Bevezetés a Tkinter használatába (2. rész)

Egy számológép elkészítése során is fedezhetünk fel új dolgokat: tudtad, hogyha véletlenül meglököd az egeredet, máris folyamatok egész sorát indíthatod el?

Tkinter-tanfolyamunk második részéhez érkeztünk. Az előző részben már láthattuk, hogyan néz ki egy egyszerű Tkinter-alkalmazás, ezúttal pedig egy kicsit bonyolultabb feladattal, egy számológépes példával folytatjuk. A „Szia Világ!"-os példa sokat elmond, amikor egy programnyelv vagy rendszer alapjaival ismerkedünk, a Tkinter viszont túlmutat ezen; és mivel terjedőben van a szokás, hogy a grafikus alkalmazások fejlesztésére szánt rendszereket egy-egy számológépes példán keresztül ismertetik meg a felhasználókkal, tegyünk így mi is. Eközben fény derül olyan titkokra, minthogy miként tudunk egy szövegbeviteli mezőt programsorból módosítani, de az is kiderül, mi minden történik olyankor, amikor látszólag nem történik semmi, csak az egerünkkel bóklászunk a képernyőn. Vágjunk bele!

Ismerkedés a számológéppel

Számológépet már mindenki látott, ezért a feladattal nagyjából tisztában vagyunk. Vegyük sorra, mi minden szükséges ahhoz, hogy az elképzeléseinket valóra váltva egy egyszerű számológép jelenjen meg a képernyőn, amely összead és kivon, továbbá a másik két alapművelettel is tisztában van. Ha a számítógép „fejével" gondolkodunk, mindjárt elakadunk, hiszen a „szegény" gép nem tudja, hogyan néz ki egy számológép, tehát pontról pontra mindent el kell neki magyaráznunk. Meg kell mondanunk például, hogy a gombok ne „csak úgy" megjelenjenek a képernyőn, hanem meg is lehessen őket nyomni; és azt is meg kell értetnünk, hogy olyankor mi történjen, ha meg is nyomjuk ezeket a gombokat. Tudnia kell, hol legyenek az egyes gombok, a többiről nem is beszélve. Nem olyan bonyolult ám ez, csak elsőre szokatlan, hiszen ezúttal olyan partnerrel akadtunk össze, aki nem biztos, hogy mindent azonnal ugyanúgy gondol, ahogyan mi elvárnánk.

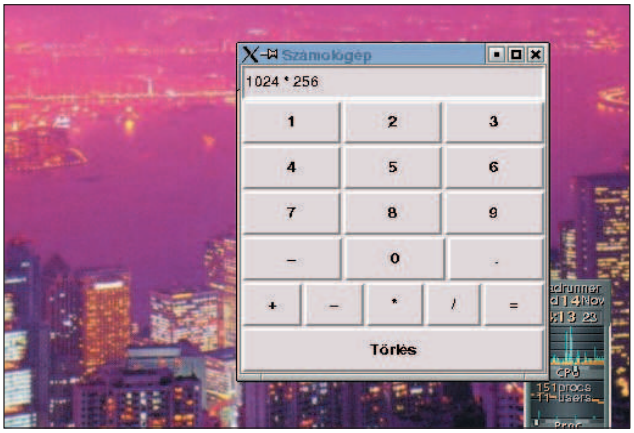
Első lépésben tehát az úgynevezett felhasználói felületet szükséges felépítenünk: ki kell találnunk, hogy milyen elemeket szeretnénk kitenni a képernyőre, és hol legyen a helyük. Esetünkben gombokra és kijelzőre lesz szükség. A következő lépésben pedig azt kell kifundálnunk, hogy az egyes gombokra kattintva mi történjék. Végül az sem árt, ha arra is felkészülünk, hogy mi legyen olyankor, ha a monitor előtt ülő felhasználó olyan dolgokat cselekszik, amelyre programunk nincs felkészülve: egyszerűen lépjünk ki valamilyen hibával a programból, ne is vegyük észre a hibát vagy tudassuk a felhasználóval, hogy rosszul csinált valamit?

Amennyiben mindezt kigondoltuk, a programtervezés nagy részén már túl is vagyunk, innentől már csak ujjgyakorlat az egész.

Az első nekifutás

Mint fentebb már említettem, számológépünkhöz két dologra lesz szükségünk: egy kijelzőre és sok-sok apró pici gombra, továbbá egy ablakra, ahol mindezt elhelyezhetjük. Az előző részben láthattuk, hogy egy gomb létrehozása csupán egy pillanatig tart: létrehozunk egy gombpéldányt és valamelyik felületkezelővel kitesszük a képernyőre. Szemléltetve:

```
w = Button(root, text= sz veg ,
           command=eljÆrÆs)
w.pack(side=LEFT, expand=YES, fill=BOTH)
```



Munkánk gyümölcse

Lambda kifejezések

Egy `lambda` kifejezés egy szokványos `def` függvénytől annyiban különbözik, hogy olyan helyeken is előfordulhat, ahol a `def` nem. Tartalma csak valamilyen egyszerű kifejezés lehet, bonyolultabb szerkezetek, mint `if` vagy `for`, nem. A `lambda` kifejezéseket elsősorban visszahívó függvényeknél használják. Visszatérési értéke mindig egy függvényobjektum, amelyet egy változóhoz rendelve hívhatunk meg. Az alábbi példa lefutása után a *b(18)* visszatérési értéke 36:

```
b = lambda a: a * 2
print b(18)
```

Minden `lambda` kifejezés saját névtérrel bír, amely nem azonos a hívó programrész névterével. Ezt a gondot egy trükkel szokták megoldani: a `lambda` értéklistájához azokat a változókat teszik hozzá, amelyekre a programrész névteréből szükségünk van. Számológépprogramunkban erre is találunk példát.

Az első sorban dől el, hogy mi lesz a gombunkra írva, és mi fog történni, ha valaki véletlenül rákattint. A `w` változó innentől kezdve az általunk megálmodott gombra mutat, ez azonban egyelőre csak a memóriában létezik. Ahhoz, hogy valódi, kattintható és látható gomb váljék belőle, ki kell tennünk a képernyőre. Ezt a nagyon fontos és cseppet sem elhanyagolható feladatot a Packer nevű felületkezelőre bízzuk. Ennek a `pack()` eljárásnak a dolga a gomb képernyőre történő helyezése, egészen pontosan az alkalmazásunk ablakába, lehetőség szerint a bal oldalra igazítva.

Számológép

```
1. from Tkinter import *
2.
3. class Calculator(Frame):
4.   def __init__(self):
5.   Frame.__init__(self)
6.   self.pack(expand=YES, fill=BOTH)
7.   self.master.title('SzÆmol gØp')
8.   self.error = 0
9.   self.ans = 0
10.
11.  display = StringVar()
12.  disp = Entry(self, relief=SUNKEN,
           textvariable=display)
13.  disp.pack(side=TOP, expand=YES,
     fill=BOTH)
14.
15.  for key in ("123", "456", "789", "-0."):
16.    fkey = frame(self, TOP)
17.    for char in key:
18.      button(fkey, char, lambda w=display,
         c=char, s=self: s.setscreen(w, c,
         1))
19.
20.  fops = frame(self, TOP)
21.  for char in "+-*/=":
22.    if char == '=':
23.      btn = button(fops, char)
24.      btn.bind('<ButtonRelease-1>',
25.        lambda e, s=self, w=display:
          s.calc(w))
26.    else:
27.      btn = button(fops, char,
28.        lambda w=display,
          c=' %s '%char, s=self:
          s.setscreen(w, c, 0))
29.
30.  clearF = frame(self, BOTTOM)
31.  button(clearF, 'T rlØs', lambda
     w=display, s=self: s.clear(w))
32.
33.  def setscreen(self, w, c, clear):
34.    if self.error == 0:
35.      if self.ans == 1:
36.        self.ans = 0
37.        if clear:
38.          w.set('')
39.      w.set(w.get() + c)
40.    else:
41.      self.clear(w)
42.      w.set(c)
43.
44.  def clear(self, w):
45.    w.set('')
46.    self.error = 0
47.
48.  def calc(self, display):
49.    try:
50.      if self.error == 0:
51.        display.set('eval(display.get())')
52.        self.ans = 1
53.      else:
54.        display.set("kattints a torlesre")
55.    except:
56.      display.set("HIBA")
57.      self.error = 1;
58.
59. def frame(root, side):
60.  w = Frame(root)
61.  w.pack(side=side, expand=YES, fill=BOTH)
62.  return w
63.
64. def button(root, text, command=None):
65.  w = Button(root, text=text,
         command=command)
66.  w.pack(side=LEFT, expand=YES, fill=BOTH)
67.  return w
68.
69. if __name__ == '__main__':
70.  Calculator().mainloop()
```

Egy felületkezelőnek ennél azonban jóval több dologra kell ügyelnie. Kezdetben az ablak mérete pontosan akkora kell legyen, hogy minden elem, amelyet belepakoltunk, elférjen benne és látszódjék, hacsak másképpen nem rendelkeztünk. Amennyiben az ablakunk méretét növeljük, gombjainknak több hely áll rendelkezésére, mint amennyire szükségünk van, és ilyenkor ugyancsak a felületkezelő feladata, hogy megmondja, miképpen változzon meg az ablakban található gomb vagy bármilyen más elem mérete és elhelyezkedése, hogy a felhasználók igényeinek a legjobban megfeleljen. Ebben az esetben feltételezzük, hogyha a felhasználó növeli az ablak méretét, bizonyára nagyobb gombokra van szüksége, így gombunk tulajdonságai közé vesszük, hogy a rendelkezésre álló helyet mindkét irányban töltse ki (`fill=BOTH`), és nőjön együtt az ablakkal (`expand=YES`). A `side` értékkel mondhatjuk meg, hogy a Packer az ablak melyik oldalára igazítsa a gombunkat.

A Packeren kívül két másik felületkezelő is létezik: a Grid és a Placer. A Grid az ablakot rácsokra osztja, és nekünk csak azt kell megadnunk, hogy az egyes elemek mely rácspontokba kerüljenek. Ez a felületkezelő a legjobban talán a HTML-ből ismert TABLE-höz fogható. Használata a Packerhez hasonlóan könnyű és egyszerű, és igazából csak szokás kérdése, hogy melyiket kedveljük jobban – viszont kétségkívül igaz, hogy olyan feladatok is akadnak, amelyek az egyikkel vagy a másikkal könnyebben kivitelezhetők.

A felületkezelők feketebáránya a legegyszerűbb, mégis a legtöbb odafigyelést igénylő ***Placer***, amelynek segítségével elemeinket pontosan igazítva helyezhetjük el az ablakban. Fontos, hogy egy kereten (vagy ablakon) belül csak egyetlen felületkezelő használható! Ez józan ésszel is belátható, hiszen mindannyian az elemek elhelyezéséért felelősek. Tegyük fel, hogy egy ablakot már rácsokra osztottunk, ilyenkor már nem pakolgathatunk benne ide-oda bármit!

Eseménykezelés

Bármilyen rendszer alatt dolgozunk is, legyen az X vagy Windows, a háttérben az egér egyetlen elmozdulását is események egész sora követi. Így van ez mindennel: esemény keletkezik, ha lenyomunk egy gombot, ha kattintunk valahol, sőt még akkor is, ha véletlenül meglökjük az egeret. A futó rendszer dönti el, hogy a pillanatnyilag zajló esemény az alkalmazásunkra tartozik-e vagy teljességgel lényegtelen. Az eseményeknek három fő típusa van: a billentyűzettel, az egérkezeléssel és az ablak helyével, illetve méretével kapcsolatosak. Ezek közül a billentyűzettel összefüggő eseményfajta az egyetlen, amelynek elfogásához az ablakunknak kell az aktív ablaknak, azaz a beviteli fókusszal bíró ablaknak lennie.
Egy eseményt kétféle módon lehet elfogni: vagy közvetlenül a `bind()` eljárással, vagy – amennyiben gombról van szó – a ***command*** tulajdonság megadásával.

```
btn.bind('<ButtonRelease-1>', eseménykezelő)
```

Ebben a példában gombunk objektumához egy eseményt rendelünk. Kikötjük, hogy amennyiben a gombunkon valaki a bal oldali egérgombot nyomja le, hajtsa végre az eseménykezelő által hivatkozott eljárást. Ennek legelső értéke kötelező érvényű, a `bind()` ezen keresztül tudatja az eljárással, hogy milyen esemény történt valójában. Ha függvényünket a ***command*** tulajdonságon keresztül hívatjuk vissza, erre a kötelező értékre nincs szükség.
Az alábbiakban felsorolunk néhány fontosabb eseményazonosítót:

ANY-ENTER	Az egérmutató az elem területére lépett.
BUTTON-1	Az egyes egérgombot lenyomták az objektum területén.
BUTTONRELEASE-2	Az objektum területén a kettes egérgombot felengedték.
KEYPRESS	Lenyomtak egy billentyűt.
CONTROL-SHIFT-F1	Az adott elemen lenyomták az adott billentyűkombinációt.
CONFIGURE	Az ablak mérete vagy helyzete megváltozott.
FOCUSIN	Ablakunk lett az aktív ablak.
DESTROY	Programunk bezárás alatt van.
A	Lenyomott A betű.

Ha gyakran végrehajtódó eseményhez rendelünk eseménykezelőt, ügyeljünk rá, hogy egy nagyobb függvény nagyon lefoglalhatja a processzorunkat, ezért igyekezzük elkerülni.
Az eseménykezelők egy alkalmazásban több szinten is beállíthatók. Alapértelmezésben a beállítás csak egy adott elemre vonatkozik. Ezenkívül még három szint létezik: az alkalmazásszint, amely egy adott alkalmazás minden ablakára és elemére vonatkozik; az osztályszint, ami egy osztály összes kezdeti példányára vonatkozik; illetve a héjszint, amely a szülő alkalmazásablakra vonatkozik.
Egy-egy létrehozott esemény legelőször azon az elemen jelentkezik, amelyen az esemény keletkezett, és attól halad lefelé egészen az alkalmazásszintig, kivéve, ha közben valamelyik szinten úgy rendelkezünk, hogy az eseményt nem engedjük tovább.

Mindez a gyakorlatban

Most, amikor az elmélettel már nagyjából tisztában vagyunk, vessünk néhány pillantást számológépünk forráskódjára.
Ha a kódsorokat begépeljük és .PY kiterjesztéssel mentjük, a `python` paranccsal meghívva fárasztó gépelésünk eredményében már gyönyörködhetünk is.
Amennyiben közelebbi pillantást vetünk a programra, látható, hogy a gerincét a ***Calculator*** osztály alkotja, amelyben a lényeg igazából az `__init__()` eljárásba van sűrítve. Azt már tudjuk, hogy ez az az eljárás, amely objektumpéldányunk létrehozásakor önmagától lefut, ennek megfelelően a benne rejlők már az alkalmazás indulásakor végrehajtódnak. Az első ismeretlenbe a 12. sorban ütközünk, itt hozzuk létre alkalmazásunk kijelzőjét, amely valójában egy egyszerű szövegbeviteli mező, és a következő sorban található `pack()` eljárással tesszük ki a képernyőre. A `relief` értékkel a kijelző stílusát adhatjuk meg, ami ebben az esetben SUNKEN, azaz süllyesztett.
A `textvariable` értékkel azt a változót jelöljük ki, amelyet összekötünk a kijelzővel. Amennyiben a változón módosítunk, változik a kijelző tartalma, ami fordítva is igaznak fog bizonyulni.
A 15. sortól kezdődően alkalmazásunk gombjait rajzoljuk ki, a 17. sorban pedig a ***key*** karaktersorozatot bontjuk szét karakterekre, amelyeket kirajzolásuk után egy lambda kifejezésen keresztül a `setscreen()` eljárásra csatolunk vissza, így ha bármelyiket lenyomjuk, az adott érték a kijelzőn jelenik meg.
A 18. sorban meghívott `button()` eljárás csak hivatkozás egy alább bevezetett függvényre, amelyet nem szabad a ***Button*** osztállyal összekevernünk, utóbbi ugyanis egy objektumpéldánnyal térne vissza.
A 16. és 20. sorokban megint csak egy saját függvényen keresztül hozunk létre egy keretet. Mivel a Packer felületkezelőt használjuk, erre azért van szükség, hogy a Packer egyértelműen eldönthesse, hova kell az adott elemet helyeznie.
A 24. és 25. sorokban a `bind()` függvényt hívjuk meg, amellyel az egérgomblenyomás eseményéhez rendelünk hozzá egy `lambda` eljárást. Az eljárás első értéke (`e`) látszólag használaton kívüli, valójában viszont azért szükséges, mert a `bind()` ezen keresztül küldi el az `event` eseményváltozót.
Az 51. sorban az `eval()` függvényen keresztül a kijelző tartalmát kiértékeljük és az eredményt kiíratjuk.

Összegzés

Ebben a részben egy számológépes példán keresztül a Tkinter alapvető lehetőségeivel ismerkedtünk meg, amelyek felhasználásával egyszerű alkalmazások készítésére leszünk képesek. A példaprogram részeit begépelés után ajánlatos módosítgatni vagy akár bővíteni. Így biztosak lehetünk benne, hogy a frissen elsajátított ismereteket nem felejtjük el azonnal, és később is fel tudjuk használni.

Gludovátz Gábor
(ggabor@sopron.hu)
1996 óta foglalkozik Linux-rendszerekkel. Egyik kedvenc időtöltése a programozás, jelenleg éppen egy C++-ban írt KDE-s játékon dolgozik, de szívesen kódol Pythonban és PHP-ben is. Honlapja ➲ http://www.sopron.hu/~ggabor/

Kapcsolódó címek

A Python hivatalos honlapja ➲ http://www.python.org/
Tkinter- tanfolyam
➲ http://www.pythonware.com/library/tkinter/introduction/