Python で Curses プログラミング

リリース *3.12.3*

Guido van Rossum and the Python development team

5 月 03, 2024

目次

1 curses ってなに? 2
1.1 Python の curses module 2
2 curses アプリケーションの起動と終了 3
3 ウィンドウとパッド 4
4 テキストの表示 6
4.1 属性とカラー 7
5 ユーザ入力 8
6 より多くの情報 10

著者

A.M. Kuchling, Eric S. Raymond

リリース

2.04

概要

このドキュメントでは `curses` 拡張モジュールでテキストモードディスプレイを制御する方法について記述します。

1 curses ってなに?

curses ライブラリは、VT100s や Linux コンソール、さまざまなプログラムが提供するエミュレーション端末といったテキストベースの端末（ターミナル）のために、端末に依存しないスクリーン描画や、キーボードの処理を提供します。端末はカーソルの移動や、画面のスクロール、領域の消去といった共通の操作を行うための様々な制御コードをサポートします。端末の種類によって大きく異なる制御コードを使うことがあり、しばしば独特の癖があります。

グラフィカルなディスプレイが当たり前の今となっては、「なんでわざわざ？」と疑問に思うかもしれません。確かに文字表示端末は時代遅れな技術ではありますが、ニッチな領域が存在していて、意匠を凝らすことができるため、いまだに価値のあるものとなっています。そのような領域の一つは、例えば X サーバーを持たない組み込みの Unix です。ほかにも、OS のインストーラや、カーネル設定などのツールで、これらはグラフィカルサポートが利用可能になる前に動作する必要があります。

The curses library provides fairly basic functionality, providing the programmer with an abstraction of a display containing multiple non-overlapping windows of text. The contents of a window can be changed in various ways---adding text, erasing it, changing its appearance---and the curses library will figure out what control codes need to be sent to the terminal to produce the right output. curses doesn't provide many user-interface concepts such as buttons, checkboxes, or dialogs; if you need such features, consider a user interface library such as Urwid.

curses ライブラリは元々 BSD UNIX 向けに書かれました; 後の AT&T から出た Unix System V バージョンで多くの機能と新機能が追加されました。BSD curses はいまやメンテナンスされておらず、これは AT&T インターフェースのオープンソース実装である ncurses にとって置き換えられました。Linux や FreeBSD のようなオープンソース Unix を利用している場合は、おそらくシステムは ncurses を利用しています。現在のほとんどの商用 Unix は System V のコードを基にしているため、ここで述べる全ての関数が利用できるはずです。しかし、古いバージョンの curses を持ついくつかのプロプライエタリ Unix は全てには対応していないでしょう。

The Windows version of Python doesn't include the `curses` module. A ported version called UniCurses is available.

1.1 Python の curses module

The Python module is a fairly simple wrapper over the C functions provided by curses; if you're already familiar with curses programming in C, it's really easy to transfer that knowledge to Python. The biggest difference is that the Python interface makes things simpler by merging different C functions such as `addstr()`, `mvaddstr()`, and `mvwaddstr()` into a single `addstr()` method. You'll see this covered in more detail later.

この HOWTO は curses と Python を使ってテキストプログラムを書くための入門記事です。curses API に対する完全な解説をすることは意図していません; その目的のためには Python ライブラリガイドの ncurses 節と C 言語マニュアルの ncurses のページを参照してください。とはいえ、この文章は基本的な考えを提供してくれるでしょう。

2 curses アプリケーションの起動と終了

Before doing anything, curses must be initialized. This is done by calling the `initscr()` function, which will determine the terminal type, send any required setup codes to the terminal, and create various internal data structures. If successful, `initscr()` returns a window object representing the entire screen; this is usually called `stdscr` after the name of the corresponding C variable.

```
import curses
stdscr = curses.initscr()
```

Usually curses applications turn off automatic echoing of keys to the screen, in order to be able to read keys and only display them under certain circumstances. This requires calling the `noecho()` function.

```
curses.noecho()
```

通常アプリケーションはまた、Enter キーを押すことなく、キーに対してすぐに反応する必要があります; これは cbreak モードと呼ばれ、通常の入力がバッファされるモードと逆に動作します。

```
curses.cbreak()
```

端末は通常、カーソルキーや Page Up や Home といった操作キーなどの特別なキーをマルチバイトエスケープシーケンスとして返します。それらのシーケンスを想定して対応する処理を行うアプリケーションを書けるように、curses はそれを `curses.KEY_LEFT` のような特別な値を返して行ってくれます。curses にその仕事をさせるには、キーパッドモードを有効にする必要があります。

```
stdscr.keypad(True)
```

curses アプリケーションを終了させるのは起動よりも簡単です。以下を呼び出す必要があります:

```
curses.nocbreak()
stdscr.keypad(False)
curses.echo()
```

curses に親和性の高い設定をもとに戻します。そして、`endwin()` 関数を呼び出し、端末を通常の操作モードに復旧します。

```
curses.endwin()
```

curses アプリケーションをデバッグするときの一般的な問題は、アプリケーションが端末を以前の状態に復旧することなく異常終了したときに端末がめちゃめちゃになることです。Python ではこの問題はコードにバグがあって、捕捉できない例外が発生したときによく起きます。タイプしたキーはもはやエコーされません、例えば、シェルを使うのが難しくなります。

Python では、これらの複雑さを避けデバッグをより容易にするために、`curses.wrapper()` 関数をインポートして、このように使います:

```
from curses import wrapper

def main(stdscr):
    # Clear screen
    stdscr.clear()

    # This raises ZeroDivisionError when i == 10.
    for i in range(0, 11):
        v = i-10
        stdscr.addstr(i, 0, '10 divided by {} is {}'.format(v, 10/v))

    stdscr.refresh()
    stdscr.getkey()

wrapper(main)
```

The `wrapper()` function takes a callable object and does the initializations described above, also initializing colors if color support is present. `wrapper()` then runs your provided callable. Once the callable returns, `wrapper()` will restore the original state of the terminal. The callable is called inside a `try...except` that catches exceptions, restores the state of the terminal, and then re-raises the exception. Therefore your terminal won't be left in a funny state on exception and you'll be able to read the exception's message and traceback.

3 ウィンドウとパッド

Windows are the basic abstraction in curses. A window object represents a rectangular area of the screen, and supports methods to display text, erase it, allow the user to input strings, and so forth.

The `stdscr` object returned by the `initscr()` function is a window object that covers the entire screen. Many programs may need only this single window, but you might wish to divide the screen into smaller windows, in order to redraw or clear them separately. The `newwin()` function creates a new window of a given size, returning the new window object.

```
begin_x = 20; begin_y = 7
height = 5; width = 40
win = curses.newwin(height, width, begin_y, begin_x)
```

Note that the coordinate system used in curses is unusual. Coordinates are always passed in the order *y,x*, and the top-left corner of a window is coordinate (0,0). This breaks the normal convention for handling coordinates where the *x* coordinate comes first. This is an unfortunate difference from most other computer applications, but it's been part of curses since it was first written, and it's too late to change things now.

Your application can determine the size of the screen by using the `curses.LINES` and `curses.COLS` variables to obtain the *y* and *x* sizes. Legal coordinates will then extend from `(0,0)` to `(curses.LINES - 1, curses.COLS - 1)`.

When you call a method to display or erase text, the effect doesn't immediately show up on the display. Instead you must call the `refresh()` method of window objects to update the screen.

This is because curses was originally written with slow 300-baud terminal connections in mind; with these terminals, minimizing the time required to redraw the screen was very important. Instead curses accumulates changes to the screen and displays them in the most efficient manner when you call `refresh()`. For example, if your program displays some text in a window and then clears the window, there's no need to send the original text because they're never visible.

In practice, explicitly telling curses to redraw a window doesn't really complicate programming with curses much. Most programs go into a flurry of activity, and then pause waiting for a keypress or some other action on the part of the user. All you have to do is to be sure that the screen has been redrawn before pausing to wait for user input, by first calling `stdscr.refresh()` or the `refresh()` method of some other relevant window.

A pad is a special case of a window; it can be larger than the actual display screen, and only a portion of the pad displayed at a time. Creating a pad requires the pad's height and width, while refreshing a pad requires giving the coordinates of the on-screen area where a subsection of the pad will be displayed.

```
pad = curses.newpad(100, 100)
# These loops fill the pad with letters; addch() is
# explained in the next section
for y in range(0, 99):
    for x in range(0, 99):
        pad.addch(y,x, ord('a') + (x*x+y*y) % 26)

# Displays a section of the pad in the middle of the screen.
# (0,0) : coordinate of upper-left corner of pad area to display.
# (5,5) : coordinate of upper-left corner of window area to be filled
#         with pad content.
# (20, 75) : coordinate of lower-right corner of window area to be
#          : filled with pad content.
pad.refresh( 0,0, 5,5, 20,75)
```

The `refresh()` call displays a section of the pad in the rectangle extending from coordinate (5,5) to coordinate (20,75) on the screen; the upper left corner of the displayed section is coordinate (0,0) on the pad. Beyond that difference, pads are exactly like ordinary windows and support the same methods.

If you have multiple windows and pads on screen there is a more efficient way to update the screen and prevent annoying screen flicker as each part of the screen gets updated. `refresh()` actually does two things:

1) それぞれのウィンドウの `noutrefresh()` メソッドを呼び出して、配下にある、スクリーンの望ましい状態を表すデータ構造を更新します。

2) `doupdate()` 関数を呼び出して、データ構造に記録された望ましい状態に合致するように、物理スクリーンを更新します。

Instead you can call `noutrefresh()` on a number of windows to update the data structure, and then call `doupdate()` to update the screen.

4 テキストの表示

From a C programmer's point of view, curses may sometimes look like a twisty maze of functions, all subtly different. For example, `addstr()` displays a string at the current cursor location in the `stdscr` window, while `mvaddstr()` moves to a given y,x coordinate first before displaying the string. `waddstr()` is just like `addstr()`, but allows specifying a window to use instead of using `stdscr` by default. `mvwaddstr()` allows specifying both a window and a coordinate.

幸運にも、Python インターフェースはこれらの詳細を全て隠蔽してくれます。`stdscr` は他のものと同様のウィンドウオブジェクトであり、`addstr()` のようなメソッドは複数の引数形式を許容してくれます。通常それらは 4 つの形式です。

形式	説明
str または *ch*	文字列 *str* または文字 *ch* を現在位置に表示します
str または *ch*, *attr*	文字列 *str* または文字 *ch* を属性 *attr* を利用して現在位置に表示します
y, *x*, *str* または *ch*	ウィンドウ内の位置 *y,x* に移動し *str* または *ch* を表示します
y, *x*, *str* または *ch*, *attr*	ウィンドウ内の位置 *y,x* に移動し属性 *attr* を利用して *str* または *ch* を表示します

属性によって表示するテキストをハイライトすることができます、ボールド体、アンダーライン、反転、カラーなど。より詳しくは次の小節で説明します。

The `addstr()` method takes a Python string or bytestring as the value to be displayed. The contents of bytestrings are sent to the terminal as-is. Strings are encoded to bytes using the value of the window's `encoding` attribute; this defaults to the default system encoding as returned by `locale.getencoding()`.

The `addch()` methods take a character, which can be either a string of length 1, a bytestring of length 1, or an integer.

Constants are provided for extension characters; these constants are integers greater than 255. For example, `ACS_PLMINUS` is a +/- symbol, and `ACS_ULCORNER` is the upper left corner of a box (handy for drawing borders). You can also use the appropriate Unicode character.

ウィンドウは最後の操作の後のカーソル位置を覚えているため、*y,x* 座標をうっかり忘れてしまっても、文字列や文字は最後の操作位置に表示されます。`move(y,x)` メソッドでカーソルを移動させることもできます。常に点滅するカーソルを表示する端末もあるため、カーソルが特定の位置にいることを保証して注意が反れないようにしたいと思うかもしれません; ランダムに見える位置でカーソルが点滅すると面を食らってしまいます。

If your application doesn't need a blinking cursor at all, you can call `curs_set(False)` to make it invisible. For compatibility with older curses versions, there's a `leaveok(bool)` function that's a synonym for `curs_set()`. When *bool* is true, the curses library will attempt to suppress the flashing cursor, and you won't need to worry about leaving it in odd locations.

4.1 属性とカラー

Characters can be displayed in different ways. Status lines in a text-based application are commonly shown in reverse video, or a text viewer may need to highlight certain words. curses supports this by allowing you to specify an attribute for each cell on the screen.

属性は整数値で、それぞれのビットが異なる属性を表わします。複数の属性ビットをセットしてテキストの表示を試みることができますが、curses は全ての組み合わせが利用可能であるかや視覚的に区別できるかどうかは保証してくれません、それらは利用している端末の能力に依存しているため、最も安全なのは、最も一般的に利用可能な属性を設定する方法です、ここに列挙します。

属性	説明
A_BLINK	テキストを点滅
A_BOLD	高輝度またはボールドテキスト
A_DIM	低輝度テキスト
A_REVERSE	反転テキスト
A_STANDOUT	利用できる最良のハイライトモード
A_UNDERLINE	下線付きテキスト

つまり、反転するステータスラインを画面の最上部に表示するには、コードをこうします:

```
stdscr.addstr(0, 0, "Current mode: Typing mode",
              curses.A_REVERSE)
stdscr.refresh()
```

curses ライブラリはカラー機能を提供している端末でのカラーもサポートしています。そんな端末の中で最も一般的なものは Linux コンソールで、color xterm もそれに続きます。

To use color, you must call the `start_color()` function soon after calling `initscr()`, to initialize the default color set (the `curses.wrapper()` function does this automatically). Once that's done, the `has_colors()` function returns TRUE if the terminal in use can actually display color. (Note: curses uses the American spelling 'color', instead of the Canadian/British spelling 'colour'. If you're used to the British spelling, you'll have to resign yourself to misspelling it for the sake of these functions.)

The curses library maintains a finite number of color pairs, containing a foreground (or text) color and a background color. You can get the attribute value corresponding to a color pair with the `color_pair()` function; this can be bitwise-OR'ed with other attributes such as `A_REVERSE`, but again, such combinations are not guaranteed to work on all terminals.

例として、テキスト行をカラーペア 1 を使って表示します:

```
stdscr.addstr("Pretty text", curses.color_pair(1))
stdscr.refresh()
```

As I said before, a color pair consists of a foreground and background color. The `init_pair(n, f, b)` function changes the definition of color pair n, to foreground color f and background color b. Color pair

0 is hard-wired to white on black, and cannot be changed.

Colors are numbered, and `start_color()` initializes 8 basic colors when it activates color mode. They are: 0:black, 1:red, 2:green, 3:yellow, 4:blue, 5:magenta, 6:cyan, and 7:white. The `curses` module defines named constants for each of these colors: `curses.COLOR_BLACK`, `curses.COLOR_RED`, and so forth.

やってみましょう。カラー 1 を白背景に赤に変更してみましょう、こうして呼び出します:

```
curses.init_pair(1, curses.COLOR_RED, curses.COLOR_WHITE)
```

カラーペアを変更するときには、既に表示された任意のテキストが利用するカラーペアを新しい色に変更します。新しいテキストをこの色で使うこともできます:

```
stdscr.addstr(0,0, "RED ALERT!", curses.color_pair(1))
```

Very fancy terminals can change the definitions of the actual colors to a given RGB value. This lets you change color 1, which is usually red, to purple or blue or any other color you like. Unfortunately, the Linux console doesn't support this, so I'm unable to try it out, and can't provide any examples. You can check if your terminal can do this by calling `can_change_color()`, which returns `True` if the capability is there. If you're lucky enough to have such a talented terminal, consult your system's man pages for more information.

5 ユーザ入力

The C curses library offers only very simple input mechanisms. Python's `curses` module adds a basic text-input widget. (Other libraries such as Urwid have more extensive collections of widgets.)

ウィンドウから入力を得るための 2 つのメソッドがあります。

- `getch()` refreshes the screen and then waits for the user to hit a key, displaying the key if `echo()` has been called earlier. You can optionally specify a coordinate to which the cursor should be moved before pausing.
- `getkey()` does the same thing but converts the integer to a string. Individual characters are returned as 1-character strings, and special keys such as function keys return longer strings containing a key name such as `KEY_UP` or `^G`.

It's possible to not wait for the user using the `nodelay()` window method. After `nodelay(True)`, `getch()` and `getkey()` for the window become non-blocking. To signal that no input is ready, `getch()` returns `curses.ERR` (a value of -1) and `getkey()` raises an exception. There's also a `halfdelay()` function, which can be used to (in effect) set a timer on each `getch()`; if no input becomes available within a specified delay (measured in tenths of a second), curses raises an exception.

The `getch()` method returns an integer; if it's between 0 and 255, it represents the ASCII code of the key pressed. Values greater than 255 are special keys such as Page Up, Home, or the cursor keys. You can compare the value returned to constants such as `curses.KEY_PPAGE`, `curses.KEY_HOME`, or `curses.KEY_LEFT`. The main loop of your program may look something like this:

```
while True:
    c = stdscr.getch()
    if c == ord('p'):
        PrintDocument()
    elif c == ord('q'):
        break  # Exit the while loop
    elif c == curses.KEY_HOME:
        x = y = 0
```

The `curses.ascii` module supplies ASCII class membership functions that take either integer or 1-character string arguments; these may be useful in writing more readable tests for such loops. It also supplies conversion functions that take either integer or 1-character-string arguments and return the same type. For example, `curses.ascii.ctrl()` returns the control character corresponding to its argument.

There's also a method to retrieve an entire string, `getstr()`. It isn't used very often, because its functionality is quite limited; the only editing keys available are the backspace key and the Enter key, which terminates the string. It can optionally be limited to a fixed number of characters.

```
curses.echo()             # Enable echoing of characters

# Get a 15-character string, with the cursor on the top line
s = stdscr.getstr(0,0, 15)
```

The `curses.textpad` module supplies a text box that supports an Emacs-like set of keybindings. Various methods of the `Textbox` class support editing with input validation and gathering the edit results either with or without trailing spaces. Here's an example:

```
import curses
from curses.textpad import Textbox, rectangle

def main(stdscr):
    stdscr.addstr(0, 0, "Enter IM message: (hit Ctrl-G to send)")

    editwin = curses.newwin(5,30, 2,1)
    rectangle(stdscr, 1,0, 1+5+1, 1+30+1)
    stdscr.refresh()

    box = Textbox(editwin)

    # Let the user edit until Ctrl-G is struck.
    box.edit()

    # Get resulting contents
    message = box.gather()
```

さらなる詳細についてはライブラリのドキュメント `curses.textpad` を参照してください。

6 より多くの情報

この HOWTO ではいくつかの進んだ話題、スクリーンスクレイピングや xterm インスタンスからマウスイベントを捉えるなど、については扱っていません。しかし、Python の `curses` モジュールのライブラリページはいまやかなり充実しています。次はこれを見るべきです。

If you're in doubt about the detailed behavior of the curses functions, consult the manual pages for your curses implementation, whether it's ncurses or a proprietary Unix vendor's. The manual pages will document any quirks, and provide complete lists of all the functions, attributes, and ACS_* characters available to you.

Because the curses API is so large, some functions aren't supported in the Python interface. Often this isn't because they're difficult to implement, but because no one has needed them yet. Also, Python doesn't yet support the menu library associated with ncurses. Patches adding support for these would be welcome; see the Python Developer's Guide to learn more about submitting patches to Python.

- Writing Programs with NCURSES: a lengthy tutorial for C programmers.
- The ncurses man page
- The ncurses FAQ
- "Use curses... don't swear": curses または Urwid を使って端末を制御する PyCon 2013 講演ビデオです。
- "Console Applications with Urwid": video of a PyCon CA 2012 talk demonstrating some applications written using Urwid.