Python-ablakok szöveges módban

Felhasználói felületek készítése szöveges módban a curses programkönyvtár segítségével.

Pythonos sorozatunk e részében a szövegalapú felhasználói felületek készítésének fortélyaival ismerkedünk meg. Látni fogjuk, hogy ezúttal talán még egyszerűbb dolgunk lesz, mint grafikus megfelelőik esetében, köszönhetően nagyrészt a *curses* bővítmény egyszerűen használható függvényeinek.
Linux és Unixok alatt általában sokféle terminháltípus létezik, és mindegyiknek megvannak a saját vezérlőkarakterei. Így ha a *curses* nélkül kezdenénk neki szöveges módú alkalmazások készítésének, bizony jókora terhet vennénk a nyakunkba. Ezt a zűrzavart elkerülendő jött létre a *curses* könyvtár, mellyel nincs szükségünk a megjelenítő részletekbe menő ismeretére, elég, ha azzal tisztában vagyunk, hogy a *curses*-t hogyan kell megszelídíteni.
Óhatatlanul felmerülhet az emberben a kérdés, hogy mi szükség van szöveges módú alkalmazásokra most, a grafikus operációs rendszerek korában? A válasz egyszerű: egy szöveges módú alkalmazás esetében nem kell segédprogramok garmadáját feltelepíteni, nincs szükség X-kiszolgálóra, ráadásul tárhelyet és a kisebb gépek esetén erőforrásokat spórolhatunk ezzel a megoldással! Nem is beszélve arról, hogy sokan még a gondolattól is ódzkodnak, hogy valamilyen élesben működő kiszolgálóra grafikus programot telepítsenek.
A DOS-on felnőtt nemzedék még biztosan emlékezik a Borland egyszerűen használható, tetszetős TurboVision programjára. TurboVisionben készíthetünk menüket, ablakokat, gombokat, játszhatunk a színekkel, létrehozhatunk gördítősávval ellátott szövegmezőket, vagy amit csak akarunk. A *curses* ugyan nem nyújt ennyi szolgáltatást, de létezik hozzá egy sor kiegészítés, melyek pont ezt hivatottak pótolni.
A curses a Python terjesztésnek lényeges részét képezi, így a feltelepítésével nem kell külön foglalkoznunk. A modulban található függvények pedig teljes egészében a rokon C könyvtárban található függvényeknek felelnek meg, egy-két egyszerűsítéstől eltekintve. Ilyen különbség például, hogy Pythonban csupán egyetlen `addstr()` függvény létezik, ami a C könyvtár `addstr()`, `mvaddstr()` és `mvwaddstr()` függvényeit helyettesíti.
Most pedig vágjunk bele!

A kezdő lépések

A *curses* bővítmény programunkba illesztését (import) követően még el kell végeznünk néhány dolgot, mielőtt a teljes képernyőt a birtokunkba vehetnénk. Elsőként meg kell hívni az `initscr()` függvényt, ami – miután felismerte az adott terminháltípust – belső adatszerkezeteit felkészíti a használatra és létrehozza őket. A függvény egy képernyőobjektummal tér vissza.

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

Az `initscr()` által visszaadott objektumot `stdscr`-nek szokás hívni, ami lefedi a programunk által használható teljes felületet.

Következő lépésként beállítjuk, hogy a lenyomott billentyűkhöz a képernyőn ne jelenjen meg karakter, vagyis csak abban az esetben, ha erre a *curses* valamilyen kiíró függvényen keresztül külön utasítást kap:

```
curses.noecho()
```

Megszokhattuk, hogy konzolon általában akkor történik valami, ha a begépelt parancssor végén ENTER-t nyomunk. Ebben az esetben azonban minden egyes lenyomott karaktert programunk csak az ENTER billentyű lenyomása után észlel. Ha szükségünk van arra, hogy az egyes billentyűkre azonnal válaszolni tudjunk, a következő utasítást kell begépelnünk:

```
curses.cbreak()
```

Ennek következtében külön-külön felelhetünk minden eseményre, viszont ha a különleges billentyűket – úgymint fel, le, home, end stb. – is megfelelően szeretnénk kezelni, ezt az utasítást is adjuk ki :

```
curses.keypad(1)
```

Ezután ha benyomjuk mondjuk a balra mutató nyilat, a *curses*

1. lista Egyszerű curses program

```
1.    def finalization():
2.      stdscr.keypad(0)
3.      curses.echo()
4.      curses.nocbreak()
5.      curses.endwin()
6.
7.    from time import sleep
8.    import curses
9.    import traceback
10.
11.   try:
12.     stdscr = curses.initscr()
13.     curses.noecho()
14.     screen = stdscr.subwin(20, 30, 3 ,3)
15.     screen.box()
16.     screen.refresh()
17.
18.     sleep(3)
19.
20.     finalization()
21.   except:
22.     finalization()
23.
24.     traceback.print_exc()
```

2. lista Játék a színekkel

```
1.    import curses.wrapper
2.
3.    def teszt1(stdscr):
4.      stdscr.box()
5.
6.      curses.init_pair(1,
        ➥curses.COLOR_WHITE,
        ➥curses.COLOR_RED)
7.      stdscr.addstr(2, 2, "Hell  vilÆg!",
        ➥curses.color_pair(1) |
        ➥curses.A_BOLD)
8.
9.      stdscr.refresh()
10.
11.     ch = stdscr.getch()
12.
13.   curses.wrapper(teszt1)
```

a saját ábrázolásának megfelelően `curses.KEY_LEFT`-ként továbbítja az alkalmazásnak.
Ha a fentieket szeretnénk semmissé tenni, az egyes soroknak megfelelő ellentétes tartalmú utasításokat kell végrehajtanunk:

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

Végül pedig az `endwin()` függvénnyel a terminált az eredeti állapotába állíthatjuk vissza:

```
curses.endwin()
```

Ha ezeket a függvényeket programunk befejezésekor nem hívjuk meg, kilépés után a terminál ebben a helyzetben marad. Ilyenkor a parancsértelmezőbe kilépve zavaró, ha nem látjuk, amit írunk, és csak egy elegánsan kiadott `reset` oldja meg a gondot:

```
# reset
```

Nem is beszélve arról, hogyha a programunk például egy megszakítás következtében lép ki, a Python által megjelenített hibaüzenet is teljesen összekuszálódik. Erre a későbbiekben fogunk megoldást látni.
Itt említeném meg a Python `curses.wrapper` nevű bővítményét, ami a terminál beállításaival és visszaállításával kapcsolatos gondokat hivatott megoldani. Ha a `curses.wrapper`-rel hozunk létre egy programot, az önmagától gondoskodik a terminál megfelelő felkészítéséről, és arra is ügyel, hogy programunk befejezésekor a megfelelő állapotban adja vissza az irányítást a parancsértelmezőnek.
A *curses*-re épülő programjainkban érdemes a `traceback` bővítményt is alkalmazni, mellyel a Python által keltett hibaüzeneteket lehet nyomon követni. Használatára később mutatunk példát.

Ablakok kezelése

A *curses* megjelenítése ablakok kezelésére épül. Ha a képernyőre szeretnénk írni valamit, először egy ablakot kell létrehoznunk. A *curses* ablakai csak afféle látszólagos ablakok,

3. lista Bemenet kezelése

```
1.    def finalization():
2.      stdscr.keypad(0)
3.      curses.echo()
4.      curses.nocbreak()
5.      curses.endwin()
6.
7.    import curses
8.    import traceback
9.
10.   try:
11.     stdscr = curses.initscr()
12.     curses.noecho()
13.     curses.halfdelay(10)
14.
15.     stdscr.box()
16.     stdscr.addstr(1, 1, "KilØpØshez
        ➥nyomj 'Q'-t!")
17.
18.     ch = stdscr.getch()
19.
20.     while ch != ord('q'):
21.    stdscr.addstr(5, 2, "    ")
22.
23.          if ch >= 0 and ch <= 255:
24.              stdscr.addch(5, 2, ch)
25.          else:
26.              stdscr.addch(5, 2, ch)
27.
28.            stdscr.addstr(6, 2, "K d:
               ➥%d  " % ch)
29.
30.          ch = stdscr.getch()
31.
32.     finalization()
33.
34.   except:
35.     finalization()
36.
37.     traceback.print_exc()
```

amelyekkel a programunk felületét tagolhatjuk részekre. Emellett a grafikus felületekhez szokott felhasználónak talán elsőre szokatlan, de a *curses* ablakainak nincs mélysége.
A *curses* csupán két dimenzióban gondolkodik, így elképzelhető, hogy két – elvileg egymást fedő – ablaknak a tartalma egyszerre látszik.
Az `stdscr` a teljes képernyőt lefedő ablakot jelöli, amiben a `curses newwin()` tagfüggvényével tetszőlegesen további ablakokat hozhatunk létre. Több ablakra lehet szükségünk akkor, ha a képernyőt valamilyen okból fel szeretnénk osztani, így különítve el az egyes részek kezelését egymástól.

```
kezdo_x = 2
kezdo_y = 5
magassag = 10
szelesseg = 20
ablak = curses.newwin(magassag, szelesseg,
➥kezdo_y, kezdo_x)
ablak.box()
```

A_BLINK	Villogás
A_BOLD	Vastag betűk vagy növelt fényerő
A_DIM	Halványabb betűk
A_REVERSE	A háttérszín felcserélése az előtér színével
A_STANDOUT	A legmagasabb fényerő
A_UNDERLINE	Aláhúzás

Mint látható, a *curses* a szokásoktól eltérően az Y koordinátát veszi előre. A koordináták számozása 0-val kezdődik, vagyis a képernyő bal felső sarkában található pontot a (0, 0) koordináta jelöli. Az `ablak.box()` utasítással azt érjük el, hogy megjelenő új ablakunk köré (de még annak belsejében) egy keret rajzolódik. Ha azonban így futtatjuk le a programot , a képernyőn látszólag nem történik semmi. Valóban nem történik semmi, mivel a *curses* csak akkor rajzol a valódi képernyőre, ha az egy ablakokhoz tartozó `refresh()` tagfüggvényt meghívjuk. A `refresh()` valójában annyit tesz, hogy a `noutrefresh()` tagfüggvény segítségével a képernyő egy részét frissítésre jelöli ki, majd meghívja a `doupdate()` függvényt, ami a frissítést ténylegesen elvégzi. Ennek ismerete olyankor hasznos, amikor a képernyőn több ablak tartalmát meg akarjuk jeleníteni, és el szeretnénk elkerülni a villódzást. Ilyenkor az ablakoknak csak a `noutrefresh()` tagfüggvényét hívjuk meg, és miután mindent frissítettünk, csak egyetlen egyszer hívjuk meg a `doupdate()` tagfüggvényt, így látszólag minden egyetlen szemvillanás alatt fog előtűnni a képernyőn (*1. lista*).

A megjelenítés és a bemenet kezelése

A *curses*-ben a megjelenítésért elsősorban az `addch()` és `addstr()` függvények, valamint az `addnstr()` függvény felelősek. Ha korábban már programoztunk *curses*-t C-ben, ez talán szokatlan. Mint már említettem, a Python az ezekhez a függvényekhez tartozó rokon függvényeket elrejti, úgy, hogy a működésüket ezeken az egyszerű utasításokon keresztül valósítja meg. Nincs külön `mvaddstr()` vagy `mvwaddstr()`, amelyekkel azt szabályozhatnánk, hogy melyik ablakban és milyen koordinátákon jelenjen meg az adott szöveg, hanem mindent egyszerűen az `addstr()` tagfüggvénnyel megoldhatunk. Az `addnstr()` függvény annyiban különbözik az `addstr()`-től, hogy amennyiben ezt a függvényt használjuk, a karaktersorozatból legfeljebb csak n számú karakter íródik ki. A függvények elsőként a megjelenítendő koordinátákat várják, amelyeket természetesen el is hagyhatunk, ezt követi a karakter vagy karaktersorozat megadása, végül pedig a szöveg tulajdonságainak megadása, amelyet úgyszintén el lehet hagyni:

```
addch([y, x,] ch [, attr])
addstr([y, x,] str [, attr])
addnstr([y, x,] str, n [, attr])
```

Ha nem adjuk meg a kiírandó szöveg koordinátáit, akkor a karaktersorozat a kurzor jelenlegi helyén fog kiíródni. Minden kiírási műveletet követően a kurzor az utolsó kiírt karakter utáni helyre vándorol. Az `addnstr()` esetében azonban hiába adjuk meg az `n` kapcsolót, ha a szöveg nem megfelelően hosszú, a kurzor akkor is csak a szöveg utáni helyre kerül. A `move(y, x)` függvénnyel a kurzort tetszőleges helyre állíthatjuk, olyan helyre, ahol nem zavar az állandó villódzásával.

Ha teljesen el szeretnénk tüntetni, a `set_cur(0)` utasítást kell kiadnunk.
A szöveg színét, illetve megjelenítésének módját az utolsó kapcsoló határozza meg. A kapcsoló lehetséges értékeit *táblázatunk* tartalmazza. A táblázatban található elemek esetén nem mindegyik esetében biztos, hogy a megfelelő megjelenítési módot a használt terminál is támogatja. Az egyes kiemelési típusokból többet is megadhatunk, ha az elemeket bitenkénti vagy ("|") műveleti jellel kapcsoljuk össze.
A szín megadása az `init_pair()` és `color_pair()` függvényekkel lehetséges. Ha a `curses.wrapper` bővítmény felhasználásával készítjük a programunkat, akkor a színek kezelése – ha a terminál is támogatja – teljesen önműködő. Ha azonban a *curses* tulajdonságait saját kezűleg állítottuk be, akkor közvetlenül az `initscr()` meghívása után a `start_color()` függvényt is meg kell hívni.
A függvények használatára a *2. lista* mutat példát.
Az adatok beolvasására a `getch()` függvény áll rendelkezésre:

```
ch = stdscr.getch()
```

Alapértelmezés szerint a `getch()` az első lenyomott billentyűig vár. Ha nem szeretnénk tétlenül várakozni, használhatjuk a `nodelay()` és `halfdelay()` függvényeket. Ha a `nodelay()`-t igaz értékkel hívjuk meg, akkor amennyiben nincs kiolvasandó karakter, a `getch()` azonnal visszatér `-1` (`ERR`) értékkel. A `halfdelay()` működése hasonló, azzal a különbséggel, hogy a függvény első értékeként megadható, hogy hány tizedmásodpercig várjon billentyűleütésre.
A `getch()` visszatérési típusa egész szám típus (`integer`). Ha az értéke 0 és 255 közötti, akkor a szám valamilyen karaktert jelöl. Ha az érték kisebb nullánál, akkor valamilyen hiba történt, ha nagyobb 255-nél, akkor a `keypad(1)` függvény következtében a szám valamilyen vezérlőkaraktert jelöl, ami lehet például `curses.KEY_LEFT` vagy `curses.KEY_HOME` stb. (*3. lista*).

Összegzés

A *curses* gyorsan tanulható és egyszerű eszköz hatékony szöveges módú alkalmazások írására. Ismertségének köszönhetően az is biztosított, hogy a felhasználásával írt programok bonyodalom nélkül képesek szinte bárhol, bármilyen Unixon futni.

Gludovátz Gábor (ggabor@sopron.hu) egy soproni cég Linux-rendszerekkel foglalkozó rendszergazdája. Kedvenc időtöltése a programozás, és a Linux lelkivilágában való kutakodás. Ha ideje engedi, szívesen hódol szenvedélyének és bringáján a környező erdőket járja. Honlapja a ➲ http://www.sopron.hu/~ggabor/ címen érhető el.

Kapcsolódó címek

A Python-honlap ➲ http://www.python.org
Curses lépésről-lépésre
➲ http://py-howto.sourceforge.net/curses/curses.html
Curses-leírás
➲ http://python.org/doc/current/lib/module-curses.html