Python で Curses プログラミング

リリース *2.7ja1*

Guido van Rossum
Fred L. Drake, Jr., editor

2011 年 12 月 25 日

Python Software Foundation
Email: docs@python.org

目次

Author A.M. Kuchling, Eric S. Raymond

Release 2.03

概要

このドキュメントでは Python 2.x でテキストモードプログラムを表示をコントロールする `curses` 拡張モジュールを利用して書く方法について記述します。

1 curses ってなに?

curses ライブラリは、テキストベースの端末、VT100s や Linux コンソールや xterm や rxvt などの X11 プログラムから提供される端末エミュレータなどの端末に依存せずに、スクリーンに描画やキーボード処理する機能を供給します。表示端末は様々な制御コードをサポートしていて、カーソル移動、画面のスクロール、領域の消去などの一般的な操作を実行できます。端末によって大きく異なるコードを使うことがあるので、しばしば独自の癖があります。

X ディスプレイの世界では、「どうしてこんな面倒なことを?」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、未だ価値あるものとなっています。その領域は例えば X サーバを持たない、小さな机上用コンピュータや組み込みの Unix です。他のものとしては OS インストーラーやカーネル設定ツールなどで、これらは X が利用できるようになる前に動作する必要があります。

curses ライブラリは異なる端末の詳細を全て隠蔽し、プログラマに対して、重なり合わない複数のウィンドウを持つ抽象化されたディスプレイを提供します。ウィンドウの内容は様々な方法で変更されます– テキストの追加、消去、外観の変更 – そして curses ライブラリは送るべき制御コードを自動的に理解し、正しい出力を生成してくれます。

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、ここれ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。

curses モジュールの Windows 移植は作られていません。Windows プラットフォーム上では、Fredrik Lundh が書いた Console モジュールを試してみてください。Console モジュールはカーソル位置を指定してテキスト出力することができ、マウス、キーボード入力の両方を完全にサポートしていて、http://effbot.org/zone/console-index.htm から入手できます。

1.1 Python の curses module

Python モジュールは curses が提供する C 関数に対するまったく単純なラッパーです; 既に C での curses プログラミングに慣れ親しんでいるなら、その知識を Python に持ち込むのは実に簡単です。最大の違いは Python インターフェースが異なる関数 `addstr()`,`mvaddstr()`,`mvwaddstr()` を一つのメソッド `addstr()` メソッドに統合して単純化していることです。このことは後でより詳しく扱います。

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簡単な入門記事です。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 ncurses の C 言語マニュアルページを参照してください。しかし、この文章が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

何をするにもまず、curses を初期化する必要があります。初期化は `initscr()` 関数を呼びだすことでできます、この関数は端末のタイプを決定し、必要とされるセットアップコードを端末に送り、様々な内部データ構造を作成します。成功すると、 `initscr()` は画面全体を表わすウィンドウオブジェクトを返します; これは、 `stdscr` と呼ばれます、この名前は C での対応する変数にちなんでいます。:

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

通常 curses アプリケーションは画面へのキーエコーを自動的に止めます、これはキーを読みとり特定状況下でのみで表示するためです。これには `noecho()` 関数を呼び出す必要があります。:

```
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(1)
```

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。curses に親和的な端末設定を元に戻すために以下を呼び出す必要があります:

```
curses.nocbreak(); stdscr.keypad(0); curses.echo()
```

そして、 `endwin()` 関数を呼び出し、端末設定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 `curses.wrapper` をインポートすることでこの複雑な問題を避け、デバッグを容易にすることができます。それは 呼び出し可能オブジェクトを引数にとることができる `wrapper()` 関数を提供します。これは上に述べた初期化をし、カラーサポートがあればカラーの初期化をします。そして、与えられた呼び出し可能オブジェクトを実行し、最終的に適切に初期化の逆操作 (deinitialization) を行います。呼び出し可能オブジェクトは例外を捕捉する try-catch 節内部で呼び出され、curses の初期化操作の逆操作を実行します、そして例外が上に渡されます。そうして端末は例外が発生しても端末はおかしな状態にならずにすみます。

3 ウィンドウとパッド

ウィンドウは curses での基本的な抽象概念です。ウィンドウオブジェクトは画面の長方形の領域を表わし、テキストの表示、消去、ユーザに文字列入力を許可などの様々なメソッドをサポートしています。

`initscr()` 関数によって返される `stdscr` オブジェクトはウィンドウオブジェクトで画面全体を扱います。多くのプログラムはこのウィンドウだけを必要としますが、画面を小さなウィンドウに分けてそれらを別々に再描画したり消去したいと思うかもしれません。`newwin()` 関数は新しいウィンドウを与えられたサイズで作成し、新しいウィンドウオブジェクトを返します。:

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

curses で利用されている座標形について、言っておくことがあります: 座標は常に *y,x* の順で渡し、ウィンドウの左上の座標を (0,0) とします。これは *x* 座標が最初にくる一般的な慣習には反しています。これは多くの計算機のアプリケーションにとって不幸な違いですが、curses が最初に書かれて以来そうだったので、今から変えるには遅すぎます。

メソッドを呼びだして、テキストを表示、消去するとき、その効果はディスプレイにすぐには現われません。それは curses が元々 300 ボーの端末接続を念頭にいれていたためです l それらの端末では画面の再描画時間を減らすことがとても重要です。このため curses は画面への変更を積み上げ、最も効率良い方法でそれらを表示します。例えば、プログラムがウィンドウにいくつかの文字を表示し、それからウィンドウを消去する場合、元々の文字を送信する必要はありません、それらはもう見られることはないのです。

従って curses に明示的にウィンドウの再描画を明示的に伝えてやる必要があります、それはウィンドウオブジェクトのメソッド `refresh()` を使うことでできます。実際には、これは curses を使ったプログラムをそれほど複雑にするものではありません。ほとんどのプログラムはせわしなく動いた後キーを押すなどのユーザからの動作を待ちます。あなたがするべきことはユーザからの入力を待つ前に単に `stdscr.refresh()` や関連するウィンドウの `refresh()` メソッドを呼び出して画面を再描画するだけです。

パッドはウィンドウの特別な場合を指します; それは実際に画面に表示されるものより大きくなることがあり、画面にはその一部だけが表示されます。パッドを作るのに必要なものはパッドの高さと幅だけです、一方でパッドの再描画には、表示される部分を指す画面上の領域の座標を指定する必要があります。:

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

#  Displays a section of the pad in the middle of the screen
pad.refresh( 0,0, 5,5, 20,75)
```

`refresh()` はパッドの一部、画面上の座標 (5,5) から (20,75) に及ぶ長方形領域を表示します; 表示される一

部の左上の角は パッドの座標 (0,0) です。この違いを除けば、パッドは通常のウィンドウと全く同じで、同じメソッドを持ちます。

複数のウィンドウとパッドが画面上にある場合、より効率的な方法があり、再描画時に画面をちらつかせるのを防いでくれます。各ウィンドウの `noutrefresh()` メソッドを利用して、画面の望ましい状態を表わすデータ構造をアップデートし、`doupdate()` 関数で望む状態に合った物理的な画面に変更します。通常の `refresh()` メソッドは自身の動作時の最後に `doupdate()` を呼び出します。

4 テキストの表示

C 言語のプログラマの視点からすると、curses はしばしば、混乱を招きやすい、微妙に似た関数を持っています。例えば `addstr()` は `stdscr` ウィンドウの現在のカーソル位置の文字列を表示し、一方で `mvaddstr()` は与えられた y,x 座標にまず移動して、文字列を表示します。`waddstr()` はほとんど `addstr()` に似ていますが、デフォルトの `stdscr` を使う代わりに使うウィンドウを指定できます。`mvwaddstr()` も同様です。

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。

`addstr()` 関数は Python 文字列を引数にとり、表示します。一方 `addch()` 関数は文字を引数にとります、引数は長さ 1 の文字列か整数のどちらでもかまいません。文字列の場合には、表示する文字は 0 から 255 の間に制限されます。SVr4 curses は文字拡張のための定数を提供しています; それらの定数は 255 より大きい整数です。例えば `ACS_PLMINUS` は +/- 記号で `ACS_ULCORNER` はボックスの左上角です (境界を描くのに便利です)。

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ちらつくとと面を食らってしまいます。

アプリケーションがちらつくカーソルを全く必要としない場合、`curs_set(0)` を呼び出してカーソル見えなくすることができます。同じことを、古い curses バージョンに対する互換性を保ちつつ行うために `leaveok(bool)` 関数があります。*bool* が true の場合、curses ライブラリは点滅するカーソルを外に出さなくするので、変な場所に現われるのを心配する必要は無くなります。

4.1 属性とカラー

文字は様々な方法で表示することができます。テキストベースアプリケーションでのステータスラインは通常反転して表示されます; テキストビュアーは特定の単語をハイライトする必要があるかもしれません。curses は属性を画面上の各セルに対して指定することで、それをサポートします。

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。

カラーを利用するには、`initscr()` を呼び出したすぐ後に `start_color()` 関数を呼びし、デフォルトカラーセットを初期化しなければいけません (`curses.wrapper.wrapper()` 関数はこれを自動的に行ないます)。一旦それを行えば、`has_colors()` 関数は、端末が実際にカラーを表示できる場合に TRUE を返します。(ノート: curses はカナダ/イギリスつづりに 'colour' ではなくアメリカつづりの 'color' を使います。イギリスつづりを使っている場合には、これらの関数のミススペルを修正する必要があります。)

curses ライブラリは有限の数の、フォラグラウンド (またはテキスト) カラーとバックグラウンドカラーペアを保持します。カラーペアに対応する属性値は `color_pair()` 関数で取得できます; これは `A_REVERSE` のような他の属性と OR 論理演算組み合わせることができます、ただし、繰り返しになりますが、組み合わせは全ての端末で保証されていません。

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()
```

前に述べたように、カラーペアはフォアグラウンドカラーとバックグラウンドカラーから構成されています。`start_color()` はカラーモードを有効にした場合 8 の基本カラーを初期化します。基本カラーは: 0:black,

1:red, 2:green, 3:yellow, 4:blue, 5:magenta, 6:cyan, 7:white です。curses モジュールは各名前に対する名前付き定数を定義しています: `curses.COLOR_BLACK`, `curses.COLOR_RED`, など。

`init_pair(n, f, b)` 関数はカラーペア *n* の定義をフォアグラウンド f バックグラウンド b に変更します。カラーペア 0 は黒背景に白で組み込まれていて変更できません。

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) )
```

凝ったターミナルは実際の色の定義を与えられた RGB 値に変更することができます。これによってふつうは赤であるカラー 1 を紫や青など好きな色に変更されます。不幸にも Linux コンソールはこれをサポートしていません、そのためこの機能は試すことも例を出すこともできません。`can_change_color()` を呼び出すことで端末が使えるのか調べることができ、対応していれば、TRUE を返します。もしそのような telnet 端末を持つ幸運に巡りあえたなら、より多くの情報の情報を得るためにシステムの man ページを参照することを勧めます。

5 ユーザ入力

curses ライブラリは自身で単純なインプット機構を備えています。Python はテキスト入力ウィジェットを追加して、いくつかの足りない部分を補っています。

ウィンドウに対する入力を得るための最も一般的な方法はウィンドウの `getch()` メソッドを利用することです。`getch()` はユーザが待機してキーを打つのを待ち、`echo()` が先に呼ばれていれば、それを表示します。オプションとして待機する前にカーソルが移動するべき座標を指定することができます。

この挙動は `nodelay()` メソッドで変更することができます。`nodelay(1)` の後、 ウィンドウに対する `getch()` はノンブロッキングになり、入力が準備されていないときには `curses.ERR` (-1 の値) を返します。`halfdelay()` 関数もあり、(事実上) 各 `getch()` に対してタイマーを設定するのに使うことができます; 指定したディレイの間に (10 分の 1 秒単位で測られます) 入力が得られなかった場合 curses は例外を送出します。

`getch()` メソッドは整数を返します; もしそれが 0 から 255 までなら、それは押されたキーの ASCII コードを表わします。255 より大きな値は Page Up, Home またはカーソルキーのような特別なキーです。返された値を `curses.KEY_PPAGE`, `curses.KEY_HOME` または `curses.KEY_LEFT` のような定数と比較することが可能です。通常プログラムのメインループはこのようになります:

```
while 1:
    c = stdscr.getch()
    if c == ord('p'): PrintDocument()
```

```
    elif c == ord('q'): break  # Exit the while()
    elif c == curses.KEY_HOME: x = y = 0
```

curses.ascii モジュールは ASCII クラスのメンバーシップ関数を提供し、整数と 1 文字引数のどちらもとることができます; これらは独自のコマンドインタプリタに対するより読み易いテストを書くのに便利でしょう。整数と 1 文字引数のどちらもとることができ、引数と同じ型を返す変換関数も提供します。例えば curses.ascii.ctrl() は引数に応じた型で制御文字を返します。

文字列全体を取得するメソッド getstr() もあります。これは頻繁に使われるものではありません、なぜならこの機能はとても制限的なものだからです; 利用可能な編集キーはバックスペースと Enter キーの文字列を終了させるものだけです。オプションとして文字列の長さを固定長に限定することもできます。:

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

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

Python の curses.textpad モジュールはよりよいものを提供します。これを使うことで、ウィンドウ を Emacs のようなキーバインドをサポートするテキストボックスにすることができます。Textbox クラスの様々なメソッドが入力の検証つきの編集をサポートシ前後のスペースつき、または無しで編集結果を収集します。詳しくは curses.textpad のライブラリドキュメントを参照して下さい。

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。

ncurses のあらゆるエントリポイントの詳細な挙動について疑問があれば、curses 実装が ncurses か、プロプライエタリな Unix ベンダーのものかによらず curses 実装のマニュアルページをみることを参照して下さい。マニュアルページにはあらゆる癖がドキュメントにされていて、全ての関数、属性、ACS_* 文字の完全なリストが提供されています。

curses API は巨大なので、Python インターフェースではいくつかの関数はサポートされていません、ですがそれは必要としている人がいまのところいないためです。気兼ねなく、足りないものを追加してパッチを提出して下さい。また、ncurese に関連したメニューやパネルライブラリのサポートも行なわれていません; 気兼ねなく追加して下さい。

面白い小さなプログラムを書いたら、新たなデモとして気兼ねなく提出して下さい。私達は常により多くのデモ使うことができます!

The ncurses FAQ: http://invisible-island.net/ncurses/ncurses.faq.html