy tokens [-h] [--blackandwhite]
```

Named Arguments

--blackandwhite, --bw By default code lines are highlighted using ANSI colour. This flag turns this off.

Default: False

print

Prints code to screen using colourized ANSI escape sequences

```
subpurdy print [-h] [--num NUM] [--highlight HIGHLIGHT]
```

Named Arguments

--num Display line numbers with code starting with the value given here

Default: -1

--highlight, --hl Highlight certain line numbers when displaying the code. Line numbers are 1-indexed. Multiple lines can be higlighted using a hyphen for range (e.g. 1-4, inclusive) or a comma separated list (e.g. 1-4,7,9 is line 1, 2, 3, 4, 7 and 9).

html

Prints code to screen formatted as an HTML div

```
subpurdy html [-h] [--num NUM] [--highlight HIGHLIGHT] [--full]
```

Named Arguments

--num Display line numbers with code starting with the value given here

Default: -1

--highlight, --hl Highlight certain line numbers when displaying the code. Line numbers are 1-indexed. Multiple lines can be higlighted using a hyphen for range (e.g. 1-4, inclusive) or a comma separated list (e.g. 1-4,7,9 is line 1, 2, 3, 4, 7 and 9).

--full By default only a snippet of HTML is displayed inside a . This flag produces a full HTML document.

Default: False

rtf

Prints code to screen formatted as an RTF document

```
subpurdy rtf [-h] [--num NUM] [--background BACKGROUND]
             [--highlight HIGHLIGHT]
```

Named Arguments

--num Display line numbers with code starting with the value given here

Default: -1

--background, --bg Change the background colour of the document. When using the –highlight option, you you should also set a background colour, otherwise the background will turn white due to how RTF supports colouring. Format of the colour is like an HTML colour, e.g. #c1b455, without the leading #

--highlight, --hl Highlight certain line numbers when displaying the code. Line numbers are 1-indexed. Multiple lines can be higlighted using a hyphen for range (e.g. 1-4, inclusive) or a comma separated list (e.g. 1-4,7,9 is line 1, 2, 3, 4, 7 and 9).

6.1.2 Library Documentation

In addition to the easy to use command-line script, you can also write programs using the purdy library. The purdy command-line script uses this library to display a simple file to the screen.

Example Program

```
# reads 'my_code.py' and displays it to the screen with line numbers

from purdy.actions import Append
from purdy.content import Code
from purdy.ui import SimpleScreen

# Screen is the entry to showing content
screen = SimpleScreen(starting_line_number=1)

# Screen has one display area called "code_box", your actions need access
# to this to write the code
code_box = screen.code_box

# read 'my_code.py' and parse it using the Python 3 lexer
blob = Code('code.py', lexer_name='py3')

# actions are like slides in the slides show that is purdy
actions = [
    # append the contents of the blob to the display code box
    Append(code_box, blob),
]
```

(continues on next page)

(continued from previous page)

```
# start the display event loop
screen.run(actions)
```

Every purdy library program needs to create a *Screen* or one of its children. The Screen is what controls the display. *Screen* and its children provide one or more *CodeBox* objects which is a widget on the screen that displays code. You combine a series of `purdy.ui.actions` to display and alter the code. View more examples in the *Sample Code* section.

6.1.3 Library API

UI (purdy.ui.py)

This module is the entry point for the code viewers. It is a lightweight proxy to implementation of a screen. Screen implementations are found in `purdy.iscreen`. All programs using the purdy library need to create a *Screen* object or one of its children. The factory in this module determines which actual implementation is loaded.

class purdy.ui.**CodeBox**(*starting_line_number=- 1*, *auto_scroll=True*, *height=0*, *compact=False*)

Specifies a box to contain code. *Screen* uses these to determine widget layout, subsequent actions are done within the context of this box. When `CodeBox.build()` is called by a *Screen* class a widget is built and this box is added to the screen.

Parameters

- **starting_line_number** – -1 means no line numbers, anything larger will be the first line number displayed. Defaults to -1
- **auto_scroll** – When True, `purdy.widgets.CodeBox` created by this specification automatically scrolls to newly added content. Defaults to True.
- **height** – Number of lines the row containing this box should be. A value of 0 indicates automatic spacing. Defaults to 0.
- **compact** – if False, the dividing line between this box and the next has a 1-line empty boundary. Parameter is ignored if there is no item after this one in the rows=[] listing. Defaults to False

class purdy.ui.**Screen**(*settings=None*, *rows=[]*, *max_height=0*)

Represents the main UI window for the TUI application. The layout is specified by passing in one or more *CodeBox* or *TwinCodeBox* objects to the constructor. Each box will have a corresponding `purdy.widgets.CodeWidget` inside of the UI for displaying code.

Parameters

- **settings** – a settings dictionary object. Defaults to *None* which uses the default settings dictionary: `settings.settings`
- **rows** – a list containing one or more *CodeBox* or *TwinCodeBox* definitions, to specify the layout of the screen
- **max_height** – maximum display height in TUI mode, defaults to 0, meaning no max

Example:

```
from purdy.actions import AppendAll
from purdy.content import Code
from purdy.ui import Screen, CodeBox, TwinCodeBox

screen = Screen(rows=[TwinCodeBox(height=8),
        CodeBox(auto_scroll=False)])

c1 = Code(contents_filename='c1.py', starting_line_number=1)
c2 = Code(contents_filename='c2.py')
c3 = Code(contents_filename='c3.py')

actions = [
    AppendAll(screen.code_boxes[0], c1),
    AppendAll(screen.code_boxes[1], c2),
    AppendAll(screen.code_boxes[2], c3),
]

screen.run(actions)
```

The above would produce a screen with two rows, the first row having two `purdy.widgets.CodeWidget` objects side by side, the second having a single one. The top left box would have line numbers turned on, both top boxes are 8 lines tall, and the bottom box has auto scrolling turned off.

The screen would be divided like this:

Command Line Parameters

Unless "deactivate_args" is set to True in `purdy.settings` (False by default), Screen will also parse command line arguments. This allows scripts calling the library to change their behaviour with switches.

Supported switches are:

- **–debugsteps** Print out the animation steps, grouped by Cell and exit
- **–export** Print out the results of the actions
- **–exportrtf** Print out the results of the actions in RTF format

run(*actions*)
Calls the main display event loop. Does not return until the UI exits.

class purdy.ui.**SimpleScreen**(*settings=None*, *starting_line_number=- 1*, *auto_scroll=True*, *max_height=0*)
Convenience implementation of *Screen* that supports a single *CodeBox*. The code box is available as `SimpleScreen.code_box`.

Parameters

- **settings** – a settings dictionary object. Defaults to *None* which uses the default settings dictionary: `settings.settings`
- **starting_line_number** – starting line number for the created code box
- **auto_scroll** – When True, the class:*ui.CodeBox* automatically scrolls to newly added content. Defaults to True.
- **max_height** – maximum display height in TUI mode, defaults to 0, meaning no max

class purdy.ui.**SplitScreen**(*settings=None*, *top_starting_line_number=- 1*, *top_auto_scroll=True*, *bottom_starting_line_number=- 1*, *bottom_auto_scroll=True*, *top_height=0*, *compact=False*, *max_height=0*)

Convenience implementation of *Screen* that supports two *CodeBox* instances, stacked vertically and separated by a dividing line. The code boxes are `SplitScreen.top` and `SplitScreen.bottom`.

Parameters

- **settings** – a settings dictionary object. Defaults to *None* which uses the default settings dictionary: `settings.settings`
- **top_starting_line_number** – starting line number for the top code box
- **top_auto_scroll** – When True, the top `ui.CodeBox` automatically scrolls to newly added content. Defaults to True.
- **bottom_starting_line_number** – starting line number for the bottom code box
- **bottom_auto_scroll** – When True, the bottom `ui.CodeBox` automatically scrolls to newly added content. Defaults to True.
- **top_height** – Number of lines the top box should be. A value of 0 indicates top and bottom should be the same size. Defaults to 0.
- **compact** – True if for the dividing line between the top and bottom screens is to have no margin. Defaults to False
- **max_height** – maximum display height in TUI mode, defaults to 0, meaning no max

class purdy.ui.**TwinCodeBox**(*left_starting_line_number=- 1*, *left_auto_scroll=True*, *left_weight=1*, *right_starting_line_number=- 1*, *right_auto_scroll=True*, *right_weight=1*, *height=0*, *compact=False*)

Specifies two side-by-side CodeBoxes. The contained CodeBoxes can be either accessed as twin.left and twin.right, or twin[0] and twin[1]. When `TwinCodeBox.build()` is called, the widgets are created and added to the `Screen.code_boxes` list sequentially, i.e. a single TwinCodeBox results in two CodeBox items in Screen's list.

Parameters

- **left_starting_line_number** –
- **right_starting_line_number** – -1 to turn line numbers off, any higher value is the first number to display. Defaults to -1
- **left_auto_scroll** –
- **right_auto_scroll** – True to specify that scrolling happens automatically for the left and right boxes created by this spec. Defaults to True.
- **left_weight** –
- **right_weight** – relative weights for the widths of the columns, if the values are the same then the columns are the same width, otherwise the widths are formed based on the ratio of left:right. Example: left_weight=2, right_weight=1 means the left side will be twice the width of the right side. Both values default to 1.
- **height** – number of lines for the row this set of boxes is in. The default of 0 specifies automatic height
- **compact** – if False, the dividing line between this box and the next has a 1-line empty boundary. Parameter is ignored if there is no item after this one in the rows=[] listing. Defaults to False

Content

Reperesntations of source code are found in this module.

class purdy.content.**Code**(*filename=''*, *text=''*, *lexer_name='detect'*, *purdy_lexer=None*)

Represents source code from the user.

Parameters

- **filename** – name of a file to read for content. If both this and *text* is given, *filename* is used first
- **text** – a string containing code
- **lexer_name** – name of lexer to use to tokenize the code, defaults to 'detect', attempting to auto detect the type of content. See *purdy.parser.PurdyLexer* for a list of available lexers.
- **purdy_lexer** – if lexer_name is "custom" this parameter is expected to contain a purdy.parser.PurdyLexer object.

fold_lines(*start*, *end*)

Call this method to replace one or more lines with a vertical elipses, i.e. a fold of the code.

Parameters

- **start** – line number of the listing to start code folding on. 1-indexed.
- **end** – line number to fold until, inclusive (1-indexed).

inline_replace(*line_no*, *pos*, *content*)

Replaces the contents of a line starting at pos with the new content

Parameters

- **line_no** – number of the line to replace, 1-indexed
- **pos** – position number to start the replacement at, 1-indexed
- **content** – content to replace with

insert_line(*line_no*, *content*)

Inserts the given line into the source, pushing the content down from the given line number

Parameters

- **line_no** – number of the line to insert at, 1-indexed
- **content** – content to insert

left_justify()

Removes a consistent amount of leading whitespace from the front of each line so that at least one line is left-justified.

> **Warning:** will not work with mixed tabs and spaces

python_portion(*name*)

Treates the source in this object as Python and then finds either the named function, class, or assigned variable and replaces the source with only the found item.

> **Warning:** If the named item is not found your source will be empty!

Parameters **name** – dot notated name of a function or class. Examples: *Foo.bar* would find the *bar* method of class *Foo* or an inner function named *bar* in a function named *Foo*

remove_double_blanks(*trim_whitespace=True*)
Removes the second of two blanks in a row. If trim_whitespace is True (default) a line with only whitespace is considered blank, otherwise it only looks for n

remove_lines(*line_no*, *count=1*)
Removes one or more lines from the source listing.

Parameters

- **line_no** – number of the line to remove, 1-indexed
- **count** – number of lines to remove, defaults to 1

replace_line(*line_no*, *content*)
Replaces the given line with new content

Parameters

- **line_no** – number of the line to replace, 1-indexed
- **content** – content to replace it with

subset(*start*, *end*)
Returns a new Code object containing just the subset of lines identified in this call

Parameters

- **start** – line number of the listing to start the subset at. 1-indexed
- **end** – line number to finish the subset on, inclusive. 1-indexed.

Returns Code object

Actions

Library users specify a series of actions that turn into the presentation animations in the Urwid client. An action is similar to a slide in a slide show, except it can both present and change lines of code on the screen.

All purdy library programs have the following basic structure:

```
screen = Screen(...)
actions = [ ... ]
screen.run(actions)
```

Each action gets translated into a series of steps defined in the `purdy.animation` module.

class purdy.actions.**Append**(*code_box*, *code*)
Adds the content of a *purdy.content.Code* object to the end of a *purdy.ui.CodeBox*.

Parameters

- **code_box** – the *purdy.ui.CodeBox* instance to insert code into
- **code** – a *purdy.content.Code* object containing the source code to insert.

class purdy.actions.**Clear**(*code_box*)
Clears the contents of a *purdy.ui.CodeBox*.

Parameters **code_box** – the *purdy.ui.CodeBox* instance where the code is to be replaced

class purdy.actions.**Fold**(*code_box*, *position*, *end=- 1*)
Folds code by replacing one or more lines with a vertical elipses symbol.

Parameters

- **code_box** – the *purdy.ui.CodeBox* instance to modify
- **position** – line number to begin the fold at. Position is 1-indexed.
- **end** – line number to finish the folding at, inclusive. A value of -1 can be used to fold to the end of the box. Defaults to -1.

class purdy.actions.**Highlight**(*code_box*, *spec*, *highlight_on*)
Cause one or more lines of code to have highlighting turned on or off

Parameters

- **code_box** – *purdy.ui.CodeBox* to perform on
- **spec** – either a string containing comma separated and/or hyphen separated integers (e.g. "1,3,7-9") or a list of integers specifying the lines in the code box to highlight. Line numbers are 1-indexed
- **highlight_on** – True to turn highligthing on, False to turn it off

class purdy.actions.**HighlightChain**(*code_box*, *spec_list*)
A common pattern with highlighting lines is to turn a highlight on for some set of lines, then turn it off and turn it on for more lines. This is a convenience wrapper to the *Highlight* action, turning items on and off in series.

Parameters

- **code_box** – *purdy.ui.CodeBox* to perform series of highlight on
- **spec_list** – a list of highlight specs (see *Highlight* for details on a spec)

class purdy.actions.**Insert**(*code_box*, *position*, *code*)
Inserts the content of a *purdy.content.Code* object to a specified line in a *purdy.ui.CodeBox*. Pushes content down, inserting at "1" is the beginning of the list. Position is 1-indexed

Parameters

- **code_box** – the *purdy.ui.CodeBox* instance to insert code into
- **position** – line number to insert code at. Position is 1-indexed. Content is pushed down, so a value of "1" inserts at the beginning. Negative indicies are supported. A value of "0" will append the code to the bottom.
- **code** – a *purdy.content.Code* object containing the source code to insert.

class purdy.actions.**Remove**(*code_box*, *position*, *size*)
Removes one or more lines of a *purdy.ui.CodeBox*.

Parameters

- **code_box** – the *purdy.ui.CodeBox* instance where the code is to be replaced
- **position** – line number to replace the code at. Position is 1-indexed. Negative indicies are supported.
- **size** – number of lines to remove.

class purdy.actions.**Replace**(*code_box*, *position*, *code*)
Replaces one or more lines of a *purdy.ui.CodeBox* using the content of a *purdy.content.Code* object. This action attempts to overwrite using the number of lines in the *purdy.content.Code* object passed in, it is up to you to make sure there is enough space in your CodeBox.

Parameters

- **code_box** – the *purdy.ui.CodeBox* instance where the code is to be replaced
- **position** – line number to replace the code at. Position is 1-indexed. Negative indicies are supported.
- **code** – a *purdy.content.Code* object containing the source code to insert.

class purdy.actions.**RunFunction**(*fn*, *undo*, **args*, ***kwargs*)
Calls the function passed in, allowing the execution of code during the playing of actions.

Parameters

- **fn** – function to be called
- **undo** – function to be called when this Action is undone, can be None
- ****kwargs** (**args,*) – any remaining arguments are passed to the functions when they are called

class purdy.actions.**Section**
Marker for the beginning of a section. In the TUI you can skip to the next section marker using "S".

class purdy.actions.**Shell**(*code_box*, *cmd*)
Runs a shell command via subprocess. Does not display the command (you're better off using a typewriter command to show it, then use this to spit out the results). Command and results are added to the *purdy.ui.CodeBox*.

Parameters

- **code_box** – the *purdy.ui.CodeBox* instance where the code is to be appended
- **cmd** – string containing the shell command and its paramters. Example: `ls -la`.

class purdy.actions.**Sleep**(*time*)
Causes animations to pause for the given amount of time. Note that this action happens within a cell, so is considered part of the group of animation steps done together. For example if Append + Sleep + Append is part of the same cell it is all done/undone together.

Parameters **time** – Either the amount of time to sleep in seconds (ints and floats supported), or a tuple containing a pair of times specifying the range of a random value to sleep.

class purdy.actions.**StopMovie**
Causes the presentation *purdy.ui.Screen* to exit movie mode

class purdy.actions.**Suffix**(*code_box*, *position*, *source*)
Adds the provided text to the end of an existing line in a *purdy.ui.CodeBox*.

Parameters

- **code_box** – the *purdy.ui.CodeBox* instance where the code is to be appended
- **position** – line number to replace the code at. Position is 1-indexed. Negative indicies are supported.
- **source** – string containing content to append to the line

class purdy.actions.**Transition**(*code_box*, *code=None*, *code_box_to_copy=None*)
Replaces the contents of a *purdy.ui.CodeBox* with new content, doing a wipe animation from top to bottom. Only one of *code* or *code_box_to_copy* should be given, both can be blank to transition to an empty screen.

Parameters

- **code_box** – the *purdy.ui.CodeBox* instance to perform the transition on
- **code** – a *purdy.content.Code* object containing the source code replacing the existing content. Should not be used at the same time as code_box_to_copy parameter.
- **code_box_to_copy** – a code box containing rendered code to copy into this one to display. This is typically a `VirtualCodeBox`. Should not be used at the same time as code parameter.

class purdy.actions.**Wait**
Causes the animations to wait for a *right arrow* key press before continuing.

Typewriter Actions

Typewriter actions display code using a typewriter animation. Code content is displayed a letter at a time as if someone is typing. The `purdy.settings` module contains default values for typing speeds and variance time between letters being pressed.

When the code in question is based on a console, the typewriter will wait for the *right arrow* to be pressed whenever it sees a prompt. For example, when appending Python REPL code, the >>> will cause the interface to wait.

class purdy.actions.**AppendTypewriter**(*code_box*, *code*)
Adds the content of a *purdy.content.Code* object to a *purdy.ui.CodeBox* using the typewriter animation.

Parameters

- **code_box** – the *purdy.ui.CodeBox* instance to append code into
- **code** – a *purdy.content.Code* object containing the source code to insert.

class purdy.actions.**InsertTypewriter**(*code_box*, *position*, *code*)
Inserts the contents of a *purdy.content.Code* object at the given position using the typewriter animation.

Parameters

- **code_box** – the *purdy.ui.CodeBox* instance to append code into
- **position** – line number to insert the code at. Position is 1-indexed.
- **code** – a *purdy.content.Code* object containing the source code to insert.

class purdy.actions.**SuffixTypewriter**(*code_box*, *position*, *source*)
Adds the provided text to the end of an existing line in a *purdy.ui.CodeBox* using a typewriter animation.

Parameters

- **code_box** – the *purdy.ui.CodeBox* instance to append code into
- **position** – line number to insert the code at. Position is 1-indexed. Negative indicies are supported.
- **source** – a string to be appended to the given line

Default Settings

settings.**settings**

```
"""
Settings (purdy.settings.py)
============================

Defines the default settings for :class:`purdy.ui.Screen` objects, can be
overridden by passing an altered dictionary into the Screen's constructor.
"""

settings = {
    # delay between characters appearing on screen (in milliseconds)
    'delay':130,

    # range of random time in milliseconds to change the delay; makes typing
    # look more natural
    'delay_variance':30,

    # movie mode: instead of waiting for key presses, play like a movie, -1
    # disables, otherwise value in milliseconds for delay between played steps
    # (like 'delay' field
    'movie_mode':-1,

    # xterm colour mode, anything but 256 gives 16 colour mode
    'colour':256,

    # if True, stops Screen from running argparse
    'deactivate_args':False,

    # max height for presentation, only works in TUI mode, 0 == no max
    'max_height':0,
}
```

6.1.4 Sample Code

Here are some examples of scripts using *purdy* as a library. Full source is available in the repository: https://github.com/cltrudeau/purdy/tree/master/extras/samples

```
#!/usr/bin/env python

### Example purdy library code
#
# Displays a colourized Python REPL session to the screen

from purdy.actions import Append
from purdy.content import Code
from purdy.ui import SimpleScreen

screen = SimpleScreen()
blob = Code('../display_code/console.repl')
```

(continues on next page)

(continued from previous page)

```
actions = [
    Append(screen.code_box, blob),
]

if __name__ == '__main__':
    screen.run(actions)
```

```
#!/usr/bin/env python

### Example purdy library code
#
# Appends the same colourized Python REPL session to the screen multiple
# times, waiting for a keypress between each

from purdy.actions import Append, Wait
from purdy.content import Code
from purdy.ui import SimpleScreen

screen = SimpleScreen(starting_line_number=1)
code_box = screen.code_box
blob = Code('../display_code/simple.repl')

actions = [
    Append(code_box, blob),
    Wait(),
    Append(code_box, blob),
    Wait(),
    Append(code_box, blob),
    Wait(),
    Append(code_box, blob),
    Wait(),
    Append(code_box, blob),
    Wait(),
    Append(code_box, blob),
    Wait(),
    Append(code_box, blob),
    Wait(),
    Append(code_box, blob),
    Wait(),
    Append(code_box, blob),
    Wait(),
    Append(code_box, blob),
    Wait(),
]

if __name__ == '__main__':
    screen.run(actions)
```

```
#!/usr/bin/env python

### Example purdy library code
```

(continues on next page)

(continued from previous page)

```
#
# Uses the typewriter animation to display a bash console session

from purdy.actions import AppendTypewriter
from purdy.content import Code
from purdy.ui import SimpleScreen

screen = SimpleScreen()
code_box = screen.code_box
blob = Code('../display_code/curl.bash')
actions = [
    AppendTypewriter(code_box, blob),
]

if __name__ == '__main__':
    screen.run(actions)
```

```
#!/usr/bin/env python

### Example purdy library code
#
# Demonstrates the Shell action that runs a subprocess and returns the result

from purdy.actions import Shell, AppendTypewriter
from purdy.content import Code
from purdy.ui import SimpleScreen

screen = SimpleScreen()
code_box = screen.code_box

cmd1 = 'echo "hello there"'
cmd2 = 'echo "it is a nice day today"'

blob = Code(text=f'$ {cmd1}')
blob2 = Code(text=f'$ {cmd2}')

actions = [
    AppendTypewriter(code_box, blob),
    Shell(code_box, cmd1),
    AppendTypewriter(code_box, blob2),
    Shell(code_box, cmd2),
]

if __name__ == '__main__':
    screen.run(actions)
```

```
#!/usr/bin/env python

### Example purdy library code
#
# Demonstrates the code folding mechanism
```

(continues on next page)

(continued from previous page)

```
from purdy.actions import Append, Fold, Wait, Clear
from purdy.content import Code
from purdy.ui import SimpleScreen

code = Code('../display_code/code.py')

screen = SimpleScreen(starting_line_number=1)
box = screen.code_box

actions = [
    Append(box, code),
    Wait(),
    Fold(box, 20, 23),
    Wait(),
    Fold(box, 27),
    Wait(),
    Clear(box),           # test long fold without wait, used to crash
    Append(box, code),
    Fold(box, 2),
]

if __name__ == '__main__':
    screen.run(actions)
```

```
#!/usr/bin/env python

### Example purdy library code
#
# Demonstrates highlighting and unhighlighting lines of code

from purdy.actions import Append, Highlight, Wait
from purdy.content import Code
from purdy.settings import settings
from purdy.ui import SimpleScreen

#settings['colour'] = 16
screen = SimpleScreen(settings, starting_line_number=1)
code_box = screen.code_box
blob = Code('../display_code/console.repl')

actions = [
    Append(code_box, blob),
    Wait(),
    Highlight(code_box, range(5, 41), True),
    Wait(),
    Highlight(code_box, '5,6,10-20', False),
]

if __name__ == '__main__':
    screen.run(actions)
```

```
#!/usr/bin/env python

### Example purdy library code
#
# Demonstrates appending strings to the end of existing lines as well as
# replacing lines. Both done with and without the typewriter animation.

from purdy.actions import (Append, Wait, Suffix, SuffixTypewriter,
    Replace, Remove, InsertTypewriter)
from purdy.content import Code
from purdy.ui import SplitScreen

screen = SplitScreen(top_starting_line_number=10)
top = screen.top
bottom = screen.bottom

source = """\
@decorator
def foo(x):
    \"\"\"Multi-line
    doc string
    \"\"\"
    for index in range(1, x):
        blah = '''
            thing'''
        # about to print
        print(index)
"""
code = Code(text=source)

source = """\
>>> a = 1
>>> b = 2
>>> c = 3
"""
repl = Code(text=source)

actions = [
    Append(top, code),
    Append(bottom, repl),
    Wait(),
    Suffix(top, 1, 's'),
    Suffix(top, 1, ' # append'),
    Suffix(top, 2, ' # append'),
    Suffix(top, 3, ' more string now'),
    Suffix(top, 5, ' # append'),
    Suffix(top, 6, ' # append'),
    Suffix(top, 7, '  inside blah mline'),
    Suffix(top, 8, ' # append'),
    Suffix(top, 9, ' more comment'),
    Suffix(top, 10, ' # append'),
    Wait(),
    SuffixTypewriter(bottom, 2, '9'),
```

(continues on next page)

(continued from previous page)

```
    Wait(),
]

blob1 = Code(text='>>> d = 4')
blob2 = Code(text="""\
>>> e = 5
>>> f = 6
""")

actions.extend([
    Replace(bottom, 3, blob1),
    Wait(),
    SuffixTypewriter(bottom, 1, '56789'),
    Wait(),
    Suffix(bottom, 1, '333'),
    SuffixTypewriter(bottom, 1, '444'),
    Wait(),
    Remove(bottom, 2, 1),
    InsertTypewriter(bottom, 2, blob2),
    InsertTypewriter(bottom, 0, blob2),
])

if __name__ == '__main__':
    screen.run(actions)
```

```
#!/usr/bin/env python

### Example purdy library code
#
# Demonstrates the slide transition animation

from purdy.actions import Append, Wait, Transition, Fold
from purdy.content import Code
from purdy.ui import SimpleScreen, VirtualCodeBox

screen = SimpleScreen(starting_line_number=10)
code_box = screen.code_box
blob = Code('../display_code/simple.repl')
blob2 = Code('../display_code/traceback.repl')
blob3 = Code('../display_code/decorator.repl')

vbox = VirtualCodeBox(starting_line_number=20, display_mode='urwid')

# prep vbox for copy
vbox.perform_actions([
    Append(vbox, blob3),
    Fold(vbox, 2, 2),
])

actions = [
    Append(code_box, blob2),
    Wait(),
```

(continues on next page)

(continued from previous page)

```
    Transition(code_box),     # test transition to empty
    Append(code_box, blob),
    Wait(),

    # Test Wait after Transition and code box copy
    Transition(code_box, code_box_to_copy=vbox),
    Wait(),
    Append(code_box, blob2),
]

if __name__ == '__main__':
    screen.run(actions)
```

6.1.5 Implementation

This is the documentation for the underlying implementing classes. For the most part you don't need to understand it if you're just calling into the purdy library.

Animation Cells

A Cell represents an animation used by the `purdy.animation.manager.AnimationManager`. A Cell is responsible for rendering or undoing work to a `purdy.widget.CodeBox`. Animating cells can partially render then wait for an alarm provided by the manager to continue rendering.

class purdy.animation.cell.**GroupCell**

Groups steps together into a bundle that can be rendered or undone together. Implements animation alarms so the manager can do timed call backs into the group and continue rendering.

purdy.animation.cell.**group_steps_into_cells**(*steps*)

Actions create multiple steps possibly with cell breaks between them. This method takes a list of steps and returns a list of Cell objects, grouping the steps by break marker

Parameters **steps** – a list of steps

Returns a list of Cell objects

Animation Management

This module handles the slide rendering animation in the urwid purdy player

Animation Steps

An animation is created through a series of steps that are executed together. A *cell.GroupCell* wraps these steps. When the user moves forwards and backwards through the animations each cell is rendered or undone. This module

exception purdy.animation.steps.**StopMovieException**

Command Line Tools

Several of the command line tools have common arguments and needs. This file defines helper functions so these are defined once.

Colour Module (purdy.colour)

Contains classes to convert tokens to colour according to the various supported renderers and palettes.

Parser

This contains methods and classes to manage parsing of code

class purdy.parser.**CodePart**(*token*, *text*)

property text
Alias for field number 1

property token
Alias for field number 0

class purdy.parser.**PurdyLexer**(*name*, *description*, *pygments_lexer_cls*, *is_console*, *palette*)
Container for the built-in supported lexers. This class is where the names of the lexers are defined. Current choices are:

- 'con' – Python 3 Console
- 'py3' – Python 3 Source code
- 'bash' – interactive Bash session
- 'dbash' – interactive Bash session using a dollar sign prompt
- 'node' – interactive JavaScript / Node.js session

purdy.parser.**parse_source**(*source*, *lexer*)
Parses blocks of source text, returning a list of `CodeLine` objects.

purdy.parser.**token_ancestor**(*token*, *ancestor_list*)
Tokens are hierarchical, in some situations you need to translate a token into one from a known list, e.g. turning a "Token.Literal.Number.Integer" into a "Number". This method takes a token and a list of approved ancestors and attempts to make the map. If no ancestor is found then a generic "Token" object is returned

Parameters

- **token** – token to translate into an approved ancestor
- **ancestor_list** – list of approved ancestor tokens

purdy.parser.**token_is_a**(*token1*, *token2*)
Returns true if token1 is the same type as or a child type of token2

Scribe Module (purdy.scribe.py)

Methods for transforming code into different representations on stdout.

purdy.scribe.**print_html**(*listing*, *snippet=True*)
Prints the code in an HTML representation.

Parameters

- **listing** – Listing object containing code to print
- **snippet** – if True, prints out just the containing the code. Otherwise, prints a full valid HTML file. Defaults to True.

purdy.scribe.**print_rtf**(*listing*, *background_colour=None*)
Prints an RTF document containing the colourized code

Parameters **listing** – Listing object containing code to print

purdy.scribe.**print_tokens**(*listing*, *colour=True*)
Prints each line in a *purdy.content.Code* object with a coloured background, then prints the parsed tokens inside that line

Parameters

- **listing** – purdy.content.Listing object containing code to print
- **colour** – set to True to print out using ANSI colour. Defaults to True

TUI Screen (purdy.tui.iscreen.py)

This module is an Urwid code viewer concrete implementation of a Screen. It is constructed by *purdy.ui.Screen* depending on its factory.

class purdy.iscreen.tui.iscreen.**BaseWindow**(*iscreen*, **args*, ***kwargs*)

keypress(*size*, *key*)
Pass the keypress to the widget in focus. Unhandled 'up' and 'down' keys may cause a focus change.

class purdy.iscreen.tui.iscreen.**ConcreteCodeBox**(*proxy_code_box*)
purdy.ui.CodeBox represents a box of code in the *purdy.ui.Screen*. This is an Urwid implementation of it.

Parameters **proxy_code_box** – the *purdy.ui.CodeBox* representing what is to be built.

class purdy.iscreen.tui.iscreen.**ConcreteTwinCodeBox**(*proxy*)
purdy.ui.TwinCodeBox represents two boxes of code in the *purdy.ui.Screen*, this is an Urwid implementation of it.

Parameters **proxy** – the *purdy.ui.TwinCodeBox* representing what is to be built.

class purdy.iscreen.tui.iscreen.**HelpDialog**(*parent*)

keypress(*size*, *key*)
Pass the keypress to the widget in focus. Unhandled 'up' and 'down' keys may cause a focus change.

class purdy.iscreen.tui.iscreen.**TUIScreen**(*parent_screen*)
Concrete, Urwid based implementation of a screen.

Parameters **parent_screen** – *purdy.ui.Screen* object that is creating this concrete implementation

run()
Calls the main display event loop. Does not return until the UI exits.

Wigets (purdy.wigets.py)

Widgets for displaying. These are called and managed through the Screen classes in *purdy.ui*.

class purdy.iscreen.tui.widgets.**CodeWidget**(*screen*, *auto_scroll*)
Urwid widget that displays the code. This implements the methods of purdy.content.RenderHook and is registered against a *purdy.ui.CodeBox* and purdy.content.Listing. As changes are made to the listing they will be rendered this widget.

The widget wraps an urwid ListBox, with each line in the box being a line of code. It also provides indiciators on the right side of the screen as to whether there is content above or below the current screen. If the parent Screen implementation has multiple instances of this class active, the scroll area will also indicate which code box is focused.

The up and down arrows as well as the page-up and page-down buttons are supported. If there are multiple code widgets, tab key will change the focus.

class purdy.iscreen.tui.widgets.**DividingLine**

class purdy.iscreen.tui.widgets.**ScrollingIndicator**

class purdy.iscreen.tui.widgets.**ScrollingListBox**(*scroll_indicator*, **args*, ***kwargs*)

keypress(*size*, *key*)
Move selection through the list elements scrolling when necessary. Keystrokes are first passed to widget in focus in case that widget can handle them.

Keystrokes handled by this widget are: 'up' up one line (or widget) 'down' down one line (or widget) 'page up' move cursor up one listbox length (or widget) 'page down' move cursor down one listbox length (or widget)

class purdy.iscreen.tui.widgets.**TwinContainer**(*widget_list*, *dividechars=0*, *focus_column=None*, *min_width=1*, *box_columns=None*)

Virtual Screen (purdy.virtual.iscreen.py)

This module mimics a code viewer, running through the requested actions and making the final result available.

class purdy.iscreen.virtual.iscreen.**VirtualScreen**(*parent_screen*)
Concrete, Urwid based implementation of a screen.

Parameters **parent_screen** – *purdy.ui.Screen* object that is creating this concrete implementation

run()
Runs the actions on the code listings.

6.2 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

R

S

T

V

W