

---

# 파이썬 curses 프로그래밍

릴리스 3.11.13

Guido van Rossum and the Python development team

7월 07, 2025

## Contents

1	curses란 무엇입니까?	1
1.1	파이썬 curses 모듈 . . . . .	2
2	curses 응용 프로그램 시작과 종료	2
3	창과 패드	3
4	텍스트 표시	4
4.1	속성과 색상 . . . . .	5
5	사용자 입력	6
6	추가 정보	7

---

저자  
A.M. Kuchling, Eric S. Raymond

버전  
2.04

### 요약

이 문서는 curses 확장 모듈을 사용하여 텍스트 모드 디스플레이를 제어하는 방법을 설명합니다.

## 1 curses란 무엇입니까?

curses 라이브러리는 텍스트 기반 터미널을 위한 터미널 독립적인 스크린 페인팅과 키보드 처리 기능을 제공합니다; 이러한 터미널에는 VT100, 리눅스 콘솔 및 다양한 프로그램에서 제공하는 시뮬레이트 된 터미널이 포함됩니다. 디스플레이 터미널은 커서 이동, 화면 스크롤 및 영역 지우기와 같은 일반적인 작업을 수행하기 위해 다양한 제어 코드를 지원합니다. 터미널마다 서로 다른 코드를 사용하며 종종 자신만의 사소한 문제가 있습니다.

그래픽 디스플레이의 세계에서, 이렇게 물을 수 있습니다. “왜 신경 써야 하나요”? 문자 셀 디스플레이 터미널은 한물간 기술이지만, 사실 여전히 가치 있는 멋진 작업을 수행할 수 있는 틈새가 존재합니다. 한가지 틈새는 X 서버를 실행하지 않는 작은 크기 혹은 내장 유닉스입니다. 다른 것으로는 그래픽 지원을 사용할 수 있기 전에 실행해야 할 OS 설치 프로그램과 커널 구성기와 같은 도구가 있습니다.

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](#).

`curses` 라이브러리는 원래 BSD 유닉스용으로 작성되었습니다; 그 이후 AT&T의 유닉스 시스템 V 버전에는 많은 개선 사항과 새로운 기능이 추가되었습니다. BSD `curses`는 더는 유지되지 않고, AT&T 인터페이스의 오픈 소스 구현인 `ncurses`로 대체되었습니다. 리눅스나 FreeBSD 와 같은 오픈 소스 유닉스를 사용하고 있다면, 시스템은 거의 확실히 `ncurses`를 사용합니다. 최신 상용 유닉스 버전은 대부분 시스템 V 코드를 기반으로 하므로, 여기에 설명된 모든 기능을 아마도 사용할 수 있을 것입니다. 그러나 일부 독점적 유닉스가 제공하는 이전 버전의 `curses`는 모든 것을 지원하지 않을 수 있습니다.

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

## 1.1 파이썬 `curses` 모듈

파이썬 모듈은 `curses`가 제공하는 C 함수에 대한 상당히 간단한 래퍼입니다; C의 `curses` 프로그래밍에 이미 익숙하다면 이 지식을 파이썬으로 쉽게 옮길 수 있습니다. 가장 큰 차이점은 파이썬 인터페이스가 `addstr()`, `mvaddstr()` 및 `mvwaddstr()` 와 같은 다른 C 함수를 단일 `addstr()` 메서드로 병합하여 작업을 단순화한다는 것입니다. 나중에 자세히 다루겠습니다.

이 HOWTO는 `curses`와 파이썬으로 텍스트 모드 프로그램을 작성하는 방법을 소개합니다. `curses` API에 대한 완전한 안내서가 되려는 것은 아닙니다; 그것을 위해서는 `ncurses`에 대한 파이썬 라이브러리 안내서 섹션과 `ncurses`에 대한 C 매뉴얼 페이지를 참조하십시오. 그러나 기본 아이디어는 제공합니다.

## 2 `curses` 응용 프로그램 시작과 종료

무언가를 하기 전에 `curses`를 초기화해야 합니다. 이것은 `initscr()` 함수를 호출함으로써 수행되는데, 터미널 유형을 결정하고, 필요한 설정 코드를 터미널에 보내고, 다양한 내부 데이터 구조를 만듭니다. 성공하면 `initscr()`은 전체 화면을 나타내는 창 객체를 반환합니다; 이를 일반적으로 해당 C 변수 이름을 따라 `stdscr`이라고 합니다.

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

일반적으로 `curses` 응용 프로그램은 키를 읽고 특정 상황에서만 표시할 수 있도록 화면으로 키를 자동 에코하는 기능을 끕니다. 이를 위해서는 `noecho()` 함수를 호출해야 합니다.

```
curses.noecho()
```

응용 프로그램은 또한 일반적으로 Enter 키를 누르지 않아도 즉시 키에 반응해야 합니다; 이것을 일반적인 버퍼 입력 모드와 대비하여 `cbreak` 모드라고 합니다.

```
curses.cbreak()
```

터미널은 일반적으로 커서 키나 Page Up, Home 등의 탐색 키와 같은 특수 키를 멀티 바이트 이스케이프 시퀀스로 반환합니다. 이러한 시퀀스를 예상하고 적절하게 처리하도록 응용 프로그램을 작성할 수는 있지만, `curses`가 이를 수행하여 `curses.KEY_LEFT`와 같은 특수 값을 반환할 수 있습니다. `curses`가 이런 일을 하도록 하려면, 키패드 모드를 활성화해야 합니다.

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

`curses` 응용 프로그램을 종료하는 것은 시작하기보다 훨씬 쉽습니다. 다음과 같이 호출합니다:

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

curses 친화적인 터미널 설정을 되돌립니다. 그런 다음 `endwin()` 함수를 호출하여 터미널을 원래 작동 모드로 복원하십시오.

```
curses.endwin()
```

curses 응용 프로그램을 디버깅할 때 일반적인 문제점은 터미널을 이전 상태로 복원하지 않고 응용 프로그램이 죽을 때 터미널이 엉망이 되는 것입니다. 파이썬에서 코드에 버그가 있고 잡히지 않는 예외를 일으킬 때 흔히 일어납니다. 예를 들어, 키를 입력해도 키가 더는 화면에 표시되지 않아 셀 사용이 어려워집니다.

파이썬에서는 `curses.wrapper()` 함수를 임포트하고 다음과 같이 사용하여 이러한 복잡성을 피하고 디버깅을 훨씬 쉽게 할 수 있습니다:

```
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)
```

`wrapper()` 함수는 콜러블 객체를 취하고 위에서 설명한 초기화를 수행합니다, 색상 지원이 있으면 색 상도 초기화합니다. 그런 다음 `wrapper()` 는 제공된 콜러블을 실행합니다. 일단 콜러블이 반환되면, `wrapper()` 는 터미널의 원래 상태를 복원합니다. 콜러블은 예외를 잡는 `try...except` 내에서 호출되고, 터미널의 상태를 복원한 다음 예외를 다시 발생시킵니다. 따라서 터미널은 예외 상황에서 망가진 상태로 남지 않고 예외 메시지와 트레이스백을 읽을 수 있습니다.

### 3 창과 패드

창은 curses의 기본 추상화입니다. 창 객체는 화면의 사각형 영역을 나타내며, 텍스트를 표시하고, 지우고, 사용자가 문자열을 입력할 수 있도록 하는 등의 메서드를 지원합니다.

`initscr()` 함수가 반환한 `stdscr` 객체는 전체 화면을 덮는 창 객체입니다. 많은 프로그램에서 이 창 하나만 필요할 수도 있지만, 따로 다시 그리거나 지우기 위해 화면을 작은 창으로 나누고 싶을 수 있습니다. `newwin()` 함수는 지정된 크기의 새 창을 만들어 새 창 객체를 반환합니다.

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

curses에 사용된 좌표계는 일반적이지 않음에 주의하십시오. 좌표는 항상  $y,x$  순서로 전달되며, 창의 왼쪽 상단 모서리는 좌표 (0,0) 입니다. 이것은  $x$  좌표가 먼저 오는, 좌표를 다루는 일반적인 규칙을 위반합니다. 이것은 대부분 다른 컴퓨터 응용 프로그램과의 불행한 차이점이지만, 처음 작성된 이후로 curses 일부였으며, 지금 되돌리기에 너무 늦었습니다.

응용 프로그램은  $y$ 와  $x$  크기를 얻기 위해 `curses.LINES`와 `curses.COLS` 변수를 사용하여 화면 크기를 결정할 수 있습니다. 그러면 유효한 좌표는 (0,0) 에서 (`curses.LINES - 1`, `curses.COLS - 1`) 에 이릅니다.

텍스트를 표시하거나 지우기 위해 메서드를 호출할 때, 효과는 즉시 디스플레이에 나타나지 않습니다. 대신 화면을 갱신하기 위해 창 객체의 `refresh()` 메서드를 호출해야 합니다.

curses는 원래 300-baud 터미널 연결을 염두에 두고 작성되었기 때문입니다; 이러한 터미널에서는 화면을 다시 그리는 데 필요한 시간을 최소화하는 것이 매우 중요했습니다. 대신 curses는 화면의 변경 사항을

누적하고 `refresh()` 를 호출할 때 가장 효율적인 방식으로 표시합니다. 예를 들어, 프로그램이 창에 일부 텍스트를 표시한 다음 창을 지우면, 원래 텍스트가 표시되지 않기 때문에 보낼 필요가 없습니다.

실제로는, 창을 다시 그리도록 명시적으로 `curses`에 지시한다고 해서 `curses` 프로그래밍이 실제로 복잡해지는 것은 아닙니다. 대부분 프로그램은 복잡한 행동을 한 다음 키 입력이나 사용자의 다른 동작을 기다리기 위해 일시 중지합니다. 여러분이 해야 할 것은 사용자 입력을 기다리기 위해 일시 중지하기 전에 `stdscr.refresh()` 나 다른 적절한 창의 `refresh()` 메서드를 먼저 호출하여 화면을 다시 그리도록 하는 것입니다.

패드(pad)는 창의 특별한 경우입니다; 실제 디스플레이 화면보다 클 수 있으며, 한 번에 패드의 일부만 표시될 수 있습니다. 패드를 만들려면 패드의 높이와 너비가 필요하지만, 패드를 새로 고치려면 패드의 서브 섹션이 표시될 화면 영역의 좌표를 지정해야 합니다.

```
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)
```

`refresh()` 호출은 화면의 좌표 (5,5)에서 좌표 (20,75) 까지 확장된 사각형에 패드 섹션을 표시합니다; 표시된 섹션의 왼쪽 위 모서리는 패드의 좌표 (0,0) 입니다. 이 차이를 제외하고, 패드는 일반 창과 똑같으며 같은 메서드를 지원합니다.

화면에 여러 개의 창과 패드가 있으면, 화면을 갱신하고 화면의 각 부분이 갱신될 때 성가신 화면 깜박임을 방지하는 더 효율적인 방법이 있습니다. `refresh()` 는 실제로 두 가지 작업을 수행합니다:

- 1) 각 창의 `noutrefresh()` 메서드를 호출하여 원하는 화면 상태를 나타내는 하부 데이터 구조를 갱신합니다.
- 2) `doupdate()` 함수를 호출하여 데이터 구조에 기록된 원하는 상태와 일치하도록 물리적 화면을 변경합니다.

대신 여러 창에서 `noutrefresh()` 를 호출하여 데이터 구조를 갱신한 다음, `doupdate()` 를 호출하여 화면을 갱신할 수 있습니다.

## 4 텍스트 표시

C 프로그래머의 관점에서 보면, `curses`는 때때로 모두 조금씩 다른 함수의 미로처럼 보일 수 있습니다. 예를 들어, `addstr()` 은 `stdscr` 창의 현재 커서 위치에 문자열을 표시하고, `mvaddstr()` 은 문자열을 표시하기 전에 주어진 `y,x` 좌표로 먼저 이동합니다. `waddstr()` 은 `addstr()` 과 비슷하지만, 기본적으로 `stdscr`을 사용하는 대신 사용할 창을 지정할 수 있습니다. `mvwaddstr()` 은 창과 좌표를 모두 지정할 수 있습니다.

다행히 파이썬 인터페이스는 이러한 모든 세부 사항을 숨깁니다. `stdscr`은 다른 것과 마찬가지로 창 객체이며, `addstr()` 과 같은 메서드는 여러 인자 형식을 허용합니다. 일반적으로 네 가지 형식이 있습니다.

형식	설명
<code>str</code> 또는 <code>ch</code>	현재 위치에 문자열 <code>str</code> 이나 문자 <code>ch</code> 를 표시합니다
<code>str</code> 또는 <code>ch, attr</code>	현재 위치에 속성 <code>attr</code> 을 사용하여 문자열 <code>str</code> 이나 문자 <code>ch</code> 를 표시합니다
<code>y, x, str</code> 또는 <code>ch</code>	창에서 <code>y,x</code> 위치로 이동하고, <code>str</code> 이나 <code>ch</code> 를 표시합니다
<code>y, x, str</code> 또는 <code>ch, attr</code>	창에서 <code>y,x</code> 위치로 이동하고, <code>attr</code> 속성을 사용하여 <code>str</code> 이나 <code>ch</code> 를 표시합니다

속성을 사용하면 굵은 체, 밑줄, 반전 코드 또는 색상과 같은 강조 표시된 형태로 텍스트를 표시할 수 있습니다. 이에 대해서는 다음 서브 섹션에서 자세히 설명합니다.

`addstr()` 메서드는 표시할 값으로 파이썬 문자열이나 바이트열을 주합니다. 바이트열의 내용은 그대로 터미널로 전송됩니다. 문자열은 창의 `encoding` 어트리뷰트 값을 사용하여 바이트열로 인코딩됩니다; 이 어트리뷰트의 기본값은 `locale.getencoding()` 에 의해 반환되는 기본 시스템 인코딩입니다.

`addch()` 메서드는 길이가 1인 문자열, 길이가 1인 바이트열 또는 정수일 수 있는 문자를 취합니다.

확장 문자를 위한 상수가 제공됩니다; 이 상수는 255보다 큰 정수입니다. 예를 들어, `ACS_PLMINUS`는 `+/-` 기호이고, `ACS_ULCORNER`는 상자의 왼쪽 위 모서리입니다 (경계를 그리기에 편리합니다). 적절한 유니코드 문자를 사용할 수도 있습니다.

창은 마지막 조작 후 커서가 있던 위치를 기억하므로, `y,x` 좌표를 생략하면 마지막 조작이 중단된 위치에 문자열이나 문자가 표시됩니다. `move(y, x)` 메서드로 커서를 이동할 수도 있습니다. 일부 터미널은 항상 깜빡이는 커서를 표시하기 때문에, 방해받지 않는 위치에 커서를 놓아야 합니다; 임의의 위치에서 커서가 깜빡이는 것은 혼란스러울 수 있습니다.

응용 프로그램에 깜빡이는 커서가 전혀 필요하지 않으면, `curs_set(False)` 를 호출하여 보이지 않게 할 수 있습니다. 이전 `curses` 버전과의 호환성을 위해, `curs_set()` 과 동의어인 `leaveok(bool)` 함수가 있습니다. `bool` 이 참이면, `curses` 라이브러리는 깜빡이는 커서를 억제하려고 시도하고, 커서를 부적절한 위치에 두는 것에 대해 걱정할 필요가 없습니다.

## 4.1 속성과 색상

문자는 다른 방식으로 표시될 수 있습니다. 텍스트 기반 응용 프로그램의 상태 줄(status line)은 일반적으로 반전 비디오로 표시되거나 텍스트 뷰어에서 특정 단어를 강조 표시해야 할 수 있습니다. `curses`는 화면에 있는 각 셀의 속성을 지정할 수 있도록 하여 이를 지원합니다.

속성은 정수이며, 각 비트는 다른 속성을 나타냅니다. 여러 속성 비트가 설정된 텍스트를 표시하려고 시도할 수 있지만, `curses`는 가능한 모든 조합을 사용할 수 있거나 시각적으로 구별됨을 보증하지 않습니다. 사용하는 터미널의 기능에 따라 다르므로, 여기에 나열된 가장 일반적으로 사용 가능한 속성을 고수하는 것이 가장 안전합니다.

속성	설명
<code>A_BLINK</code>	깜박거리는 텍스트
<code>A_BOLD</code>	매우 밝거나 굵은 텍스트
<code>A_DIM</code>	질만 밝기의 텍스트
<code>A_REVERSE</code>	반전 비디오 텍스트
<code>A_STANDOUT</code>	사용 가능한 최고 강조 표시 모드
<code>A_UNDERLINE</code>	밑줄이 그어진 텍스트

따라서 화면 상단 줄에 반전 비디오 상태 줄을 표시하려면, 다음과 같이 코딩할 수 있습니다:

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

`curses` 라이브러리는 또한 색상을 제공하는 터미널에서 색상을 지원합니다. 이러한 터미널 중 가장 일반적인 터미널은 리눅스 콘솔이고, 그다음은 컬러 `xterm`입니다.

색상을 사용하려면, `initscr()` 을 호출한 직후 `start_color()` 함수를 호출하여, 기본 색상 집합을 초기화해야 합니다 (`curses.wrapper()` 함수는 이것을 자동으로 수행합니다). 일단 이렇게 하면, `has_colors()` 함수는 사용 중인 터미널이 실제로 색상을 표시할 수 있으면 `True`를 반환합니다. (참고: `curses`는 캐나다/영국 철자법 'colour' 대신 미국식 철자법 'color' 를 사용합니다. 영국 철자법에 익숙하다면 이러한 함수를 위해 철자법을 바꾸는 것을 감수해야 합니다.)

`curses` 라이브러리는 전경(또는 텍스트)색과 배경색을 포함하여 유한한 수의 색 쌍을 유지합니다. `color_pair()` 함수를 사용하여 색상 쌍에 해당하는 속성값을 얻을 수 있습니다; 이것은 `A_REVERSE`

와 같은 다른 속성과 비트별 OR 될 수 있지만, 다시 한번, 이러한 조합이 모든 터미널에서 작동하는 것은 아닙니다.

색상 쌍 1을 사용하여 텍스트 줄을 표시하는 예:

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

앞에서 말했듯이, 색상 쌍은 전경색과 배경색으로 구성됩니다. `init_pair(n, f, b)` 함수는 색상 쌍 `n`의 정의를 전경색 `f`와 배경색 `b`로 변경합니다. 색상 쌍 0은 검은 배경에 흰 전경으로 강제되어 있으며 변경할 수 없습니다.

색상은 번호가 매겨지며, `start_color()`는 색상 모드를 활성화할 때 8가지 기본 색상을 초기화합니다. 0:검정(black), 1:빨강(red), 2:녹색(green), 3:노랑(yellow), 4:파랑(blue), 5:자홍색(magenta), 6:청록색(cyan) 및 7:하양(white)입니다. `curses` 모듈은 `curses.COLOR_BLACK`, `curses.COLOR_RED` 등 각 색상에 대해 이름 붙인 상수를 정의합니다.

이 모든 것을 써봅시다. 색상 1을 흰색 배경의 빨간색 텍스트로 변경하려면, 다음과 같이 호출할 수 있습니다:

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

색상 쌍을 변경할 때, 해당 색상 쌍을 사용하여 이미 표시된 텍스트가 새 색상으로 변경됩니다. 이 색상으로 새 텍스트를 표시 할 수도 있습니다:

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

매우 멋진 터미널은 실제 색상의 정의를 주어진 RGB 값으로 변경할 수 있습니다. 이를 통해 일반적으로 빨간색인 색상 1을 보라색이나 파란색 또는 원하는 어떤 색상으로도 변경할 수 있습니다. 불행히도, 리눅스 콘솔은 이것을 지원하지 않아서, 저는 시도해 볼 수 없고 예제를 제공할 수 없습니다. `can_change_color()`를 호출하여 여러분의 터미널이 이를 수행 할 수 있는지를 확인할 수 있습니다. 기능이 있으면 `True`를 반환합니다. 이러한 재능있는 터미널을 보유할 만큼 운이 좋다면, 자세한 내용은 시스템 매뉴얼 페이지를 참조하십시오.

## 5 사용자 입력

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.)

창에서 입력을 얻는 메서드는 두 가지가 있습니다.:

- `getch()`는 화면을 새로 고친 다음 사용자가 키를 누를 때까지 기다립니다. `echo()`가 이전에 호출되었으면 키를 표시합니다. 일시 정지하기 전에 커서를 이동시킬 좌표를 선택적으로 지정할 수 있습니다.
- `getkey()`는 같은 작업을 수행하지만, 정수를 문자열로 변환합니다. 개별 문자는 1-문자 문자열로 반환되며, 기능 키와 같은 특수키는 `KEY_UP`이나 `^G`와 같은 키 이름을 포함하는 더 긴 문자열을 반환합니다.

`nodelay()` 창 메서드를 사용하여 사용자를 기다리지 않을 수 있습니다. `nodelay(True)` 이후에는, 창의 `getch()`와 `getkey()`가 블로킹 되지 않습니다. 입력이 준비되지 않았다는 신호를 보내기 위해 `getch()`는 `curses.ERR(-1 값)`을 반환하고 `getkey()`는 예외를 발생시킵니다. `halfdelay()` 함수도 있는데, 각 `getch()`에 (효과적으로) 타이머를 설정하는 데 사용할 수 있습니다; 지정된 지연 시간(10분의 1초로 측정됩니다) 내에 입력이 없으면 `curses`는 예외를 발생시킵니다.

`getch()` 메서드는 정수를 반환합니다; 0에서 255 사이이면, 누른 키의 ASCII 코드를 나타냅니다. 255보다 큰 값은 Page Up, Home 또는 커서 키와 같은 특수 키입니다. 반환 값을 `curses.KEY_PPAGE`, `curses.KEY_HOME` 또는 `curses.KEY_LEFT`와 같은 상수와 비교할 수 있습니다. 프로그램의 메인 루프는 이런 식입니다:



```
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
```

curses.ascii 모듈은 정수나 1문자 문자열 인자를 취하는 ASCII 클래스 멤버십 함수를 제공합니다; 이러한 루프에서 더 읽기 쉬운 검사를 작성하는 데 유용 할 수 있습니다. 이것은 또한 정수나 1문자 문자열 인자를 취하고 같은 유형을 반환하는 변환 함수를 제공합니다. 예를 들어, curses.ascii.ctrl() 은 인자에 해당하는 제어 문자를 반환합니다.

전체 문자열을 꺼내는 메서드도 있습니다, getstr(). 기능이 상당히 제한되어 있기 때문에 자주 사용되지 않습니다; 사용 가능한 편집 키는 백스페이스키와 문자열을 종료하는 Enter 키뿐입니다. 고정된 수의 문자로 선택적으로 제한될 수 있습니다.

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

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

curses.textpad 모듈은 Emacs와 같은 키 바인딩 집합을 지원하는 텍스트 상자를 제공합니다. Textbox 클래스의 다양한 메서드는 입력 유효성 검증을 사용한 편집과 후행 공백이 있거나 없는 편집 결과 수집을 지원합니다. 예를 들면 다음과 같습니다:

```
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()
```

자세한 내용은 curses.textpad의 라이브러리 설명서를 참조하십시오.

## 6 추가 정보

이 HOWTO는 화면의 내용을 읽거나 xterm 인스턴스에서 마우스 이벤트를 캡처하는 등의 고급 주제를 다루지 않지만, curses 모듈의 파이썬 라이브러리 페이지는 이제 어느 정도 완전합니다. 다음으로 그 페이지를 보십시오.

curses 함수의 자세한 동작에 대해 확신이 없으면, curses 구현(ncurses이건 독점 유닉스 벤더의 것이건)에 대한 매뉴얼 페이지를 참조하십시오. 매뉴얼 페이지는 모든 뒤틀림(quirks)을 문서화하고, 사용 가능한 모든 함수, 속성 및 ACS\_\* 문자의 전체 목록을 제공합니다.

curses API가 아주 크기 때문에, 일부 함수는 파이썬 인터페이스에서 지원되지 않습니다. 종종 구현하기가 어렵기 때문이 아니라, 아직 아무도 원하지 않았기 때문입니다. 또한, 파이썬은 ncurses와 관련된 메뉴 라

이브러리를 아직 지원하지 않습니다. 이들에 대한 지원을 추가하는 패치를 환영합니다; 파이썬에 패치를 제출하는 방법에 대한 자세한 내용은 [파이썬 개발자 지침서](#)를 참조하십시오.

- [Writing Programs with NCURSES](#): C 프로그래머를 위한 긴 자습서.
- [ncurses 매뉴얼 페이지](#)
- [The ncurses FAQ](#)
- “Use curses… don’ t swear”: curses나 Urwid를 사용하여 터미널을 제어하는 PyCon 2013 발표 비디오.
- “Console Applications with Urwid”: Urwid를 사용하여 작성된 몇몇 응용 프로그램을 보여주는 PyCon CA 2012 발표 비디오.