Programación de Curses con Python

Versión 3.10.10

Guido van Rossum
and the Python development team

marzo 27, 2023

Python Software Foundation
Email: docs@python.org

Índice general

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.

La biblioteca curses proporciona una funcionalidad bastante básica, proporcionando al programador una abstracción de una pantalla que contiene múltiples ventanas de texto no superpuestas. El contenido de una ventana se puede cambiar de varias maneras (agregando texto, borrándolo, cambiando su apariencia) y la biblioteca curses descubrirá que códigos de control deben enviarse al terminal para producir la salida correcta. curses no proporciona muchos conceptos de interfaz de usuario como botones, casillas de verificación o cuadros de diálogo; Si necesita tales funciones, considere una biblioteca de interfaz de usuario como *Urwid <https://pypi.org/project/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

El módulo Python es un envoltorio bastante simple sobre las funciones C proporcionadas por curses; si ya está familiarizado con la programación de curses en C, es muy fácil transferir ese conocimiento a Python. La mayor diferencia es que la interfaz de Python simplifica las cosas al combinar diferentes funciones de C como `addtr()` , `mvaddstr()` , y `mvwaddstr()` en un solo método `addstr()` . Verá esto cubierto con más detalle más adelante.

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

Antes de hacer cualquier cosa, curses deben ser inicializadas. Esto se realiza llamando a la función `initscr()` , que determinará el tipo de terminal, enviará los códigos de configuración necesarios al terminal y creará varias estructuras de datos internas. Si tiene éxito, `initscr()` retorna un objeto de ventana que representa la pantalla completa; Esto generalmente se llama `stdscr` después del nombre de la variable C correspondiente.

```
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)
```

La función `wrapper()` toma un objeto invocable y realiza las inicializaciones descritas anteriormente, también inicializa los colores si hay soporte de color. `wrapper()` luego ejecuta su invocable proporcionado. Una vez que vuelve a llamarse, `wrapper()` restaurará el estado original del terminal. El invocable se llama dentro de `try...except` que captura excepciones, restaura el estado del terminal y luego vuelve a generar la excepción. Por lo tanto, su terminal no se quedará en un estado extraño de excepción y podrá leer el mensaje de la excepción y el rastreo.

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.

Esto se debe a que curses se escribieron originalmente teniendo en cuenta las conexiones de terminales lentas de 300 baudios; con estos terminales, era muy importante minimizar el tiempo requerido para volver a dibujar la pantalla. En cambio, curses acumula cambios en la pantalla y los muestra de la manera más eficiente cuando llama `refresh()`. Por ejemplo, si su programa muestra algo de texto en una ventana y luego borra la ventana, no hay necesidad de enviar el texto original porque nunca son visibles.

En la práctica, decirle explícitamente a curses que vuelva a dibujar una ventana no complica mucho la programación con curses. La mayoría de los programas entran en una gran cantidad de actividad y luego se detienen a la espera de que se presione una tecla u otra acción por parte del usuario. Todo lo que tiene que hacer es asegurarse de que la pantalla se haya re-dibujado antes de hacer una pausa para esperar la entrada del usuario, llamando primero a `stdscr.refresh()` o al método `refresh()` de alguna otra ventana relevante.

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)
```

La llamada `refresh()` muestra una sección del pad en el rectángulo que se extiende desde la coordenada (5,5) hasta la coordenada (20,75) en la pantalla; la esquina superior izquierda de la sección mostrada es la coordenada (0,0) en el pad. Más allá de esa diferencia, los pads son exactamente como las ventanas normales y admiten los mismos métodos.

Si tiene varias ventanas y almohadillas en la pantalla, hay una manera más eficiente de actualizar la pantalla y evitar el molesto parpadeo de la pantalla a medida que se actualiza cada parte de la pantalla. `refresh()` en realidad hace dos cosas:

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.

En su lugar, puede llamar a `noutrefresh()` en varias ventanas para actualizar la estructura de datos, y luego llamar a `doupdate()` para actualizar la pantalla.

4 Mostrando el texto

Desde el punto de vista de un programador en C, curses a veces puede parecer un laberinto retorcido de funciones, todas sutilmente diferentes. Por ejemplo, `addtr()` muestra una cadena en la ubicación actual del cursor en la ventana `stdscr`, mientras que `mvaddstr()` se mueve a una determinada coordenada y, x primero antes de mostrar la cadena . `waddstr()` es como `addtr()`, pero permite especificar una ventana para usar en lugar de usar `stdscr` por defecto. `mvwaddstr()` permite especificar tanto una ventana como una coordenada.

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
str o *ch*	Mostrar la cadena *str* o el carácter *ch* en la posición actual
str o *ch*, *attr*	Muestra la cadena *str* o el carácter *ch*, utilizando el atributo *attr* en la posición actual
y, *x*, *str* o *ch*	Moverse a la posición *y,x* dentro de la ventana, y mostrar *st* o *ch*
y, *x*, *str* o *ch*, *attr*	Muévase a la posición *y, x* dentro de la ventana y muestre *str* o *ch*, usando el atributo *attr*

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.

El método `addstr()` toma una cadena de Python o una cadena de bytes como el valor que se mostrará. El contenido de las cadenas de bytes se envía al terminal tal como está. Las cadenas se codifican en bytes utilizando el valor del atributo de la ventana `encoding`; Esto se establece de manera predeterminada en la codificación predeterminada del sistema que retorna `locale.getpreferredencoding()`.

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
A_BLINK	Texto parpadeante
A_BOLD	Texto extra brillante o en negrita
A_DIM	Texto medio brillante
A_REVERSE	Texto de video inverso
A_STANDOUT	El mejor modo de resaltado disponible
A_UNDERLINE	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

La biblioteca C curses ofrece solo mecanismos de entrada muy simples. El módulo Python `curses` agrega un widget básico de entrada de texto. (Otras bibliotecas como Urwid tienen colecciones más extensas de 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`.

Es posible no esperar al usuario utilizando el método de ventana `nodelay()`. Después de `nodelay(True)`, `getch()` y `getkey()` para que la ventana no se bloquee. Para indicar que no hay ninguna entrada lista, `getch()` retorna `curses.ERR` (un valor de -1) y `getkey()` genera una excepción. También hay una función `halfdelay()`, que se puede usar para (en efecto) establecer un temporizador en cada `getch()`; Si no hay entrada disponible dentro de un retraso especificado (medido en décimas de segundo), curses generan una excepción.

El método `getch()` retorna un entero; si está entre 0 y 255, representa el código ASCII de la tecla presionada. Los valores superiores a 255 son teclas especiales como Re Pág, Inicio o las teclas del cursor. Puede comparar el valor retornado a constantes como `curses.KEY_PPAGE`, `curses.KEY_HOME`, o `curses.KEY_LEFT`. El bucle principal de su programa puede verse así:

```
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()
```

(continué en la próxima página)

(proviene de la página anterior)

```
    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.

Si tiene dudas sobre el comportamiento detallado de las funciones de curses, consulte las páginas del manual para su implementación de curses, ya sea ncurses o un proveedor exclusivo de Unix. Las páginas del manual documentarán cualquier peculiaridad y proporcionarán listas completas de todas las funciones, atributos y caracteres `ACS_*` disponibles para usted.

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.

- Writing Programs with NCURSES: a lengthy tutorial for C programmers.
- La página web con el manual de ncurses
- The ncurses FAQ
- «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 of a PyCon CA 2012 talk demonstrating some applications written using Urwid.