Programmazione in curses con Python

Release 2.01

A.M. Kuchling, Eric S. Raymond

12 dicembre 2003

amk@amk.ca, esr@thyrsus.com

Sommario

Questo vademecum mostra come scrivere programmi in modalità testuale con Python 2.x, usando il modulo di estensione `curses` per il controllo dello schermo. Traduzione italiana a cura di Carlos (enne.enne at tiscalinet.it) e Ferdinando Ferranti (zappagalattica at inwind.it).

È disponibile, in lingua originale, presso la pagina dei Python HOWTO http://www.python.org/doc/howto, la traduzione presso la pagina http://www.zonapython.it/doc/howto/.

Indice

1 Che cos'è curses?

La libreria curses consente ai terminali testuali (come VT100, la console di Linux e i terminali simulati di programmi per X11, quali xterm e rxvt) di gestire schermo e tastiera indipendentemente dal terminale. Gli schermi dei terminali supportano vari codici di controllo per compiere operazioni comuni, come muovere il cursore, scorrere lo schermo e cancellare aree. Terminali diversi usano codici molto differenti e spesso hanno i propri piccoli capricci.

In un mondo di schermate di X, qualcuno dirà "Embè?" Ora, è vero che i terminali a caratteri-cella sono tecnologia obsoleta, ma ci sono ancora nicchie in cui la capacità di fare con essi ciò che si preferisce è preziosa: per esempio, in piccoli sistemi e macchine con Unix sprovvisti di serventi X, o quella di strumenti per installazione di SO e configurazione del kernel, la cui esecuzione potrebbe rendersi necessaria prima che X sia disponibile.

La libreria curses nasconde i dettagli di tutti i diversi terminali e dà al programmatore un porzione di schermo contenente numerose finestre non sovrapposte. Il contenuto di una finestra può essere cambiato in vari modi,

(aggiungendo o cancellando del testo o variandone l'aspetto) la libreria calcolerà automaticamente quali codici di controllo inviare al terminale, per generare il giusto risultato.

Essa venne scritta, in origine, per BSD Unix; le successive versioni System V di Unix, della AT&T, aggiunsero migliorie e nuove funzioni. Curses di BSD non è più mantenuta, poiché sostituita da ncurses, realizzazione a sorgente aperto dell'interfaccia della AT&T. Un sistema Unix a sorgente aperto, come Linux o FreeBSD, quasi sicuramente usa ncurses e, poiché le più diffuse versioni commerciali di Unix sono basate su codice di tipo System V, probabilmente potrà disporre di tutte le funzioni qui descritte. Ciononostante, può darsi che versioni più vecchie di curses fornite da alcuni Unix proprietari non garantiscano pieno supporto.

Non esiste un port per Windows del modulo curses; per questa piattaforma, si provi il modulo Console, scritto da Fredrik Lundh e disponibile presso il sito http://effbot.org/efflib/console, che dà un risultato testuale indirizzabile col cursore e un pieno supporto per le istruzioni di mouse e tastiera.

1.1 Il modulo curses di Python

Il modulo di Python è un semplicissimo involucro delle funzioni in C di curses; avendo già nozioni di programmazione di curses in C, è davvero facile trasferirle in Python: la più grande differenza è che l'interfaccia di Python facilita le cose, unendo varie funzioni di C, come `addstr`, `mvaddstr`, `mvwaddstr`, nel solo metodo `addstr()`. Ma di questo si tratterà oltre.

Questo vademecum, pur fornendo delle idee di base, è semplicemente un'introduzione alla scrittura di programmi in modalità testuale con curses e Python e non vuole essere un manuale completo alle API di curses; in proposito, cfr. la guida della libreria di Pyhton e le pagine di manuale di C su ncurses.

2 Avviare e fermare un'applicazione in curses

Anzitutto, curses dev'essere avviata; basta richiamare la funzione `initscr()`, che determina il tipo di terminale, inviando ad esso il codice di configurazione richiesto e creando varie strutture di dati interni. In caso di successo, ne risulta un oggetto finestra che rappresenta l'intero schermo; di solito, questo viene chiamato `stdscr`, dal nome della corrispondente variabile del linguaggio C.

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

Per lo più, le applicazioni in curses escludono la visualizzazione automatica su schermo (echo) dei tasti premuti, e li mostra solo in certe circostanze; a tal fine, bisogna richiamare la funzione `noecho()`.

```
curses.noecho()
```

Grazie alla modalità chbreak, contrariamente alla consueta modalità d'inserimento bufferizzata, le applicazioni reagiscono istantaneamente ai tasti, senza richiedere che venga premuto quello di Invio (Enter).

```
curses.cbreak()
```

Abitualmente, i terminali convertono la pressione di tasti speciali, come quelli di direzione o PgUp e Home, in forma di sequenze multiple di caratteri escape. Per evitarlo, si dovrebbe scrivere l'applicazione in modo che le riconosca e le tratti di conseguenza; invece, abilitando la modalità keypad, curses provvede a ciò, restituendo un valore speciale come `curses.KEY_LEFT`.

```
stdscr.keypad(1)
```

Fermare un'applicazione in curses è più semplice che avviarla, basta chiamare:

```
curses.nocbreak(); stdscr.keypad(0); curses.echo()
```

per annullare le impostazioni di terminale adatte a curses. Si chiami `endwin()` per ripristinare la modalità operativa d'origine.

```
curses.endwin()
```

Un problema frequente, in sede di verifica di un'applicazione in curses, è ritrovarsi un terminale in disordine, quando l'applicazione muore senza averlo riportato allo stato precedente. In Python, questo accade quando il codice è bacato e solleva un'eccezione: per esempio la digitazione potrebbe non produrre più alcun risultato sullo schermo, e questo renderebbe difficile l'uso della shell.

In Python si possono evitare tali complicazioni e facilitare di molto la correzione, importando il modulo `curses.wrapper`, un involucro (`wrapper`) per funzioni che richiedono un argomento d'aggancio (hook). Esso compie l'avvio già descritto (inizializzando anche i colori, se c'è il relativo supporto), esegue quindi l'aggancio e, infine, arresta in modo corretto. L'aggancio è richiamato all'interno di una clausola di verifica che intercetta le eccezioni e arresta curses, scavalcando le eccezioni medesime. Così si evita il blocco del terminale in un bizzarro stato interlocutorio.

3 Finestre e taccuini

In curses le finestre sono l'idea di base. Un oggetto finestra rappresenta un'area rettangolare di schermo e permette all'utente di visualizzare, cancellare, inserire stringhe di testo, e così via, in vari modi.

L'oggetto `stdscr` restituito dalla funzione `initscr()` è un oggetto finestra che copre l'intero schermo. A molti programmi quest'unica finestra basta già, ma può anche darsi che si desideri dividere lo schermo in finestre più piccole, per ridisegnarle o svuotarle separatamente. La funzione `newwin()` crea una nuova finestra di una certa dimensione, restituendo il nuovo oggetto finestra.

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

A proposito del sistema di coordinate usato in curses: sono scritte sempre nell'ordine *y,x* e la coordinata 0,0 indica l'angolo superiore sinistro; questo contravviene alla notazione comune, in cui si segna per prima la coordinata *x*; è un'infelice difformità rispetto alla maggior parte delle altre applicazioni, ma fa parte di curses dalla sua prima scrittura ed è tardi per cambiare.

Quando si invoca un metodo per visualizzare o cancellare del testo, l'effetto non si manifesta immediatamente, poiché curses è stata scritta, all'origine, pensando alle lente connessioni di terminale a 300 baud: con quei terminali, era molto importante minimizzare il tempo necessario a ridisegnare lo schermo. Ciò permette a curses di accumulare cambiamenti sullo schermo e visualizzarli nel modo più efficiente. Per esempio, se un programma scrive dei caratteri in una finestra ma la svuota subito, non c'è bisogno di visualizzarli, perché non persisterebbero che per pochi istanti sullo schermo.

Di conseguenza, curses richiede che gli si dica esplicitamente di ridisegnare le finestre, mediante il metodo `refresh()` degli oggetti finestra. In pratica, ciò non complica di molto la programmazione con curses. La maggior parte dei programmi ha un'attività frenetica e poi si ferma in attesa di una digitazione o di un'altra azione dell'utente. Bisogna solo assicurarsi che lo schermo sia stato ridisegnato prima della pausa in attesa di istruzioni, invocando `stdscr.refresh()` o il metodo `refresh()` per qualche altra finestra rilevante.

Un taccuino è una finestra di tipo speciale: può essere più ampio dello schermo reale ed essere visualizzato solo un settore per volta. Creare un taccuino richiede che se ne stabiliscano l'altezza e la larghezza, mentre il suo aggiornamento necessita delle coordinate dell'area dello schermo in cui sia visualizzata una sua sezione.

```
pad = curses.newpad(100, 100)
#  Questi cicli di azioni riempiono il taccuino di lettere; questo è
#  spiegato nella sezione seguente
for y in range(0, 100):
    for x in range(0, 100):
        try: pad.addch(y,x, ord('a') + (x*x+y*y) % 26 )
        except curses.error: pass

#  Visualizza una sezione di taccuino al centro dello schermo
pad.refresh( 0,0, 5,5, 20,75)
```

L'invocazione di `refresh()` visualizza una sezione del taccuino nel rettangolo fra le coordinate (5,5) e (20,75) sullo schermo; l'angolo superiore sinistro della sezione visualizzata è il punto (0,0) del taccuino. Eccettuata questa differenza, i taccuini sono esattamente come le normali finestre e supportano gli stessi metodi.

Se sullo schermo ci sono molte finestre e taccuini, un sistema più efficiente evita il noioso sfarfallio dello schermo ad ogni aggiornamento di schermo: si usino i metodi `noutrefresh()` e/o `noutrefresh()` su ogni finestra, per aggiornare la struttura di dati che indica lo stato desiderato dello schermo; quindi, si modifichi di conseguenza lo schermo fisico in una sola volta chiamando la funzione `doupdate()`. Il normale metodo `refresh()` invoca `doupdate()` come passo finale.

4 Visualizzare del testo

Dal punto di vista di un programmatore in C, curses può talvolta sembrare un tortuoso dedalo di funzioni sottilmente diverse. Ad esempio, `addstr()` crea una stringa all'attuale posizione del cursore nella finestra `stdscr`, mentre `mvaddstr()` va ad una certa coordinata y,x prima di visualizzare la stringa. `waddstr()` è proprio come `addstr()`, ma permette di specificare la finestra da usare, invece di usare a priori `stdscr`. `mvwaddstr()` agisce di conseguenza.

Fortunatamente, l'interfaccia di Python nasconde tutti i dettagli; `stdscr` è una finestra come qualunque altra e metodi come `addstr()` accettano diverse forme di argomento: di solito ce ne sono quattro differenti.

Forma	**Descrizione**
str o *ch*	Visualizza la stringa str *str* o il carattere *ch*
str o *ch*, *attr*	Visualizza la stringa *str* o il carattere *ch*, usando l'attributo *attr*
y, *x*, *str* o *ch*	Va alla posizione *y,x* nella finestra e visualizza *str* o *ch*
y, *x*, *str* o *ch*, *attr*	Va alla posizione *y,x* nella finestra e visualizza *str* o *ch*, usando l'attributo *attr*

Gli attributi permettono di visualizzare del testo evidenziato, sia esso grassetto, sottolineato, in negativo o a colori. Cfr. la prossima sezione per maggiori dettagli.

La funzione `addstr()` assume una stringa di Python come valore da visualizzare, la funzione `addch()` un carattere, che può essere sia una stringa di Python di lunghezza 1, sia un numero intero. Se è una stringa, si possono visualizzare caratteri solo fra 0 e 255. Curses SVr4 fornisce, come costanti per caratteri di estensione, numeri interi maggiori di 255. Ad esempio, `ACS_PLMINUS` è un simbolo di +/- e `ACS_ULCORNER` è l'angolo superiore sinistro di una casella (utile, per disegnare cornici).

Le finestre conservano la posizione del cursore dopo l'ultima operazione, così che, se non si danno nuove coordinate *y,x*, la stringa o il carattere saranno visualizzati in quel punto. Si può spostare il cursore anche con il metodo `move(`*y,x*`)`. Poiché alcuni terminali presentano sempre un cursore lampeggiante, si potrebbe desiderare che lo facciano dove non costituisca una distrazione, il che è meglio dell'avere un cursore lampeggiante che appare a casaccio.

Se, poi, esso non è affatto essenziale per l'applicazione, si possono invocare le funzioni `curs_set(0)` o, ciò che ha eguale effetto, ma è compatibile con versioni più vecchie di curses, `leaveok(`*bool*`)`. Quando il valore *booleano* è vero, curses cerca di eliminare il cursore lampeggiante e non ci si deve più preoccupare di lasciarlo in punti strani.

4.1 Attributi e colori

I caratteri possono essere visualizzati in vari modi. Le linee di stato, in un'applicazione testuale, di solito sono in negativo; ad un visualizzatore di testo può essere necessario evidenziare certe parole: curses supporta ciò, permettendo di specificare un attributo per ogni cella dello schermo.

Un attributo è un numero intero (ogni bit rappresenta un diverso attributo); si può provare a visualizzare un testo con una serie di bit dai molteplici attributi, ma curses non garantisce la disponibilità o la distinzione visiva di tutte le loro possibili combinazioni: dipende dalla flessibilità d'uso del terminale, per cui meglio attenersi alla seguente tabella che elenca quelli più usati.

Attributo	Descrizione
A_BLINK	Testo lampeggiante
A_BOLD	Evidenziato o grassetto
A_DIM	Testo oscurato
A_REVERSE	Testo in negativo
A_STANDOUT	La migliore evidenziazione possibile
A_UNDERLINE	Sottolineato

Così, per visualizzare una linea di stato in negativo alla prima riga dello schermo, si potrebbe digitare il seguente codice:

```
stdscr.addstr(0, 0, "Attuale modalità: Scrittura",
      curses.A_REVERSE)
stdscr.refresh()
```

La libreria curses supporta anche il colore, sui terminali che possono offrirlo, il più comune dei quali è, probabilmente, la console di linux, seguita dagli xterm a colori.

Per usare i colori, subito dopo `initscr()` bisogna invocare la funzione `start_color()`, che avvia la serie di colori predefinita - a questo provvede automaticamente la funzione `curses.wrapper.wrapper()`; ciò fatto, la funzione `has_colors()` restituisce il valore TRUE (vero), se il terminale in uso può visualizzare di fatto i colori. (Nota da AMK: curses usa la grafia americana 'color', invece della Canadese/Inglese 'colour': ci si rassegni all'errore, in cambio delle funzioni).

La libreria di curses mantiene un numero limitato di coppie di colori, che comprendono un colore in primo piano (o di testo) e uno di sfondo. Con la funzione `color_pair()` si può ottenere il valore di attributo corrispondente a una coppia di colori; mediante bitwise-OR ("|"), questa può essere ancora arricchita di altri attributi come A_REVERSE, ma, ancora una volta, non è detto che tali combinazioni funzionino su tutti i terminali.

Un esempio, che visualizza una riga di testo usando la coppia di colori 1:

```
stdscr.addstr( "Forza Genoa", curses.color_pair(1) )
stdscr.refresh()
```

Come già visto, una coppia di colori consta di uno in primo piano e uno di sfondo. Quando attiva, la modalità colore, `start_color()` avvia gli 8 colori base, cioè: 0:nero, 1:rosso, 2:verde, 3:giallo, 4:blu, 5:magenta, 6:azzurro e 7:bianco. Il modulo di curses definisce per ognuno dei colori una costante che ne porta il nome inglese: `curses.COLOR_BLACK`, `curses.COLOR_RED`, e così via.

La funzione `init_pair(`*n, f, b*`)` passa dalla definizione della coppia di colori *n* ad una con f in primo piano e b sullo sfondo. La coppia 0 è collegata rigidamente al bianco o al nero e non può essere cambiata.

Ricapitolando, per passare da colore 1 a testo rosso su sfondo bianco, si dovrebbe invocare:

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

Quando si cambia una coppia di colori, qualunque testo visualizzato con essa passerà ai nuovi, con i quali si può anche creare un nuovo testo, tramite:

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

Terminali davvero fantastici possono passare dalle definizioni degli attuali colori a un dato valore RGB; ciò permette di cambiare color 1 (solitamente rosso) in viola o blu o quello che si preferisce, ma purtroppo la console di linux non lo consente, sicché non posso provarla, né fornire alcun esempio. Si controlli se il proprio terminale può farlo, invocando `can_change_color()`, che restituisce TRUE in caso positivo. Se si ha la fortuna di avere un terminale del genere, per maggiori ragguagli si vedano le pagine man del sistema.

5 Input dall'utente

La libreria curses offre già dei semplici meccanismi di input; il supporto di Python aggiunge dei widget text-input che ne colmano le lacune.

Il modo più comune per catturare un input in una finestra è usare il suo metodo `getch()`, che opera una pausa e attende che l'utente prema un tasto, visualizzandolo, se prima è stato invocato `echo()`. A scelta, si può specificare la coordinata su cui spostare il cursore prima della pausa.

Si può cambiare tale comportamento con il metodo `nodelay()`. Dopo `nodelay(1)`, `getch()` per la finestra diventa non-bloccante e restituisce ERR (-1) quando non ci sono istruzioni. C'è anche la funzione `halfdelay()`, che può essere usata per impostare (effettivamente) un intervallo per ogni `getch()`; se entro il numero di millisecondi specificato come argomento di `halfdelay()` non ci sono istruzioni disponibili, curses solleva un'eccezione.

Il metodo `getch()` restituisce un numero intero che, se compreso fra 0 e 255, rappresenta il codice ASCII del tasto premuto. Valori superiori a 255 indicano tasti speciali come Page Up, Home, o i tasti cursore. Si può confrontare il valore restituito con costanti come `curses.KEY_PPAGE`, `curses.KEY_HOME`, o `curses.KEY_LEFT`. Di solito, il ciclo principale del programma è simile a:

```
while 1:
    c = stdscr.getch()
    if c == ord('p'): PrintDocument()
    elif c == ord('q'): break  # Esce dal ciclo while()
    elif c == curses.KEY_HOME: x = y = 0
```

Il `curses.ascii` offre funzioni appartenenti alla classe ASCII, che assumono come argomenti sia numeri interi, sia stringhe di un carattere (queste possono essere utili per scrivere test più leggibili dagli interpreti di comando) e funzioni di conversione che assumono come argomenti sia numeri interi, sia stringhe di un carattere ed in più restituiscono lo stesso tipo di dato. Ad esempio, `curses.ascii.ctrl()` restituisce il carattere di controllo corrispondente al suo argomento.

Esiste anche un metodo per ritrovare un'intera stringa: `getstr()`. Non è usato spesso, avendo una funzionalità limitata: i soli tasti disponibili sono quelli di cancella indietro (backspace) e Invio (Enter), che chiude la stringa. A scelta, può essere limitato a un dato numero di caratteri.

```
curses.echo()            # Abilita la visualizzazione dei caratteri

# ottiene una stringa di 15 caratteri col cursore sulla prima riga in alto
s = stdscr.getstr(0,0, 15)
```

Il modulo di Python `curses.textpad` offre qualcosa di meglio: con esso, si può trasformare una finestra in una casella di testo che supporta una serie di tasti-funzione in stile Emacs. Vari metodi della classe `Textbox` supportano la modifica con convalida dell'istruzione e raccolta dei risultati con o senza tenere traccia degli spazi. Cfr. la documentazione della libreria su `curses.textpad`.

6 Per maggiori informazioni

Questo vademecum non ha affrontato alcuni argomenti avanzati, come il “raschiare” lo schermo o catturare gli eventi del mouse da un’istanza xterm, ma ora la pagina della libreria di Python per il moduli di curses è abbastanza completa. Sarebbe utile darle un’occhiata, in futuro.

In caso di dubbi sul preciso comportamento di qualunque voce di ncurses, cfr. le pagine del manuale sull’esecuzione di curses, che si tratti di ncurses o di Unix proprietari. Esse ne documenteranno i vari ghiribizzi, fornendo un lista completa di tutte le funzioni, gli attributi e i caratteri `ACS_*` disponibili.

Siccome l’API di curses è così ampia, nell’interfaccia Python alcune funzioni non sono supportate, non perché difficili da eseguire, ma perché non se n’è ancora sentito il bisogno. Si aggiungano pure e si invii la correzione relativa. Ancora, non c’è supporto per i menu o le librerie per i pannelli associate a ncurses; chi vuole, può aggiungerle.

Se qualcuno scrive un programmino interessante, lo mandi come demo senza alcun timore. Così possiamo usarne sempre di più!

Le FAQ di ncurses: http://dickey.his.com/ncurses/ncurses.faq.html