Programação em Curses com Python

Release 3.11.11

Guido van Rossum and the Python development team

dezembro 07, 2024

Python Software Foundation
Email: docs@python.org

Sumário

Autor
A.M. Kuchling, Eric S. Raymond

Versão
2.04

Resumo

Este documento descreve como usar o módulo de extensão `curses` para controlar visualização em modo texto.

1 O que é curses?

A biblioteca curses fornece formas que facilitam a impressão no terminal e o tratamento de entrada do teclado para interfaces baseados em texto; tais como interfaces produzidas para terminais incluindo VT100s, o console Linux, e terminais fornecidos por vários programas. Terminais visuais suportam vários códigos de controle para executar várias operações comuns como mover o cursor, apagar áreas e rolagem de tela. Diferentes terminais usam uma gama de diferentes códigos, e frequentemente têm suas próprias peculiaridades.

No mundo dos displays gráficos, uma pergunta pode vir à tona "por que isso, jovem?". É verdade que os terminais de exibição de caracteres são uma tecnologia obsoleta, mas há nichos nos quais a capacidade de fazer coisas sofisticadas com eles continua sendo valorizada. Um nicho são os programas de small-footprint ou os Unixes embarcados, os quais não rodas um servidor X, ou seja, não possuem interface gráfica. Outro nicho são o de ferramentas como instaladores de sistemas operacionais e configurações de kernels que devem rodar antes que qualquer suporte para interfaces gráficas esteja disponível.

A biblioteca curses fornece funcionalidade bastante básica, proporcionando ao programador uma abstração de um monitor contendo janelas de texto não sobrepostas. Os conteúdos de uma janela podem ser modificados de diversas formas — adicionando texto, apagando-o, modificando sua aparência — e a biblioteca curses irá descobrir quais códigos de controle precisam ser enviados ao terminal para produzir a saída correta. A biblioteca curses não fornece muitos conceitos de interface de usuário como botões, caixas de seleção ou diálogos; se você necessitar dessas funcionalidades, considere uma biblioteca de interface de usuário como Urwid.

A biblioteca curses foi originalmente escrita para BSD Unix; as versões mais recentes do System V do Unix da AT&T adicionaram muitos aprimoramentos e novas funções. BSD curses não é mais mantida, tendo sido substituída por ncursess, que é uma implementação de código aberto da interface da AT&T. Se você estiver usando um sistema operacional de código aberto baseado em Unix, tal como Linux ou FreeBSD, seu sistema provavelmente usa ncurses. Uma vez que a maioria das versões comerciais do Unix são baseadas no código do sistema V, todas as funções descritas aqui provavelmente estarão disponíveis. No entanto, as versões antigas de curses carregadas por alguns Unixes proprietários podem não suportar tudo.

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

1.1 O módulo curses de Python

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.

Este documento é uma introdução à escrita de programas em modo texto com curses e Python. Isto não pretende ser um guia completo da API curses; para isso, veja a seção ncurses no guia da biblioteca Python, e o manual de ncurses. Isto, no entanto, lhe dará uma ideia básica.

2 Começando e terminando uma aplicação curses

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()
```

Geralmente aplicações curses desativam o envio automático de teclas para a tela, para que seja possível ler chaves e somente exibi-las sob certas circunstâncias. Isto requer a chamada da função `noecho()`.

```
curses.noecho()
```

Aplicações também irão comumente precisar reagir a teclas instantaneamente, sem requisitar que a tecla Enter seja pressionada; isto é chamado de modo cbreak, ao contrário do modo de entrada buferizada usual.

```
curses.cbreak()
```

Terminais geralmente retornam teclas especiais, como as teclas de cursor ou de navegação como Page Up e Home, como uma sequência de escape mutibyte. Enquanto você poderia escrever sua aplicação para esperar essas sequências e processá-las de acordo, curses pode fazer isto para você, retornando um valor especial como `curses.KEY_LEFT`. Para permitir que curses faça esse trabalho, você precisará habilitar o modo keypad.

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

Finalizar uma aplicação curses é mais fácil do que iniciar uma. Você precisará executar:

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

para reverter as configurações de terminal amigáveis da curses. Então chame a função `endwin()` para restaurar o terminal para seu modo de operação original.

```
curses.endwin()
```

Um problema comum ao debugar uma aplicação curses é deixar seu terminal bagunçado quando a aplicação para sem restaurar o terminal ao seu estado anterior. Em Python isto comumente acontece quando seu código está com problemas e eleva uma exceção não capturada. As teclas não são mais enviadas para a tela quando você as digita, por exemplo, o que torna difícil utilizar o shell.

No Python você pode evitar essas complicações e fazer depurações de forma mais simples importando a função `curses.wrapper()` e utilizando-a desta forma:

```
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 Janelas e Pads

Janelas são a abstração mais básica em curses. Uma objeto janela representa uma área retangular da tela, e suporta métodos para exibir texto, apagá-lo, e permitir ai usuário inserir strings, e assim por diante.

O objeto `stdscr` retornado pela função `initscr()` é um objeto janela que cobre a tela inteira. Muitos programas podem precisar apenas desta janela única, mas você poderia desejar dividir a tela em janelas menores, a fim de redefini-las ou limpá-las separadamente. A função `newwin()` cria uma nova janela de um dado tamanho, retornando o novo objeto janela.

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

Note que o sistema de coordenadas utilizado na curses é incomum. Coordenadas geralmente são passadas na ordem *y,x*, e o canto superior-esquerdo da janela é a coordenada (0,0). Isto quebra a convenção normal para tratar coordenadas onde a coordenada *x* vem primeiro. Isto é uma diferença infeliz da maioria das aplicações computacionais, mas tem sido parte da curses desde que ela foi inicialmente escrita, e agora é tarde demais para mudar isso.

Sua aplicação pode determinar o tamanho da tela usando as variáveis `curses.LINES` e `curses.COLS` para obter os tamanhos *y* e *x*. Coordenadas legais estenderão de `(0,0)` a `(curses.LINES - 1, curses.COLS - 1)`.

Quando você chamar um método para exibir ou apagar texto, o efeito não é exibido imediatamente na tela. Em vez disso você deve chamar o método `refresh()` dos objetos janela para atualizar a tela.

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.

Um pad é um caso especial de janela; ele pode ser mais largo que a tela atual, e apenas uma porção do pad exibido por vez. Criar um pad requer sua altura e largura, enquanto atualizar o pad requer dar as coordenadas da área na tela onde uma subseção do pad será exibida.

```
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) Chama o método `noutrefresh()` de cada janela para atualizar uma estrutura de dados subjacente representando o estado desejado da tela.
2) Chama a função `doupdate()` para modificar a tela física para corresponder com o estado original na estrutura de dados.

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

4 Exibindo texto

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.

Felizmente, a interface do Python oculta todos estes detalhes. `stdscr` é um objeto de janela como qualquer outro, e métodos como `addstr()` aceitam múltiplas formas de argumentos.

Forma	Descrição
str ou *ch*	Mostra a string *str* ou caractere *ch* na posição atual.
str ou *ch*, *attr*	Mostra a string *str* ou caractere *ch*, usando o atributo *attr* na posição atual.
y, *x*, *str* ou *ch*	Move para a posição *y,x* dentro da janela, e exibe *str* ou *ch*
y, *x*, *str* ou *ch*, *attr*	Mover para a posição *y,x* dentro da janela, e exibir *str* ou *ch*, usando o atributo *attr*

Atributos permitem exibir texto de formas destacadas como negrito, sublinhado, código invertido, ou colorido. Elas serão explicadas em mais detalhes na próxima subseção.

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()`.

Os métodos `addch()` pegam um caractere, que pode ser tanto uma string de comprimento 1, um bytestring de comprimento 1, ou um inteiro.

Constantes são providas para caracteres de extensão; estas constantes são inteiros maiores que 255. Por exemplo, `ACS_PLMINUS` é um símbolo de +/-, e `ACS_ULCORNER` é o canto superior esquerdo de uma caixa (útil para desenhar bordas). Você pode usar o caractere Unicode apropriado.

Janelas lembram onde o cursor estava após a última operação, então se você omitir as coordenadas *y,x*, a string ou caractere serão exibidos onde quer que a última operação foi deixada. Você também pode mover o cursos com o método `move(y,x)`. Porque alguns terminais sempre exibem um cursos piscando, você pode querer garantir que o cursor está posicionado em algum local onde ele não será uma distração; pode ser confuso ter o cursor piscando em um local aparentemente aleatório.

Se sua aplicação não necessita de forma alguma de um cursor piscando, você pode chamar `curs_set(False)` para torná-lo invisível. Para compatibilidade com versões anteriores de curses, há a função `leaveok(bool)` que é um sinônimo para `curs_set()`. Quando *bool* é verdadeiro, a biblioteca curses tentará suprimir o cursor piscando, e você não precisará se preocupar ao deixá-lo em localizações incomuns.

4.1 Atributos e Cor

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.

An attribute is an integer, each bit representing a different attribute. You can try to display text with multiple attribute bits set, but curses doesn't guarantee that all the possible combinations are available, or that they're all visually distinct. That depends on the ability of the terminal being used, so it's safest to stick to the most commonly available attributes, listed here.

Atributo	Descrição
`A_BLINK`	Blinking text
`A_BOLD`	Extra bright or bold text
`A_DIM`	Half bright text
`A_REVERSE`	Reverse-video text
`A_STANDOUT`	The best highlighting mode available
`A_UNDERLINE`	Texto sublinhado

So, to display a reverse-video status line on the top line of the screen, you could code:

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

The curses library also supports color on those terminals that provide it. The most common such terminal is probably the Linux console, followed by color xterms.

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.

An example, which displays a line of text using color pair 1:

```
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.

Let's put all this together. To change color 1 to red text on a white background, you would call:

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

When you change a color pair, any text already displayed using that color pair will change to the new colors. You can also display new text in this color with:

```
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 Entrada de usuário

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.)

There are two methods for getting input from a window:

- `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

```

(continua na próxima página)

(continuação da página anterior)

```
# 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()
```

See the library documentation on `curses.textpad` for more details.

6 Para mais informações

This HOWTO doesn't cover some advanced topics, such as reading the contents of the screen or capturing mouse events from an xterm instance, but the Python library page for the `curses` module is now reasonably complete. You should browse it next.

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": video of a PyCon 2013 talk on controlling terminals using curses or Urwid.
- "Console Applications with Urwid": video of a PyCon CA 2012 talk demonstrating some applications written using Urwid.