

---

# Programação em Curses com Python

*Release 3.13.7*

**Guido van Rossum and the Python development team**

outubro 01, 2025

Python Software Foundation  
Email: docs@python.org

## Sumário

<b>1</b>	<b>O que é curses?</b>	<b>1</b>
1.1	O módulo curses de Python . . . . .	2
<b>2</b>	<b>Começando e terminando uma aplicação curses</b>	<b>2</b>
<b>3</b>	<b>Janelas e Pads</b>	<b>3</b>
<b>4</b>	<b>Exibindo texto</b>	<b>4</b>
4.1	Atributos e Cor . . . . .	5
<b>5</b>	<b>Entrada de usuário</b>	<b>6</b>
<b>6</b>	<b>Para mais informações</b>	<b>8</b>

---

### Autor

A.M. Kuchling, Eric S. Raymond

### Versão

2.04

### Resumo

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

## 1 O que é curses?

A biblioteca `curses` fornece formas que facilitam a impressão no terminal e o tratamento de entrada via teclado para interfaces de terminal baseadas em texto, tais como VT100s, o console Linux, e emuladores de 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 Unixes de baixa pegada de recursos ou embarcados, que não rodam um servidor X, ou seja, não possuem interface gráfica. Outro nicho é o de ferramentas como instaladores de

sistemas operacionais e configurações de kernels que podem precisar rodar antes que qualquer suporte a interfaces gráficas esteja disponível.

A biblioteca `curses` fornece funcionalidades bastante básicas, 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 precisar desses recursos, considere uma biblioteca de interface de usuário como [Urwid](#).

A biblioteca `curses` foi originalmente escrita para o BSD Unix; as versões mais recentes do System V do Unix da AT&T adicionaram muitos aprimoramentos e novas funções. A BSD `curses` não é mais mantida, tendo sido substituída pela `ncurses`, 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 System V, todas as funções descritas aqui provavelmente estarão disponíveis. No entanto, as versões antigas de `curses` carregadas por alguns Unixes proprietários podem não prover tudo.

A versão do Python para Windows não inclui o módulo `curses`. Uma versão portada chamada [UniCurses](#) está disponível.

## 1.1 O módulo `curses` de Python

O módulo Python é um invólucro razoavelmente simples sobre as funções C disponibilizadas pela `curses`; Se você já estiver familiarizado com programação `curses` em C, é bem fácil transmitir esse conhecimento para o Python. A maior diferença é que a interface Python simplifica as coisas ao juntar diferentes funções C como `addstr()`, `mvaddstr()` e `mvwaddstr()` em um único método `addstr()`. Veremos isso mais adiante.

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`

Antes de qualquer coisa, a `curses` precisa ser inicializada. Para isso, chame a função `initscr()`, que vai determinar o tipo do terminal, enviar para ele quaisquer códigos de inicialização necessários, e criar várias estruturas de dados internas. Se bem-sucedida, a `initscr()` retorna um objeto de janela representando a tela inteira; ele é comumente chamado `stdscr` por causa da variável C correspondente.

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

Geralmente aplicações `curses` desativam o envio automático de teclas para a tela, para que seja possível ler teclas e exibi-las somente 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 usadas pela biblioteca curses. Então, chame a função `endwin()` para restaurar o terminal para seu modo de operação original.

```
curses.endwin()
```

Um problema comum ao depurar 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 levanta 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):
    # Limpa a tela
    stdscr.clear()

    # Levanta ZeroDivisionError quando i == 10.
    for i in range(0, 11):
        v = i-10
        stdscr.addstr(i, 0, '10 dividido por {} é {}'.format(v, 10/v))

        stdscr.refresh()
        stdscr.getkey()

wrapper(main)
```

A função `wrapper()` leva um objeto chamável e faz as inicializações descritas acima, inicializando também as cores se o terminal as prover. A `wrapper()` então roda o chamável. Quando ele retornar, a `wrapper()` vai restaurar o estado original do terminal. O chamável é chamado dentro de um bloco `try...except` que captura exceções, restaura o estado do terminal, e então as levanta novamente. Dessa forma, o seu terminal não vai ser deixado em nenhum estado engraçadinho caso haja qualquer erro, e você poderá ler a mensagem e o traceback da exceção.

### 3 Janelas e Pads

Janelas são a abstração mais básica em curses. Uma objeto janela representa uma área retangular da tela, e provê métodos para exibir texto, apagá-lo, e permitir ao 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 válidas vão variar de `(0,0)` até `(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.

Isso é porque a `curses` foi desenvolvida originalmente no contexto de conexões lentas de até 300 Bauds com o terminal; para esses terminais, minimizar o tempo necessário para redesenhar a tela era importantíssimo. No lugar disso, a `curses` acumula mudanças na tela, e as exibe da forma mais eficiente possível quando você chama `refresh()`. Por exemplo, se o seu programa mostra um texto em uma janela e então limpa a janela, não tem porque enviar o texto original já que ele nunca seria visível.

Na prática, ficar pedindo ao `curses` explicitamente para redesenhar a tela não torna a programação com ele tão complicada assim. A maioria dos programas faz uma enxurrada de atividades, e então pausa esperando uma tecla digitada ou alguma outra ação por parte do usuário. Tudo que você deve fazer é garantir que a tela foi redesenhada antes de pausar esperando pela entrada do usuário, chamando antes a `stdscr.refresh()` ou o método `refresh()` de alguma outra janela apropriada.

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)
# Estes laços preenchem o pad com letras; a addch()
# é explicada na seção a seguir.
for y in range(0, 99):
    for x in range(0, 99):
        pad.addch(y,x, ord('a') + (x*x+y*y) % 26)

# Exibe um pedaço de um pad no meio da tela.
# (0,0) : coordenada do canto superior esquerdo da área do pad a ser
#        exibida.
# (5,5) : coordenada do canto superior esquerdo da área da janela a ser
#        preenchida com o conteúdo do pad.
# (20, 75) : coordenada inferior direita da área da janela a ser
#           : preenchida com o conteúdo do pad.
pad.refresh( 0,0, 5,5, 20,75)
```

A chamada da `refresh()` exibe um pedaço do `pad` no retângulo que vai da coordenada `(5,5)` à coordenada `(20,75)` da tela; o canto superior esquerdo do pedaço exibido corresponde à coordenada `(0,0)` do `pad`. Fora essa diferença, `pads` são exatamente como janelas comuns, e provêm os mesmos métodos.

Se você tem múltiplas janelas e `pads` na tela, existe um modo mais eficiente de atualizar a tela que evita as cintilações irritantes que ocorrem à medida que cada parte da tela vai sendo atualizada. O `refresh()` na verdade faz duas coisas:

- 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 desejado presente na estrutura de dados.

Em vez disso você pode chamar `noutrefresh()` em várias janelas para atualizar a estrutura de dados, e então chamar `doupdate()` para atualizar a tela.

## 4 Exibindo texto

Do ponto de vista de um programador C, a `curses` à vezes pode parecer como um labirinto sinuoso de funções, todas ligeiramente diferentes. Por exemplo, `addstr()` exibe uma string na posição atual do cursor na janela `stdscr`, enquanto `mvaddstr()` primeiro move o cursor para uma dada coordenada `y,x` antes de exibir a string. A `waddstr()` é como a `addstr()`, mas permite especificar uma janela para utilizar em vez de usar a `stdscr` por padrão. A `mvwaddstr()` permite especificar tanto uma janela quanto uma coordenada.

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
<code>str</code> ou <code>ch</code>	Mostra a string <code>str</code> ou caractere <code>ch</code> na posição atual.
<code>str</code> ou <code>ch, attr</code>	Mostra a string <code>str</code> ou caractere <code>ch</code> , usando o atributo <code>attr</code> na posição atual.
<code>y, x, str</code> ou <code>ch</code>	Mover para a posição <code>y,x</code> dentro da janela, e exibe <code>str</code> ou <code>ch</code>
<code>y, x, str</code> ou <code>ch, attr</code>	Mover para a posição <code>y,x</code> dentro da janela, e exibir <code>str</code> ou <code>ch</code> , usando o atributo <code>attr</code>

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

O método `addstr()` recebe uma string ou bytearray Python como valor a ser exibido. Os conteúdos de bytestrings são enviados ao terminal tais como estão. Strings são codificadas em bytes usando o valor do atributo `encoding` da janela; o valor padrão usado é a codificação padrão do sistema como retornado por `locale.getencoding()`.

Os métodos `addch()` pegam um caractere, que pode ser tanto uma string de comprimento 1, um bytearray 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 cursor com o método `move(y, x)`. Porque alguns terminais sempre exibem um cursor 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

Caracteres podem ser exibidos de diferentes formas. Linhas de status em aplicações baseadas em texto são tradicionalmente exibidas em vídeo reverso, ou um visualizador de texto pode precisar destacar certas palavras. Para isso, a `curses` permite que você especifique um atributo para cada célula na tela.

Um atributo é um inteiro em que cada bit representa um atributo diferente. Você sempre pode tentar exibir texto com múltiplos bits de atributo ligados ao mesmo tempo, mas a `curses` não garante que todas as combinações possíveis estarão disponíveis, ou que elas são todas visualmente distintas. Isso depende da habilidade do terminal sendo usado, de forma que é mais seguro se ater aos atributos mais comumente disponíveis, listados aqui.

Atributo	Descrição
<code>A_BLINK</code>	Texto piscante
<code>A_BOLD</code>	Texto em negrito ou brilho extra
<code>A_DIM</code>	Texto em brilho médio
<code>A_REVERSE</code>	Texto em vídeo reverso
<code>A_STANDOUT</code>	O modo mais destacado disponível
<code>A_UNDERLINE</code>	Texto sublinhado

Assim, pra exibir uma linha de status em vídeo reverso na linha no topo da tela, você poderia escrever:

```
stdscr.addstr(0, 0, "Modo atual: Modo de digitação",
              curses.A_REVERSE)
stdscr.refresh()
```

A biblioteca `curses` também permite o uso de cores nos terminais que as provêm. Destes, o terminal mais comum é provavelmente o console Linux, seguido pelos `xterms` com cores.

Para usar cores, você deve chamar a função `start_color()` logo depois de chamar `initscr()`, para inicializar a paleta de cores padrão (a função `curses.wrapper()` faz isso automaticamente.) Uma vez feito isso, a função `has_colors()` retorna `True` se o terminal em uso de fato suportar cores. (Nota: a `curses` usa a grafia Americana ‘color’ ao invés da grafia Britânica/Canadense ‘colour.’ Se você estiver acostumado com a grafia Britânica, você terá que se resignar a escrever com a grafia trocada ao chamar essas funções.)

A biblioteca `curses` mantém uma quantidade finita de pares de cores, contendo uma cor de primeiro plano (ou de texto) e uma de fundo. Você pode obter o valor do atributo correspondente a um par de cores com a função `color_pair()`; esse valor pode ser combinado com OU (OR) bit a bit com outros atributos como `A_REVERSE`, mas, novamente, não é garantido que tais combinações vão funcionar em todos os terminais.

Um exemplo, exibindo uma linha de texto com o par de cores 1:

```
stdscr.addstr("Texto colorido", curses.color_pair(1))
stdscr.refresh()
```

Como dito anteriormente, um par de cores consiste de uma cor de primeiro plano e uma cor de fundo. A função `init_pair(n, f, b)` muda a definição do par de cores *n* para a cor de primeiro plano *f* e cor de fundo *b*. O par 0 é pré-definido como branco sobre preto, e não pode ser mudado.

As cores são numeradas, e a `start_color()` inicializa 8 cores básicas ao ativar o modo colorido. São elas: 0:preto, 1:vermelho, 2:verde, 3:amarelo, 4:azul, 5:magenta, 6:ciano e 7:branco. O módulo `curses` define constantes nomeadas para cada uma dessas cores: `curses.COLOR_BLACK`, `curses.COLOR_RED`, e assim por diante.

Vamos juntar tudo isso. Para redefinir a cor 1 como texto vermelho sobre fundo branco, você chamaria:

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

Quando você redefine um par de cores, qualquer texto que já tenha sido exibido usando esse par de cores vai mudar para a nova definição. Você pode também exibir mais texto nessa nova cor com:

```
stdscr.addstr(0,0, "ALERTA VERMELHO!", curses.color_pair(1))
```

Terminais super sofisticados podem trocar as definições das cores em si para um dado valor RGB. Isso permite que você troque a cor 1, que geralmente é vermelha, para roxo ou azul ou qualquer outra cor que você queira. Infelizmente, o terminal Linux não provê essa funcionalidade, de forma que eu não consigo testá-la para fornecer exemplos. Você pode verificar se o seu terminal tem esse recurso chamando a função `can_change_color()`, que retorna `True` se o recurso existir. Se por ventura o seu terminal for talentoso assim, consulte as páginas man do seu sistema para mais informações.

## 5 Entrada de usuário

A biblioteca `curses` em C oferece somente mecanismos de entrada muito simples. O módulo `curses` do Python adiciona um componente básico de entrada de texto. (Outras bibliotecas como [Urwid](#) têm coleções mais extensas de componentes.)

Há dois métodos para capturar entrada de uma janela:

- O `getch()` atualiza a tela e então espera o usuário apertar alguma tecla, exibindo o valor dela caso `echo()` tenha sido chamada anteriormente. Opcionalmente, você pode especificar a coordenada onde o cursor deve ser posicionado antes de pausar.
- O `getkey()` faz a mesma coisa, mas converte o inteiro para string. Caracteres individuais são retornados como strings de 1 caractere, e teclas especiais como teclas funcionais retornam strings maiores contendo nomes de teclas como `KEY_UP` ou `^G`.

É possível não esperar pelo usuário usando o método `nodelay()` das janelas. Após `nodelay(True)`, os métodos `getch()` e `getkey()` da janela viram não-bloqueantes. Para sinalizar que não há entrada disponível, o `getch()` retorna `curses.ERR` (com valor de -1) e o `getkey()` levanta uma exceção. Também há uma função `halfdelay()`,

que pode ser usada para (efetivamente) definir um temporizador em cada `getch()`; se nenhuma entrada estiver disponível após um intervalo de espera especificado (medido em décimos de segundo), a `curses` levanta uma exceção.

O método `getch()` retorna um inteiro. Se ele estiver entre 0 e 255, ele representa o código ASCII da tecla digitada. Valores maiores que 255 são teclas especiais como Page Up, Home, ou as teclas de seta. Você pode comparar o valor retornado com constantes como `curses.KEY_PPAGE`, `curses.KEY_HOME` ou `curses.KEY_LEFT`. O laço principal do seu programa pode ser algo como:

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

O módulo `curses.ascii` fornece funções de teste de pertencimento que levam como argumento tanto inteiros quanto strings de 1 caractere; elas podem ser úteis para escrever testes mais legíveis para tais laços. Ele também fornece funções de conversão que levam como argumento tanto inteiros quanto strings de 1 caractere, e retornam objetos do mesmo tipo. Por exemplo, `curses.ascii.ctrl()` retorna o caractere de controle correspondente ao argumento.

Também há um método para pegar uma string inteira, `getstr()`. Ele não costuma ser usado porque a sua funcionalidade é um pouco limitada: as únicas teclas de edição disponíveis são o Backspace e o Enter, que termina a string. Ele pode opcionalmente ser limitado a um número fixo de caracteres.

```
curses.echo() # Abilita visualização dos caracteres

# Lê uma string de 15 caracteres, com o cursor na linha do topo
s = stdscr.getstr(0, 0, 15)
```

O módulo `curses.textpad` fornece uma caixa de texto que provê um conjunto de atalhos de teclado estilo Emacs. Vários métodos da classe `Textbox` provêm edição com validação de entrada e recuperação dos resultados da edição com ou sem espaços em branco no final.

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

def main(stdscr):
    stdscr.addstr(
        0, 0,
        "Digite a mensagem instantânea: (aperte Ctrl-G para enviar)")

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

    box = Textbox(editwin)

    # Deixa o usuário editar até apertar Ctrl-G.
    box.edit()

    # Recupera o conteúdo resultante.
    message = box.gather()
```

Consulte a documentação de biblioteca do `curses.textpad` para mais detalhes.

## 6 Para mais informações

Este COMOFAZER não cobre alguns tópicos avançados, como ler o conteúdo da tela ou capturar eventos de mouse de uma instância xterm, mas a página de biblioteca Python para o módulo `curses` está agora razoavelmente completa. Ela deve ser o seu próximo destino.

Se estiver em dúvida quanto aos detalhes do comportamento das funções `curses`, consulte as páginas de manual para a sua implementação `curses`, seja ela o `ncurses` ou de algum fornecedor Unix proprietário. As páginas de manual vão documentar quaisquer peculiaridades, e prover listas completas de todas as funções, atributos e caracteres ACS\_\* disponíveis para você.

Devido à API da `curses` ser tão grande, algumas funções não são providas pela interface Python. Muitas vezes isso não é por dificuldade de implementação, e sim porque ninguém precisou delas ainda. Além disso, o Python ainda não suporta a biblioteca de menus associada à `ncurses`. Patches que adicionem suporte a essas funcionalidades seriam bem-vindos; veja o [Guia do Desenvolvedor do Python](#) para aprender melhor sobre como submeter patches ao Python.

- [Writing Programs with NCURSES](#): um tutorial comprido para programadores C.
- [Página man do ncurses](#)
- [FAQ do ncurses](#)
- “Use `curses`... don’t swear”: vídeo de uma palestra na PyCon 2013 sobre controle de terminais usando `curses` ou `Urwid`.
- “[Console Applications with Urwid](#)”: vídeo de uma palestra na PyCon CA 2012 demonstrando algumas aplicações escritas com `Urwid`.