
Programación de curses con Python

Versión 3.13.3

Guido van Rossum and the Python development team

abril 23, 2025

Python Software Foundation
Email: docs@python.org

Índice general

1	¿Qué es curses?	1
1.1	El módulo curses de Python	2
2	Iniciar y finalizar una aplicación de curses	2
3	Ventanas y pads	3
4	Mostrando el texto	4
4.1	Atributos y color	5
5	Input del usuario	6
6	Para más información	8

Autor

A.M. Kuchling, Eric S. Raymond

Versión

2.04

Resumen

Este documento describe cómo usar el módulo de extensión `curses` para controlar las pantallas en modo texto.

1 ¿Qué es curses?

La biblioteca `curses` proporciona una instalación independiente del terminal para manejo de teclado e impresión de pantalla en terminales basados en texto; tales terminales incluyen VT100, la consola de Linux y el terminal simulado proporcionado por varios programas. Los terminales de pantalla admiten varios códigos de control para realizar operaciones comunes, como mover el cursor, desplazar la pantalla y borrar áreas. Diferentes terminales utilizan códigos muy diferentes y, a menudo, tienen sus propias peculiaridades menores.

En un mundo de pantallas gráficas, uno podría preguntarse « ¿por qué molestarse »? Es cierto que los terminales de pantalla de celdas de caracteres son una tecnología obsoleta, pero hay nichos en los que poder hacer cosas elegantes con ellos sigue siendo valioso. Un nicho está en pequeños *Unix* incrustados o pequeñas marcas que no ejecutan un

servidor X. Por otro lado herramientas como los instaladores del sistema operativo y los configuradores de kernel que pueden tener que ejecutarse antes de que haya soporte gráfico disponible.

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](#).

La biblioteca `curses` se escribió originalmente para BSD Unix; Las versiones posteriores de Unix de *System V* de AT&T agregaron muchas mejoras y nuevas funciones. BSD `curses` ya no se mantiene, ya que ha sido reemplazado por `ncurses`, que es una implementación de código abierto de la interfaz AT&T. Si está utilizando un Unix de código abierto como Linux o FreeBSD, su sistema casi seguro usa `ncurses`. Como la mayoría de las versiones comerciales actuales de Unix se basan en el código del Sistema V, todas las funciones descritas aquí probablemente estarán disponibles. Sin embargo, las versiones anteriores de `curses` llevadas por algunos Unix propietarios no pueden admitir todo.

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

1.1 El 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 HOWTO es una introducción a la escritura de programas en modo texto con `curses` y Python. No intenta ser una guía completa de la API de `curses`; para eso, vea la sección de la guía de la biblioteca de Python sobre `ncurses`, y las páginas del manual de C para `ncurses`. Sin embargo, le dará las ideas básicas.

2 Iniciar y finalizar una aplicación de `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()
```

Por lo general, las aplicaciones de `curses` desactivan el eco automático de las teclas en la pantalla, para poder leer las teclas y solo mostrarlas bajo ciertas circunstancias. Esto requiere llamar a la función `noecho()` function.

```
curses.noecho()
```

Las aplicaciones también tendrán que reaccionar a las teclas al instante, sin necesidad de presionar la tecla Intro; Esto se llama modo `cbreak`, en oposición al modo de entrada almacenado en memoria intermedia habitual.

```
curses.cbreak()
```

Los terminales generalmente retornan teclas especiales, como las teclas del cursor o las teclas de navegación, como `Re Pág` e `Inicio`, como una secuencia de escape multibyte. Si bien puede escribir su aplicación para esperar tales secuencias y procesarlas en consecuencia, `curses` pueden hacerlo por usted, retornando un valor especial como `curses.KEY_LEFT`. Para obtener que `curses` haga el trabajo, deberá habilitar el modo de teclado.

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

Terminar una aplicación de `curses` es mucho más fácil que iniciar una. Tendrás que llamar a

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

para revertir la configuración de terminal amigable de `curses`. Llame luego a la función `endwin()` para restaurar el terminal a su modo operativo original.

```
curses.endwin()
```

Un problema común al depurar una aplicación `curses` es hacer que su terminal se estropee cuando la aplicación muere sin restaurar el terminal a su estado anterior. En Python, esto ocurre comúnmente cuando su código tiene errores y genera una excepción no detectada. Las teclas ya no se repiten en la pantalla cuando las escribe, por ejemplo, lo que dificulta el uso del shell.

En Python puede evitar estas complicaciones y facilitar la depuración importando la función `curses.wrapper()` y usándola así:

```
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 Ventanas y pads

Las ventanas son la abstracción básica en `curses`. Un objeto de ventana representa un área rectangular de la pantalla y admite métodos para mostrar texto, borrarlo, permitir al usuario ingresar cadenas, etc.

El objeto `stdscr` retornado por la función `initscr()` es un objeto de ventana que cubre toda la pantalla. Es posible que muchos programas solo necesiten esta única ventana, pero es posible que desee dividir la pantalla en ventanas más pequeñas para volver a dibujarlas o borrarlas por separado. La función `newwin()` crea una nueva ventana de un tamaño dado, retornando el nuevo objeto de ventana.

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

Tenga en cuenta que el sistema de coordenadas utilizado en `curses` es inusual. Las coordenadas siempre se pasan en el orden `y,x`, y la esquina superior izquierda de una ventana es la coordenada `(0,0)`. Esto rompe la convención normal para manejar coordenadas donde la coordenada `x` es lo primero. Esta es una desafortunada diferencia de la mayoría de las otras aplicaciones informáticas, pero ha sido parte de `curses` desde que se escribió por primera vez, y ahora es demasiado tarde para cambiar las cosas.

Su aplicación puede determinar el tamaño de la pantalla utilizando las variables `curses.LINES` y `curses.COLS` para obtener los tamaños `y` y `x`. Las coordenadas legales se extenderán de `(0,0)` a `(curses.LINES - 1, curses.COLS - 1)`.

Cuando llama a un método para mostrar o borrar texto, el efecto no aparece inmediatamente en la pantalla. En su lugar, debe llamar al método `refresh()` de los objetos de ventana para actualizar la pantalla.

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.

Un pad es un caso especial de una ventana; puede ser más grande que la pantalla de visualización real, y solo se muestra una parte del pad a la vez. La creación de un pad requiere la altura y el ancho del pad, mientras que la actualización de un pad requiere dar las coordenadas del área en pantalla donde se mostrará una subsección del pad.

```
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) Llama al método `noutrefresh()` de cada ventana para actualizar una estructura de datos subyacente que representa el estado deseado de la pantalla.
- 2) Llama a la función `doupdate()` para cambiar la pantalla física para que coincida con el estado deseado registrado en la estructura de datos.

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

4 Mostrando el 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.

Afortunadamente, la interfaz de Python oculta todos estos detalles. `stdscr` es un objeto de ventana como cualquier otro, y métodos como `addstr()` aceptan múltiples formas de argumento. Por lo general, hay cuatro formas diferentes.

Formas	Descripción
<code>str</code> o <code>ch</code>	Mostrar la cadena <code>str</code> o el carácter <code>ch</code> en la posición actual
<code>str</code> o <code>ch</code> , <code>attr</code>	Muestra la cadena <code>str</code> o el carácter <code>ch</code> , utilizando el atributo <code>attr</code> en la posición actual
<code>y</code> , <code>x</code> , <code>str</code> o <code>ch</code>	Moverse a la posición <code>y,x</code> dentro de la ventana, y mostrar <code>str</code> o <code>ch</code>
<code>y</code> , <code>x</code> , <code>str</code> o <code>ch</code> , <code>attr</code>	Muévase a la posición <code>y</code> , <code>x</code> dentro de la ventana y muestre <code>str</code> o <code>ch</code> , usando el atributo <code>attr</code>

Los atributos permiten mostrar texto en formas resaltadas como negrita, subrayado, código inverso o en color. Se explicarán con más detalle en la siguiente subsección.

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()`.

Los métodos `addch()` toman un carácter, que puede ser una cadena de longitud 1, una cadena de bytes de longitud 1 o un entero.

Se proporcionan constantes para los caracteres de extensión; estas constantes son enteros mayores que 255. Por ejemplo `ACS_PLMINUS` es un símbolo +/-, y `ACS_ULCORNER` es la esquina superior izquierda de un cuadro (útil para dibujar bordes). También puede usar el carácter Unicode apropiado.

Windows recuerda dónde se dejó el cursor después de la última operación, por lo que si deja de lado las coordenadas `y`, `x`, la cadena o el carácter se mostrarán donde se haya quedado la última operación. También puede mover el cursor con el método `move(y, x)`. Debido a que algunos terminales siempre muestran un cursor parpadeante, es posible que desee asegurarse de que el cursor esté ubicado en algún lugar donde no distraiga; Puede ser confuso tener el cursor parpadeando en alguna ubicación aparentemente aleatoria.

Si su aplicación no necesita un cursor parpadeante, puede llamar a `curs_set(False)` para hacerla invisible. Para compatibilidad con versiones anteriores de `curses`, hay una función `jeaveok(bool)` que es sinónimo de `curs_set()`. Cuando `bool` es verdadero, la biblioteca `curses` intentará suprimir el cursor parpadeante, y no tendrá que preocuparse por dejarlo en ubicaciones extrañas.

4.1 Atributos y color

Los caracteres se pueden mostrar de diferentes maneras. Las líneas de estado en una aplicación basada en texto se muestran comúnmente en video inverso, o un visor de texto puede necesitar resaltar ciertas palabras. `curses` admite esto al permitirle especificar un atributo para cada celda en la pantalla.

Un atributo es un entero, cada bit representa un atributo diferente. Puede intentar mostrar texto con múltiples conjuntos de bits de atributos, pero `curses` no garantizan que todas las combinaciones posibles estén disponibles, o que todas sean visualmente distintas. Eso depende de la capacidad del terminal que se utilice, por lo que es más seguro apegarse a los atributos más comúnmente disponibles, enumerados aquí.

Atributo	Descripción
<code>A_BLINK</code>	Texto parpadeante
<code>A_BOLD</code>	Texto extra brillante o en negrita
<code>A_DIM</code>	Texto medio brillante
<code>A_REVERSE</code>	Texto de video inverso
<code>A_STANDOUT</code>	El mejor modo de resaltado disponible
<code>A_UNDERLINE</code>	Texto subrayado

Entonces, para mostrar una línea de estado de video inverso en la línea superior de la pantalla, puede codificar:

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

La biblioteca `curses` también admite color en los terminales que lo proporcionen. El terminal más común es probablemente la consola de Linux, seguido de *xterms* en color.

Para usar el color, debe llamar a la función `start_color()` poco después de llamar `initscr()`, para inicializar el conjunto de colores predeterminado (la función `curses.wrapper()` hace esto automáticamente). Una vez hecho esto, la función `has_colors()` retorna `TRUE` si el terminal en uso realmente puede mostrar color. (Nota: `curses` usa el 'color' de la ortografía estadounidense, en lugar del 'color' de la ortografía canadiense/británica. Si está acostumbrado a la ortografía británica, tendrá que resignarse a escribir mal por el bien de estas funciones.)

La biblioteca `curses` mantiene un número finito de pares de colores, que contienen un color de primer plano (o texto) y un color de fondo. Puede obtener el valor del atributo correspondiente a un par de colores con la función `color_pair()`; esto puede ser operado bit a bit con otros atributos como `A_REVERSE`, pero nuevamente, no se garantiza que tales combinaciones funcionen en todos los terminales.

Un ejemplo, que muestra una línea de texto usando el par de colores 1

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

Como dije antes, un par de colores consiste en un color de primer plano y de fondo. La función `init_pair(n, f, b)` cambia la definición del par de colores `n`, a color de primer plano `f` y color de fondo `b`. El par de colores 0 está cableado a blanco sobre negro, y no se puede cambiar.

Los colores están numerados y `start_color()` inicializa 8 colores básicos cuando activa el modo de color. Son: 0:negro, 1:rojo, 2:verde, 3:amarillo, 4:azul, 5:magenta, 6:cian y 7:blanco. El módulo `curses` define constantes con nombre para cada uno de estos colores: `curses.COLOR_BLACK`, `curses.COLOR_RED`, y así sucesivamente.

Pongamos todo esto juntos. Para cambiar el color 1 al texto rojo sobre un fondo blanco, debe llamar a:

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

Cuando cambia un par de colores, cualquier texto que ya se muestre usando ese par de colores cambiará a los nuevos colores. También puede mostrar texto nuevo en este color con:

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

Los terminales muy elegantes pueden cambiar las definiciones de los colores reales a un valor RGB dado. Esto le permite cambiar el color 1, que generalmente es rojo, a púrpura o azul o cualquier otro color que desee. Desafortunadamente, la consola de Linux no admite esto, por lo que no puedo probarlo y no puedo proporcionar ningún ejemplo. Puede verificar si su terminal puede hacer esto llamando a `can_change_color()`, que retorna `Verdadero` si la capacidad está ahí. Si tiene la suerte de tener un terminal tan talentoso, consulte las páginas de manual de su sistema para obtener más información.

5 Input del usuario

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.)

Hay dos métodos para obtener información desde una ventana:

- `getch()` actualiza la pantalla y luego espera a que el usuario presione una tecla, mostrando la tecla si `echo()` ha sido llamado anteriormente. Opcionalmente, puede especificar una coordenada a la que se debe mover el cursor antes de pausar.
- `getkey()` hace lo mismo pero convierte el entero en una cadena. Los caracteres individuales se retornan como cadenas de 1 carácter, y las teclas especiales como las teclas de función retornan cadenas más largas que contienen un nombre de tecla como `KEY_UP` o `^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
```

El módulo `curses.ascii` proporciona funciones de membresía de clase ASCII que toman argumentos de cadena de entero o de 1 carácter; Estos pueden ser útiles para escribir pruebas más legibles para dichos bucles. También proporciona funciones de conversión que toman argumentos enteros o de cadena de 1 carácter y retornen el mismo tipo. Por ejemplo, `curses.ascii.ctrl()` retorna el carácter de control correspondiente a su argumento.

También hay un método para recuperar una cadena completa, `getstr()`. No se usa con mucha frecuencia, porque su funcionalidad es bastante limitada; Las únicas teclas de edición disponibles son la tecla de retroceso y la tecla Intro, que termina la cadena. Opcionalmente, puede limitarse a un número fijo de caracteres.

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

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

El módulo `curses.textpad` proporciona un cuadro de texto que admite un conjunto de teclas tipo Emacs. Varios métodos de la clase `Textbox` admiten la edición con validación de entrada y recopilan los resultados de edición con o sin espacios finales. Aquí hay un ejemplo:

```
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()
```

Consulte la documentación de la biblioteca sobre `curses.textpad` para más detalles.

6 Para más información

Este CÓMO no cubre algunos temas avanzados, como leer el contenido de la pantalla o capturar eventos del mouse desde una instancia de xterm, pero la página de la biblioteca de Python para el módulo `curses` ahora está razonablemente completa. Deberías buscarlo posteriormente.

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.

Debido a que la API de curses es tan grande, algunas funciones no son compatibles con la interfaz de Python. A menudo, esto no se debe a que sean difíciles de implementar, sino a que nadie los ha necesitado todavía. Además, Python aún no admite la biblioteca de menús asociada con ncurses. Los parches que agregan soporte para estos serían bienvenidos; consulte [la Guía del desarrollador de Python](#) para obtener más información sobre cómo enviar parches a Python.

- [Escribir programas con NCURSES](#): un tutorial extenso para programadores en C.
- [La página web con el manual de ncurses](#)
- [Preguntas y respuestas frecuentes de ncurses](#)
- «Use curses... don't swear»: video de una charla de PyCon 2013 sobre el control de terminales usando curses o Urwid.
- «Console Applications with Urwid»: video de una charla en PyCon CA 2012 que muestra algunas aplicaciones escritas usando Urwid.