Full Circle

AZ UBUNTU LINUX KÖZÖSSÉG FÜGGETLEN MAGAZINJA

Programozói sorozat – Különkiadás

Programozói sorozat Különkiadás

Programozzunk Pythonban

5. Kötet

A Full Circle magazin különkiadása

Full Circle

AZ UBUNTU LINUX KÖZÖSSÉG FÜGGETLEN MAGAZINJA

Üdvözöllek egy újabb, „egyetlen témáról szóló különkiadásban”

Válaszul az olvasók igényeire, néhány sorozatként megírt cikk tartalmát összegyűjtjük dedikált kiadásokba.

Most ez a **„Programozzunk Pythonban” 27–31. részének** az újabb kiadása (a magazin 53–59. számaiból), semmi extra, csak a tények.

Kérlek, ne feledkezz meg az eredeti kiadási dátumról. A hardver és szoftver jelenlegi verziói eltérhetnek az akkor közöltektől, így ellenőrizd a hardvered és szoftvered verzióit, mielőtt megpróbálod emulálni/utánozni a különkiadásokban lévő ismertetőket. Előfordulhat, hogy a szoftver későbbi verziói vannak meg neked, vagy érhetők el a kiadásod tárolóiban.

Jó szórakozást!

Programozzunk Pythonban - 27. rész

Írta Grag Walters

Ha valaha is vártatok pénztárnál egy mozijegyért, akkor már voltatok legalább egy sorban. Ha már kellett állnotok csúcsforgalomban, akkor ismét csak egy sorban voltatok. Ha már várakoztatok egy hivatalban egy olyan kis jeggyel, amire a 98 van írva és a táblán a 42-es szám volt látható, akkor ott is egy sorban voltatok.

A számítógépek világában a sorok nagyon gyakoriak. Felhasználóként legtöbbször nem is gondolunk rájuk, mi több, fel sem tűnnek. De ha egyszer valósidejű eseményekkel kell majd dolgoznotok, akkor foglalkozni kell majd velük. Mindez csak néhány különböző típusú, feldolgozásra váró adatról szól. Ha egyszer belekerülnek a sorba, akkor onnan csak akkor tűnnek el, ha hozzájuk akarunk férni. A következő adat értékét addig nem használhatjuk, amíg az előtte lévőt ki nem vettük a sorból. Például nem érhetjük el a 15. elemet, csak akkor, ha a többi 14-et is kiolvastuk. Miután lekezeltük, már nem lesz a sor része, és ha elfelejtettük elmenteni, akkor nincs már mód a visszahozására.

Többféle sor is létezik. A leggyakoribb típusok a FIFO (a legelső kivétele legelőször), a LIFO (a legutolsó kivétele legelőször), a Prioritásos és a Gyűrű. A gyűrűkről majd máskor fogunk beszélni.

A FIFO sorok azok, amikkel a hétköznapjainkban is találkozunk. A fentebb felsorolt példák FIFO sorok. A sorban lévő első emberrel fognak legelőször foglalkozni, majd amint az elmegy, mindenki eggyel előrébb léphet. Egy FIFO típusú pufferen nincs semmilyen (az értelmesség határán belüli) korlát a tárolható elemek számát illetően. Egyszerűen csak egymás után vannak rakva. Amint egy elemet feldolgozunk és kihúzunk (vagy eltávolítunk) a sorból, mindenki eggyel előrébb lép.

A LIFO-k már ritkábban fordulnak elő az életben, de még így is vannak példák rájuk. Az egyik, ami most eszembe jut, a konyhaszekrényben lévő edények kupaca. Amikor a edényeket elmostuk és megszárítottuk, azokat berakjuk a szekrénybe. Az utoljára bekerülő lesz az első, amit újra használhatunk. Az összes többinek akár napokat is kell várniuk, hogy sorra kerüljenek. Ugye nem is olyan rossz dolog, hogy a mozijegyvásárlás egy FIFO sor? Mint a FIFO-nál, itt sincs korlátozva az elemszám. A sorban lévő első elemnek mindaddig várnia kell, amíg az újabbakat ki nem olvassuk (azaz a tányérokat levesszük), és ő lesz az utolsó még feldolgozatlan elem.

A prioritásos sorokat már egy kissé nehezebb lehet elképzelni. Gondoljunk egy olyan cégre, amelynek egy nyomtatója van. Mindenki ezt az egy nyomtatót használja. A nyomtatási kérelmeket a részlegek fontossága szerint szolgáljuk ki. A bérszámfejtés (hála égnek) fontosabb, mint mondjuk egy programozó. Mi viszont nagyobb prioritásúak vagyunk (természetesen), mint mondjuk a recepciós. Röviden összefoglalva: a magasabb prioritású adatok a kisebb fontosságúak előtt lesznek lekezelve és kiolvasva.

> Többféle sor is létezik. A leggyakoribb típusok a FIFO (a legelső kivétele legelőször), a LIFO (a legutolsó kivétele legelőször), a Prioritásos és a Gyűrű.

FIFO

A FIFO sorokat egész egyszerű elképzelni az adatok szintjén. A Python listák egy jó elméleti elképzelés alapjai lehetnek. Vegyük például a következőt...

```
[1,2,3,4,5,6,7,8,9,10]
```

10 elemem van a listában. Listaként ezeket index szerint tudjuk elérni. Egy sorban viszont erre nincs lehetőségünk. Mindig a soron következővel kell foglalkozni és a lista sem állandó. Ez NAGYON is dinamikus. Amikor a sor következő elemét kérdezzük le, egyúttal töröljük is azt a sorból. Így a fenti példa szerint egy elem kiolvasásakor visszakapjuk az első

```
import Queue
fifo = Queue.Queue()
for i in range(5):
    fifo.put(i)

while not fifo.empty():
    print fifo.get()
```

elemet (1) és a sor a következőképpen fog kinézni.

```
[2,3,4,5,6,7,8,9,10]
```

Olvassunk ki még kettőt és a 2-est illetve a 3-ast kapjuk vissza, majd a sor a következő lesz.

```
[4,5,6,7,8,9,10]
```

Azt hiszem így már világos a dolog. A Pythonban rendelkezésünkre áll egy egyszerű, meglepő módon Queue-nak nevezett könyvtár a kis és közepes méretű (kb. 500 elemig) sorok előállításához. Fentebb van erre egy rövid példa.

Inicializáljuk a sort (fifo = Queue.Queue()), majd 0-tól 4-ig belehelyezzük a számokat (fifo.put(i)). Ezután használjuk a belső .get() metódust az elemek sorból való kivételére, amíg az üres nem lesz (.empty()). A visszakapott értékek a 0,1,2,3 és 4. A sor által

```
import Queue

fifo = Queue.Queue(12)
for i in range(13):
    if not fifo.full():
        fifo.put(i)

while not fifo.empty():
    print fifo.get()
```

kezelhető maximális elemszámot is meg tudjuk adni, ha a méret értékével inicializálunk a következő módon.

```
fifo = Queue.Queue(300)
```

Miután a maximális elemszám betöltődött, a sor nem engedi további elemek elhelyezését. Ennek az a mellékhatása, hogy a program úgy viselkedik, mintha "holtpontra" jutott volna. Ezt egyszerűen elkerülhetjük, ha használjuk a Queue.full() ellenőrzést. Ebben az esetben a sor 12 elem fogadására van felkészítve. Amikor elkezdjük 0-tól 11-ig egyesével behelyezni az elemeket, ahogy elértük a 12-es számot, a puffer meg fog telni. Mivel a berakás előtt ellenőrizzük, hogy van-e még szabad hely a sorban, ezért az utolsó elem eldobható.

Vannak még más lehetőségek is, de ezeknek más mellékhatásaik vannak, ezért csak egy későbbi cikkben foglalkozunk majd velük. Így a legtöbb esetben vagy korlátozás nélkül használjuk a sort, vagy több helyet hagyunk az elemeknek, mint amennyire előreláthatólag szükségünk lesz.

LIFO

A Queue modul a LIFO sorokat

```
import Queue
lifo = Queue.LifoQueue()
for i in range(5):
    lifo.put(i)
while not lifo.empty():
    print lifo.get()
```

is támogatja. Ismét csak az előbbi listás példát fogjuk alkalmazni. A sor kezdetben a következőképpen néz ki:

```
[1,2,3,4,5,6,7,8,9,10]
```

Három elem kiolvasása után így fog kinézni:

```
[1,2,3,4,5,6,7]
```

Emlékezzünk, hogy a LIFO esetében az utolsónak berakott (Last-in) elem fog legelőször kikerülni (First-out). Itt van a rövid példa LIFO-s változata...

Futtatáskor a "4,3,2,1,0"

```
pq = Queue.PriorityQueue()
pq.put((3,'Medium 1'))
pq.put((4,'Medium 2'))
pq.put((10,'Low'))
pq.put((1,'high'))

while not pq.empty():
    nex = pq.get()
    print nex
    print nex[1]
```

eredményt kapjuk.

Mint a FIFO-nál is, itt is megadhatjuk a sor méretét, illetve használhatjuk a .full() metódust.

PRIORITÁSOS SOROK

Ugyan nem túl gyakran használjuk, de néhány esetben igen

```
(1, 'high')
high
(3, 'Medium')
Medium
(4, 'Medium')
Medium
(10, 'Low')
Low
```

nagy szolgálatot tehetnek a prioritással rendelkező sorok. Teljesen hasonlóan működik mint a többi hasonló adatszerkezet, azzal a különbséggel, hogy egy tuple-t kell átadnunk neki, ami mind a prioritás értéket, mind az adatot tartalmazza. Az előző oldal jobb alsó sarkában van egy rövid példa a Queue könyvtár használatával.

Először inicializáljuk a sort, majd elhelyezzük az elemeket a sorban. Vegyük észre, hogy a (prioritás, adat) elrendezést használjuk. A könyvtár az adatokat prioritás szerinti növekvő sorrendben rendezi el. Amikor kiolvasunk valamit, az ugyanúgy egy tuple-ként jön ki, mint ahogy bekerült. Az adatot index szerint tudjuk elérni. A visszakapott elemek a következők...

Az első két példában egyszerűen csak kiírattuk a sorból kijövő adatokat. Mindez példák erejéig nagyon jó, de a valós életben valószínűleg kezdeni is kéne valamit a kiolvasott adatokkal, mielőtt azok eltűnnének. Amikor a 'print fifo.get' utasítást használjuk, az adatok elküldődnek a terminálra, majd pedig törlődnek. Ezt tartsuk észben.

```
import sys
from Tkinter import *
import ttk
import tkMessageBox
import Queue

class QueueTest:
    def __init__(self,master = None):
        self.DefineVars()
        f = self.BuildWidgets(master)
        self.PlaceWidgets(f)
        self.ShowStatus()
```

Most pedig használjunk néhány, a tkinternél megtanult dolgot egy sorokkal foglalkozó demo program elkészítéséhez. Ennek a demónak két kerete lesz. Az első frame három gombot fog tartalmazni. Egyet-egyet egy FIFO, egy LIFO és egy Prioritásos sornak. A másik keretben egy szövegmező lesz két gombbal (az egyik elemeket helyez el a sorban, a másik pedig elemeket vesz ki onnan), illetve még három címke, amelyek rendre megmutatják, hogy a sor üres vagy teli-e, illetve kiírják az utoljára kiolvasott elemet. Készítünk továbbá még egy kis kódot, ami középre fogja igazítani az ablakot. Fentebb látható a kód eleje.

Itt vannak az importjaink és az osztályunk eleje. Mint eddig is, létrehoztuk az __init__ rutint a DefineVars, BuildWidgets, és PlaceWidgets rutinokkal. Van még

```
    def DefineVars(self):
        self.QueueType = ''
        self.FullStatus = StringVar()
        self.EmptyStatus = StringVar()
        self.Item = StringVar()
        self.Output = StringVar()
        # Define the queues
        self.fifo = Queue.Queue(10)
        self.lifo = Queue.LifoQueue(10)
        self.pq = Queue.PriorityQueue(10)
        self.obj = self.fifo
```

```
def BuildWidgets(self,master):
    # Define our widgets
    frame = Frame(master)
    self.f1 = Frame(frame,
        relief = SUNKEN,
        borderwidth=2,
        width = 300,
        padx = 3,
        pady = 3
    )
    self.btnFifo = Button(self.f1,
        text = "FIFO"
    )
    self.btnFifo.bind('<ButtonRelease-1>',
        lambda e: self.btnMain(1)
    )
    self.btnLifo = Button(self.f1,
        text = "LIFO"
    )
    self.btnLifo.bind('<ButtonRelease-1>',
        lambda e: self.btnMain(2)
    )
    self.btnPriority = Button(self.f1,
        text = "PRIORITY"
    )
    self.btnPriority.bind('<ButtonRelease-1>',
        lambda e: self.btnMain(3)
    )
```

ezen kívül egy ShowStatusnak nevezett metódus is, ami... nos, megmutatja a sor állapotát.

Létrehozzuk a DefineVars rutint. Van négy StringVar() objektumunk, egy üres változó QueueType néven és három különböző sor objektumunk, amikkel játszadozni fogunk. A sorok maximum méretét a demóban 10-re korlátoztuk. Készítettünk egy másik objektumot is obj néven, melyet a FIFO sorhoz rendeltük. Amikor kiválasztjuk a sort valamelyik gombbal, beállítjuk az objektumot a neki megfelelőnek. Így a sor akkor lesz előállítva, amikor másik sorfajtára kapcsolunk át.

Elkezdjük widget-ek definiálását. Elkészítjük az első ablakunkat, a három gombot és beállítjuk a kezelőiket. Érdemes megjegyezni hogy ugyanazt a rutint használjuk a gombok callbackjeinek kezeléséhez. Minden gomb egy értéket küld a callback rutinnak, ami jelezi, hogy melyik gombra kattintottak. Nagyon egyszerűen lehet minden gombhoz dedikált rutint készíteni. Habár, mind a három gombunk ugyanazt csinálja, úgy gondolom jó lenne ha csoportként működnének.

Következő lépésben előkészítjük a következő ablakot, az entry widget-et és két gombot. Egyedül az entry widget bind-ja érdekes. Itt hozzákapcsoljuk a <Return> gombot a self.AddToQueue rutinhoz. Így a felhasználónak nem kell az egeret használnia az adatok hozzáadásakor. Egyszerűen csak beírjuk az adatot az entry widgetbe és megnyomhatjuk a <Return>-t.

Az utolsó három widget definíciója a következő oldal alján látható. Mind a három címke lesz. Beállítjuk a textvariable attribútumot a korábban definiált változókra. Emlékezzünk arra, hogy a változó módosulását automatikusan követni fogja a címke szövege. Pár dolgot a lblData címkén is egy kicsit másként csinálunk. Más fajta betűtípust fogunk használni, hogy jobban kiemeljük a sorból kiolvasott adatot. Ne felejtsük el vissza adni a keret objektumot, hogy használni tudjuk a PlaceWidget rutinban.

A PlaceWidgets rutin eleje a következő oldal közepén látható. Vegyük észre, hogy öt üres címkét raktunk a fő ablak legtetejére. Ezt

```
self.f2 = Frame(frame,
    relief = SUNKEN,
    borderwidth=2,
    width = 300,
    padx = 3,
    pady = 3
)
self.txtAdd = Entry(self.f2,
    width=5,
    textvar=self.Item
)
self.txtAdd.bind('<Return>',self.AddToQueue)
self.btnAdd = Button(self.f2,
    text='Add to Queue',
    padx = 3,
    pady = 3
)
self.btnAdd.bind('<ButtonRelease-1>',self.AddToQueue)
self.btnGet = Button(self.f2,
    text='Get Next Item',
    padx = 3,
    pady = 3
)
self.btnGet.bind('<ButtonRelease-1>',self.GetFromQueue)
```

```
        self.lblEmpty = Label(self.f2,
            textvariable=self.EmptyStatus,
            relief=FLAT
        )
        self.lblFull = Label(self.f2,
            textvariable=self.FullStatus,
            relief=FLAT
        )
        self.lblData = Label(self.f2,
            textvariable=self.Output,
            relief = FLAT,
            font=("Helvetica", 16),
            padx = 5
        )

        return frame
```

az elrendezés miatt csináltam. Ezzel a kis “csalással” sokkal könnyebb az ablak kinézetét kialakítani. Ezután beállíthatjuk az első keretet és egy másik “trükkös” címkét, illetve a három gombot.

Itt helyezzük el a második keretet és egy másik “csaló” címkét, majd a maradék widgeteket.

```
def Quit(self):
    sys.exit()
```

A következő, a saját “alapértelmezett” kilépési rutinunk, ami egyszerűen csak meghívja a sys.exit() függvényt.

A fő gombunk callback rutinja a btnMain. Emlékezzünk, hogy ez megkapja (a p1 paraméteren keresztül) a megnyomott gombot. A self.QueueType változót hivatkozásként használjuk az általunk használt sortípusra, majd a self.obj változót hozzárendeljük a megfelelő sorhoz, végül módosítjuk a főablak címét, hogy megjelenítse az általunk használtat. Ezután kiíratjuk a terminálba a sor típusát (ezt nem feltétlenül fontos megtenni), majd meghívjuk a ShowStatus rutint. Most készítsük el a ShowStatus nevű rutint.

```
    def btnMain(self,p1):
        if p1 == 1:
            self.QueueType = 'FIFO'
            self.obj = self.fifo
            root.title('Queue Tests - FIFO')
        elif p1 == 2:
            self.QueueType = 'LIFO'
            self.obj = self.lifo
            root.title('Queue Tests - LIFO')
        elif p1 == 3:
            self.QueueType = 'PRIORITY'
            self.obj = self.pq
            root.title('Queue Tests - Priority')

        print self.QueueType
        self.ShowStatus()
```

```
        self.f2.grid(column = 0,row = 2,sticky='nsew',columnspan=5,padx = 5, pady = 5)
        l = Label(self.f2,text='',width = 15,anchor = 'e').grid(column = 0, row = 0)
        self.txtAdd.grid(column=1,row=0)
        self.btnAdd.grid(column=2,row=0)
        self.btnGet.grid(column=3,row=0)
        self.lblEmpty.grid(column=2,row=1)
        self.lblFull.grid(column=3,row = 1)
        self.lblData.grid(column = 4,row = 0)
```

```
    def PlaceWidgets(self, master):
        frame = master
        # Place the widgets
        frame.grid(column = 0, row = 0)
        l = Label(frame,text='',relief=FLAT,width = 15, anchor = 'e').grid(column = 0, row = 0)
        l = Label(frame,text='',relief=FLAT,width = 15, anchor = 'e').grid(column = 1, row = 0)
        l = Label(frame,text='',relief=FLAT,width = 15, anchor = 'e').grid(column = 2, row = 0)
        l = Label(frame,text='',relief=FLAT,width = 15, anchor = 'e').grid(column = 3, row = 0)
        l = Label(frame,text='',relief=FLAT,width = 15, anchor = 'e').grid(column = 4, row = 0)

        self.f1.grid(column = 0,row = 1,sticky='nsew',columnspan=5,padx = 5,pady = 5)
        l = Label(self.f1,text='',width = 25,anchor = 'e').grid(column = 0, row = 0)
        self.btnFifo.grid(column = 1,row = 0,padx = 4)
        self.btnLifo.grid(column = 2,row = 0,padx = 4)
        self.btnPriority.grid(column = 3, row = 0, padx = 4)
```

Amint láthatjuk, mindez nagyon egyszerű. Beállítjuk a címke változókat a megfelelő állapotukra, így jelezni tudják, ha az általunk használt sorok beteltek, üresek, vagy valahol a kettő között vannak.

```
if __name__ == '__main__':
  def Center(window):
      # Get the width and height of the screen
      sw = window.winfo_screenwidth()
      sh = window.winfo_screenheight()
      # Get the width and height of the window
      rw = window.winfo_reqwidth()
      rh = window.winfo_reqheight()
      xc = (sw-rw)/2
      yc = (sh-rh)/2
      window.geometry("%dx%d+%d+%d"%(rw,rh,xc,yc))
      window.deiconify()
```

```
  def ShowStatus(self):
      # Check for Empty
      if self.obj.empty() == True:
          self.EmptyStatus.set('Empty')
      else:
          self.EmptyStatus.set('')
      # Check for Full
      if self.obj.full() == True:
          self.FullStatus.set('FULL')
      else:
          self.FullStatus.set('')
```

Az AddToQueue rutin is kimondottan egyértelmű. Az adatokat a .get() függvénnyel kapjuk meg az entry widgetből. Ezután ellenőrizzük, hogy a jelenlegi sortípus prioritásos sor-e. Ha igen, biztosítanunk kell, hogy a megfelelő formátumú legyen. Ezt a vessző meglétének ellenőrzésével tesszük. Ha nincs vessző, ezt jelezzük egy hibaüzenettel a felhasználónak. Ha minden helyesnek tűnik, meg kell bizonyosodnunk arról, hogy a jelenleg használt sor be van-e telve. Emlékezzünk, ha a sor betelik, a put rutin blokkolódik és a program várakozni fog. Ha minden rendben van, akkor hozzáadjuk az elemet a listához és frissítjük a állapotot.

A GetFromQueue rutin még egyszerűbb. Megnézzük, hogy üres-e a sor - így elkerüljük, hogy blokkolódjunk - és ha nem, kiolvassuk az adatot a sorból, megjelenítjük és frissítjük az állapotot.

```
def GetFromQueue(self,p1):
    self.Output.set('')
    if not self.obj.empty():
        temp = self.obj.get()
        self.Output.set("Pulled
{0}".format(temp))
    self.ShowStatus()
```

Mindjárt az alkalmazás végére érünk. Itt van az ablakot középre rakó rutinunk. Először megkapjuk az aktuális ablak szélességét és magasságát. Majd megkapjuk a főablak szélességét és magasságát a tkinter winfo_reqwidth() és winfo_reqheight() rutinjaival. Ha ezeket a megfelelő időben hívjuk meg, akkor visszaadják a főablak szélességét és magasságát a widget pozíciója alapján. Ha túl korán hívod meg, akkor is visszakapunk

```
def AddToQueue(self,p1):
    temp = self.Item.get()
    if self.QueueType == 'PRIORITY':
        commapos = temp.find(',')
        if commapos == -1:
            print "ERROR"
            tkMessageBox.showerror('Queue Demo',
                'Priority entry must be in format\r(priority,data)')
        else:
            self.obj.put(self.Item.get())
    elif not self.obj.full():
        self.obj.put(self.Item.get())
    self.Item.set('')
    self.ShowStatus()
```

valamit, de nem azt, amire valójában szükségünk van. Majd kivonjuk az igényelt ablak szélességét a képernyő szélességéből, és ezt elosztjuk kettővel, illetve ugyanezt tesszük a magassággal is. Végül mindezeket felhasználjuk a geometry meghívásánál. Mindez TÖBBNYIRE csodálatosan működik. De előfordulhat, hogy kézzel kell beállítanunk a kívánt szélességet és magasságot.

```
root = Tk()
root.title('Queue Tests - FIFO')
demo = QueueTest(root)
root.after(3,Center,root)
root.mainloop()
```

Végül példányosítjuk a főablakot, beállítjuk az címsorát, majd példányosítjuk a QueueTest osztályt. Ezután meghívjuk a root.after rutint, amely x ezredmásodpercet vár (ebben az esetben 3-at) miután a főablakot példányosítottuk, és meghívja a Center rutint. A főablak mostmár indulásra készen áll, így már lekérdezhetjük a szélességét és a magasságát. Némi finomhangolás nem árthat a késleltetésnél. Bizonyos gépek sokkal gyorsabbak, mint mások. A 3 jól működik az én gépemen, de a tiéd másmilyen sebességű lehet. Végül, de nem utolsó sorban, meghívjuk a főablak főciklusát az alkalmazás futtatásához.

A programmal való játszadozás közben, figyeljük meg, hogy ha beteszünk néhány adatot egy sorba (mondjuk egy FIFO-ba), majd átváltunk egy másikra (mondjuk LIFO-ra), a FIFO sorba helyezett adatok megmaradnak számodra. Teljesen vagy részben feltölthetjük mindhárom sort és tesztelhetjük a működésüket.

Nos, a mostani alkalomra ennyi. Jó szórakozást a sorokkal! A QueueTest kódja a következő címen található http://pastebin.com/5BBUiDce.

Greg Walters a RainyDay Solutions Kft. tulajdonosa, amely egy tanácsadó cég a coloradói Auróra-ban. Greg 1972 óta foglalkozik programozással. Szeret főzni, túrázni, zenét hallgatni, valamint a szabadidejét családjával tölteni. Weblapja a www.thedesignatedgeek.com címen található meg.

Programozzunk Pythonban – 28. rész

Ez alkalommal még több tkinker felhasználói felület elemet fogunk megismerni, név szerint a menüt, a combo boxot, a spin boxot, a separatort, a progress bart és a notebookot. Lássuk ezeket egy kicsit részletesebben.

Akárhány alkalmazást használtál eddig, jóformán mindben találkozhattál menükkel. Egy menü létrehozásához a tkinter NAGYON egyszerű támogatást kínál. A combo box hasonló a legutóbb bemutatott list boxhoz, azonban a tartalmazott lista nem állandóan látható, hanem úgymond „legördül”. A spin box egy szám kiválasztását teszi lehetővé egy intervallum végigjárásán keresztül. Ha például 1 és 100 között egy számot szeretnénk kiválasztani, érdemes ezt az elemet felhasználni. A progress bar segítségével jelezhetjük, hogy az alkalmazásunk nem fagyott le, csupán a program futása tart hosszú ideig (pl. adatbázis olvasás esetén). Kétféle verziója létezik, a határozott és a határozatlan. Az elsőt akkor használjuk, ha tudjuk, hogy mennyi elem feldogozására kell várni, a másodikat pedig akkor, ha nem lehet kiszámítani, hogy a hosszan tartó művelet hány százaléknál tart. Mind a két típusú elemre fogunk példát látni. Utoljára a notebook (vagy tabbed) a sok elemet tartalmazó – például konfigurációs – felületeken hasznos; „füleket” definiálhatunk, amik alá aztán egy halom más elemet tehetünk be.

Kezdjünk is neki: Ahogy korábban is, most is egy alap alkalmazásból indulunk ki, amihez egyre újabb és újabb elemeket adunk hozzá. A következő kód már ismerős kell, hogy legyen:

Mentsük el widgetdemo2a.py néven. Ez lesz az alapja a teljes demonak. Kezdjük a munkát a menü elkészítésével. A következő lépéseken kell végigmenni.

Először is hozzunk létre egy változót, ami a menüt reprezentálja. A többi elemhez hasonlóan a formátum a következő:

```
OurVariable = Widget(parent, options).
```

```
import sys
from Tkinter import *
import ttk
# Shows how to create a menu
class WidgetDemo2:

    def __init__(self,master = None):
        self.DefineVars()
        f = self.BuildWidgets(master)
        self.PlaceWidgets(f)

    def DefineVars(self):
        pass
```

A program második része pedig szintén nem tartalmaz újdonságot:

```
if __name__ == '__main__':
    def Center(window):
        # Get the width and height of the screen
        sw = window.winfo_screenwidth()
        sh = window.winfo_screenheight()
        # Get the width and height of the window
        rw = window.winfo_reqwidth()
        rh = window.winfo_reqheight()
        xc = (sw-rw)/2
        yc = (sh-rh)/2
        print "{0}x{1}".format(rw,rh)
        window.geometry("%dx%d+%d+%d"%(rw,rh,xc,yc))
        window.deiconify()

    root = Tk()
    root.title('More Widgets Demo')
    demo = WidgetDemo2(root)
    root.after(13,Center,root)
    root.mainloop()
```

Menü esetén a parent paraméter értéke a master lesz. Az értékadás a BuildWidgets függvényben történik. Következőnek létrehozunk egy másik, filemenu nevű menü elemet, majd hozzáadjuk a parancsokat és elválasztókat, végül pedig hozzáadjuk a menühöz. A példánkban egy menubar van File, Edit és Help elemekkel. Fogjunk hozzá:

Folytassuk a File menü elkészítésével, amiben öt elemet definiálunk: New, Open, Save, egy elválasztó, és Exit. Az elemek létrehozásához a .add_command() metódust használjuk. Annyit kell tennünk, hogy meghívjuk a metódust a címkével (label =), aztán pedig egy ún. callback függvényt kell megadunk, ami az elemre való kattintáskor hajtódik végre. Utolsó lépésként a menubar.add_cascade() függvénnyel beillesztjük az új menüelemet a többi mellé.

Mivel az exit parancs a „root.quit” parancs segítségével zárja be az alkalmazást, ezért nincs szükség külön callback függvény definiálására. Az Edit és Help menüpontokat a következő programrészlettel lehet létrehozni.

Figyeljük meg a menücsoportok definiálásánál a „tearoff=0” parancsot. Ha az értéket átírjuk 1-re, akkor a menü szaggatott vonallal jelenne meg, az egérrel azt „leszakítva” egy új ablak jelenne meg. Habár ez is egy hasznos funkció, most nem szeretnénk ezt felhasználni.

Utoljára következzen a menü elhelyezése. Ne használjuk a .grid() függvény által kínált elrendező függvényt, helyette egyszerűen hívjuk meg a parent.config függvényt.

Mindez kerüljön a BuildWidgets függvénybe. Most már csak egy általános ablak és egy visszatérési érték beállítás kell, mielőtt a PlaceWidgets függvényre térnénk.

Végül definiáljuk a callback függvényeket. A példa kedvéért ezek a függvények mindössze egy-egy szöveget írnak ki az indításhoz használt terminálba:

```
def BuildWidgets(self,master):
    frame = Frame(master)
    #==============================
    #       MENU STUFF
    #==============================
    # Create the menu bar
    self.menubar = Menu(master)
```

```
# Create the File Pull Down, and add it to the menu bar
  filemenu = Menu(self.menubar, tearoff = 0)
  filemenu.add_command(label = "New", command = self.FileNew)
  filemenu.add_command(label = "Open", command = self.FileOpen)
  filemenu.add_command(label = "Save", command = self.FileSave)
  filemenu.add_separator()
  filemenu.add_command(label = "Exit", command = root.quit)
  self.menubar.add_cascade(label = "File", menu = filemenu)
```

```
# Create the Edit Pull Down
editmenu = Menu(self.menubar, tearoff = 0)
editmenu.add_command(label = "Cut", command = self.EditCut)
editmenu.add_command(label = "Copy", command = self.EditCopy)
editmenu.add_command(label = "Paste", command = self.EditPaste)
self.menubar.add_cascade(label = "Edit", menu = editmenu)
# Create the Help Pull Down
helpmenu = Menu(self.menubar, tearoff=0)
helpmenu.add_command(label = "About", command = self.HelpAbout)
self.menubar.add_cascade(label = "Help", menu = helpmenu)
```

```
  # Now, display the menu
master.config(menu = self.menubar)
  #=========================================
  #           End of Menu Stuff
  #=========================================
```

Kész is vagyunk! Mentsük el és indítsuk el a programot. Próbáljuk ki a menüelemeket.

Adjunk hozzá egy combo boxot. Mentsük el a fájlt widgetdemo2b.py néven, és folytassuk a munkát. Az import rész, az osztálydefiníció és az __init__ függvény ugyanaz marad, egyedül a DefineVars függvényhez adunk két új sort. Vagy kommenteljük ki a „pass” utasítást, vagy pedig töröljük ki és írjuk be a következőt:

Először egy labelt hozunk létre – amit már korábban is csináltunk, – majd egy combo boxot definiálunk. A szülőobjektum a „ttk.Combobox”, a magasság 19, a szélesség 20, a textvariable értéke pedig „self.cmbo1Val” legyen. A textvariable-t az előző demoban használtuk, ha valaki nem emlékezne. Ez az érték a combo box mindenkori értékét tartalmazza, definíciója a DefineVars metódusban található, típusa pedig StringVar. Ezután betöltjük a combo box lehetséges értékeit (amit szintén a DefineVars-ban adtunk meg), majd bekötjük a <<ComboboxSelected>> virtuális eseményt a cm-

```
        self.f1 = Frame(frame,
                        relief = SUNKEN,
                        borderwidth = 2,
                        width = 500,
                        height = 100
                        )

        return frame
```

Most – mint már előzőleg többször is – a következő módon helyezzük el az elemeket:

```
    def PlaceWidgets(self,master):
        frame = master
        frame.grid(column = 0, row = 0)

        self.f1.grid(column = 0,
                     row = 0,
                     sticky = 'nsew'
                     )
```

```
    def DefineVars(self):
        self.cmbo1Val = StringVar()
        self.c1Vals = ['None','Option 1','Option 2','Option 3']
```

A WindowBuilder függvényben a self.f1 definiálása és a „return frame” sor közé írjuk be a következőt:

```
        # Combo Box
        self.lblcb = Label(self.f1, text = "Combo Box: ")
        self.cmbo1 = ttk.Combobox(self.f1,
                                  height = "19",
                                  width = 20,
                                  textvariable = self.cmbo1Val
                                 )
        self.cmbo1['values'] = self.c1Vals
        # Bind the virtual event to the callback
        self.cmbo1.bind("<<ComboboxSelected>>",self.cmbotest)
```

```
    def FileNew(self):
        print "Menu - File New"

    def FileOpen(self):
        print "Menu - File Open"

    def FileSave(self):
        print "Menu - File Save"

    def EditCut(self):
        print "Menu - Edit Cut"

    def EditCopy(self):
        print "Menu - Edit Copy"

    def EditPaste(self):
        print "Menu - Edit Paste"

    def HelpAbout(self):
        print "Menu - Help About"
```

botest függvénybe, amit mindjárt megadunk.

Illesszük be a combo boxot és a labelt a felületre.

Mentsük el és próbáljuk ki az alkalmazást.

Mentsük el a progit widgetdemo2c.py néven, ami a separator bar működését fogja szemléltetni. Bár a legfrissebb tkinter csomag tartalmazza ezt az elemet, nekem még nem sikerült működésre bírnom. Alternatív megoldás, hogy létrehozunk egy 2 magas keretet a BuildWidgets függvényben a combo box „bind” utasítása után:

Ez a kód újfent ismerős kell, hogy legyen, szóval mentsük el és próbáljuk ki. A legfelső ablakot lehet hogy nagyobbra kell venni, hogy látszódjon az elválasztó, de a következő példában sokkal könnyebben észrevehető lesz.

Adjuk hozzá a DefineVars metódushoz a következőt:

```
self.spinval = StringVar()
```

Ezt az értéket bárhonnan elérhetjük. Most adjuk hozzé a BuildWidgets függvényhez a következőket a „return frame” utasítás elé:

Itt egy labelt és egy spin elemet definiálunk. Ez utóbbi a következő paramétereket használja:

```
elem = Spinbox(szülö,minimum, maximum, szélesség, szöveg, átfordulás)
```

A „from” paraméter azért „from_” mert az eredeti egy nyelvi kulcsszó, így nem használható paraméternévként. A from és to paraméter float típusú, a mi esetünkben értékük 1 és 10. A wrap paraméterrel beállítható, hogy ha az elem a maximumon van, akkor növeléskor a minimumértéktől újrakezdődik a számolás, false esetén azonban megáll a szélsőértéknél.

Adjuk hozzá az elemeket a PlaceWidgets függvényben.

És megint készen vagyunk, mentsük és próbáljuk ki. Most már tényleg látható a separator. Mentsük el a programot widgetdemo2e.py néven, majd folytassuk a progress barral.

Megint szükségünk van a néhány változóra, adjuk hozzá a Defi-

```
        self.lblcb.grid(column = 0,row = 2)
        self.cmbo1.grid(column = 1,
                        row = 2,
                        columnspan = 4,
                        pady = 2
                        )
```

Adjuk meg a callback függvényt, ami a terminálra írja a felhasználó által választott értéket.

```
    def cmbotest(self,p1):
        print self.cmbo1Val.get()
```

```
        self.fsep = Frame(self.f1,
                          width = 140,
                          height = 2,
                          relief = RIDGE,
                          borderwidth = 2
                          )
```

A PlaceWidgets metódusba pedig a következő kód kerüljön:

```
        self.fsep.grid(column = 0,
                       row = 3,
                       columnspan = 8,
                       sticky = 'we',
                       padx = 3,
                       pady = 3
                       )
```

```
    self.lblsc = Label(self.f1, text = "Spin Control:")
    self.spin1 = Spinbox(self.f1,
                         from_ = 1.0,
                         to = 10.0,
                         width = 3,
                         textvariable = self.spinval,
                         wrap=True
                         )
```

neVars függvényhez a következő kódot.

```
self.spinval2 = StringVar()
self.btnStatus = False
self.pbar2val = StringVar()
```

A változók nevei magukért beszélnek. A „self.btnStatus”-al később foglalkozunk, haladjunk inkább az elemek definiálásával a BuildWidgets függvényben (jobbra).

A kód ismét a „return frame” utasítás elé kell, hogy kerüljön. A kód egy keretet hoz létre, ahova az elemeinket pakolhatjuk, majd berak két jelzőként szolgáló labelt. Ezután következik a progress bar. A paraméterek közül a „length”, a „mode”, és a „maximum” szorul további magyarázatra. A length a progress bar szélessége pixelekben megadva. A maximum a megjelenített folyamat maximuma, ami jelen esetben 100, mivel százalékos értéket szeretnénk megjeleníteni. A mode értéke most 'indeterminate', ami azt fejezi ki, hogy nem tudjuk, hogy a folyamatunk hol tart, csak azt, hogy valami történik.

Ezután hozzáadunk egy buttont, egy labelt, egy másik progress bart, és egy új spin elemet. A második progress bar 'determinate' típusú, a folyamat előrehaladását a spinnel fogjuk beállítani. Adjuk hozzá a PlaceWidgets függvényhez a következő sorokat.

A progress bar vezérléséhez hozzuk létre az alábbi két függvényt.

A TestPBar függvény a határozott progress bart irányítja, ami a progress barba épített számlálót indítja el és állítja meg. A „self.pbar.start(10)” a számlálót 10 msec-re állítja, amitől a progress bar igen gyorsan végigmegy. Próbáljunk ki más időérték beállításokat is megadni. A Spin2Do függvény a progress barnak tetszőleges értéket állít be, amit ki

```
        self.lblsc.grid(column = 0, row = 4)
        self.spin1.grid(column = 1,
                        row = 4,
                        pady = 2
                        )
```

```
        #=======================================
        # Progress Bar Stuff
        #=======================================
        self.frmPBar = Frame(self.f1,
                             relief = SUNKEN,
                             borderwidth = 2
                             )

        self.lbl0 = Label(self.frmPBar,
                             text = "Progress Bars"
                             )
        self.lbl1 = Label(self.frmPBar,
                             text = "Indeterminate",
                             anchor = 'e'
                             )
        self.pbar = ttk.Progressbar(self.frmPBar,
                             orient = HORIZONTAL,
                             length = 100,
                             mode = 'indeterminate',
                             maximum = 100
                             )
        self.btnptest = Button(self.frmPBar,
                             text = "Start",
                             command = self.TestPBar
                             )
        self.lbl2 = Label(self.frmPBar,
                             text = "Determinate"
                             )
        self.pbar2 = ttk.Progressbar(self.frmPBar,
                             orient = HORIZONTAL,
                             length = 100,
                             mode = 'determinate',
                             variable = self.pbar2val
                             )
        self.spin2 = Spinbox(self.frmPBar,
                             from_ = 1.0,
                             to = 100.0,
                             textvariable = self.spinval2,
                             wrap = True,
                             width = 5,
                             command = self.Spin2Do
                             )
```

is ír a terminálba.

Készen vagyunk, mentsük el és próbáljuk ki a programunkat.

Mentsük el a programot widget-demo2f.py néven, hogy hozzáadhassuk a tabbed notebook elemet is. A BuildWidgets függvénybe a „return frame” elé a következő kód kerüljön.

Nézzük, mit is csinál mindez. Először definiálunk egy keretet a notebooknak, majd hozzáadjuk azt. A beállításokat már mind átnéztük korábban. Létrehozunk két keretet, amik a füleket reprezentálják. A következő két sor (self.notebook.add) hozzáadja a kereteket a notebookhoz, amik így fülekké válnak. A fülekhez szöveget rendelünk, végül pedig az első oldalra felrakunk egy labelt, csak hogy legyen rajta valami.

A PlaceWidgets függvlénybe illesszük be a következő kódot.

Az egyetlen furcsa dolog a label lehet a második oldalon, mivel

```
        #=======================================
        #               NOTEBOOK
        #=======================================
        self.nframe = Frame(self.f1,
                            relief = SUNKEN,
                            borderwidth = 2,
                            width = 500,
                            height = 300
                            )
        self.notebook = ttk.Notebook(self.nframe,
                                     width = 490,
                                     height = 290
                                     )
        self.p1 = Frame(self.notebook)
        self.p2 = Frame(self.notebook)
        self.notebook.add(self.p1,text = 'Page One')
        self.notebook.add(self.p2,text = 'Page Two')
        self.lsp1 = Label(self.p1,
                          text = "This is a label on
page number 1",
                          padx = 3,
                          pady = 3
                          )
```

```
        # Progress Bar
        self.frmPBar.grid(column = 0,
                          row = 5,
                          columnspan = 8,
                          sticky = 'nsew',
                          padx = 3,
                          pady = 3
                          )
        self.lbl0.grid(column = 0, row = 0)
        self.lbl1.grid(column = 0,
                       row = 1,
                       pady = 3
                       )
        self.pbar.grid(column = 1, row = 1)
        self.btnptest.grid(column = 3, row = 1)
        self.lbl2.grid(column = 0,
                       row = 2,
                       pady = 3
                       )
        self.pbar2.grid(column = 1, row = 2)
        self.spin2.grid(column = 3, row = 2)
```

```
    def TestPBar(self):
        if self.btnStatus == False:
            self.btnptest.config(text="Stop")
            self.btnStatus = True
            self.pbar.start(10)
        else:
            self.btnptest.config(text="Start")
            self.btnStatus = False
            self.pbar.stop()

    def Spin2Do(self):
        v = self.spinval2.get()
        print v
        self.pbar2val.set(v)
```

ugynazzal az utasítással hozzuk létre az elemet és határozzuk meg annak helyét.

Kész. Mentsünk, próbáljuk ki.

Mint mindig, a teljes forráskód elérhető a pastebin weboldalon: http://pastebin.com/qSPkSNU1

Használjátok egészséggel; legközelebb még több adatbázis programozással jelentkezem.

```
self.nframe.grid(column = 0,
                 row = 6,
                 columnspan = 8,
                 rowspan = 7,
                 sticky = 'nsew'
                 )
self.notebook.grid(column = 0,
                   row = 0,
                   columnspan = 11,
                   sticky = 'nsew'
                   )
self.lsp1.grid(column = 0,row = 0)
self.lsp2 = Label(self.p2,
                  text = 'This is a label on PAGE 2',
                  padx = 3,
                  pady = 3
                  ).grid(
                         column = 0,
                         row = 1
                         )
```

Programozzunk Pythonban – 29. rész

Nemrég megkértek arra, hogy oldjam meg egy MySQL adatbázis SQLite-ra való konverzióját. A weben egy gyors és egyszerű (illetve ingyenes) megoldást keresgélve, nem találtam olyan módszert, ami együtt tudott volna működni a legfrissebb MySQL verzióval. Ezért egy saját módszer kidolgozása mellett döntöttem.

A MySQL Administrator program segítségével lehetőségünk van adatbázisok egyszerű szöveges fájlba való mentésére. Sok SQLite nézegető pedig be tud olvasni sima sql parancsfájlokat és újra létre tudja hozni belőlük az adatbázist. A gond csak az, hogy a MySQL-nek sok olyan szolgáltatása van, amit az SQLite nem támogat. Tehát, ebben a hónapban egy olyan átalakító programot fogunk készíteni, ami MySQL dumpok SQLite változatát készíti el.

Kezdésként vizsgáljuk meg a MySQL dump formátumát. Az egész egy olyan résszel kezdődik, ami létrehozza az adatbázist, majd az ezt követő szekciók leírják a benne lévő táblákat és azok tartalmát abban az esetben, ha azok a dump fájlban megtalálhatóak. (Lehetőségünk van csak sémák exportálására). Jobbra egy példa, tábla létrehozására:

Először meg kellene válnunk a végén lévő sortól. Az utolsó zárójel után mindent törölhetünk. (Az SQLite nem támogatja az InnoDB adatbázist). Ezen felül, az SQLite a „PRIMARY KEY” sort sem tudja értelmezni. SQLite-ban elsődleges kulcsot az „INTEGER PRIMARY KEY AUTOINCREMENT” segítségével állíthatunk be egy mezőre. A következő dolog, ami ismét csak nem működik, az az „unsigned” kulcsszó.

Az adatok esetében az „INSERT INTO” utasítások sem lesznek kompatibilisek. Itt a probléma abban ledzik, hogy az SQLite nem enged meg több beszúrást egyetlen utasításban. Itt jobbra van egy rövid példa a MySQL dumpból. Vegyük észre a sorvéget jelző pontosvesszőt:

Ezen felül figyelmen kívül fogunk hagyni minden megjegyzést, CREATE DATABASE és USE utasítást. Miután készen vagyunk az át-

```
DROP TABLE IF EXISTS `categoriesmain`;
CREATE TABLE `categoriesmain` (
  `idCategoriesMain` int(10) unsigned NOT NULL
auto_increment,
  `CatText` char(100) NOT NULL default '',
  PRIMARY KEY  (`idCategoriesMain`)
) ENGINE=InnoDB AUTO_INCREMENT=40 DEFAULT
CHARSET=latin1;
```

```
INSERT INTO `categoriesmain`
(`idCategoriesMain`,`CatText`) VALUES
 (1,'Appetizer'),
 (2,'Snack'),
 (3,'Barbecue'),
 (4,'Cake'),
 (5,'Candy'),
 (6,'Beverages');
```

Ahhoz, hogy ez működjön, az utasításokat fel kell darabolni több utasítás sorozatra. Például így...

```
INSERT INTO `categoriesmain`
(`idCategoriesMain`,`CatText`) VALUES (1,'Appetizer');
INSERT INTO `categoriesmain`
(`idCategoriesMain`,`CatText`) VALUES (2,'Snack');
INSERT INTO `categoriesmain`
(`idCategoriesMain`,`CatText`) VALUES (3,'Barbecue');
INSERT INTO `categoriesmain`
(`idCategoriesMain`,`CatText`) VALUES (4,'Cake');
INSERT INTO `categoriesmain`
(`idCategoriesMain`,`CatText`) VALUES (5,'Candy');
INSERT INTO `categoriesmain`
(`idCategoriesMain`,`CatText`) VALUES (6,'Beverages');
```

alakított SQL fájllal, az a szabadon használható SQLite Database Browserhez hasonló programok segítségével létrehozza az adatbázist, illetve a táblákat és az általuk tárolt adatokat.

Vágjunk is bele. Készítsünk egy új mappát, majd egy python fájlt a projektünknek. Nevezzük MySQL2SQLite.py-nak.

Jobbra van az import utasítás, a class definíció és az __init__ rutin.

Ez egy parancssori alkalmazás lesz, ezért létre kell hoznunk az „if __name__” utasítást, a parancssori argumentumok kezelőjét, illetve egy használati utasítást kiíró rutint (ha a felhasználó nem ismerné a programot). Mindez a program legvégére fog kerülni. Minden más kódot a következő sor fölé kell elhelyezni:

```
def error(message):

    print >> sys.stderr,
str(message)
```

A használati utasítás kiírásáért felelős rész jobbra lent található:

A DoIt() rutin akkor hívódik meg, amikor a programot a parancssorból futtatják (ez lenne az elsődleges használati módja). Ha azonban egy olyan library-ként szeretnénk használni, amit más programokba is be lehet építeni, akkor csak az osztályra van szükségünk. Itt több olyan változót is be kell állítani, amelyek a helyes működéshez elengedhetetlenek. A következő kódrészlet a kapott parancssori argumentumok értelmezéséért felelős, és felkészül a fontosabb metódusok hívására.

```
def DoIt():
    #=====================================
    #            Setup Variables
    #=====================================
    SourceFile = ''
    OutputFile = ''
    Debug = False
    Help = False
    SchemaOnly = False
    #=====================================
```

```
    #!/usr/bin/env python
    #=====================================
    # MySQL2SQLite.py
    #=====================================
    #            IMPORTS
    import sys
    #=====================================

    #=====================================
    #     BEGIN CLASS MySQL2SQLite
    #=====================================
    class MySQL2SQLite:
        def __init__(self):
            self.InputFile = ""
            self.OutputFile = ""
            self.WriteFile = 0
            self.DebugMode = 0
            self.SchemaOnly = 0
            self.DirectMode = False
```

```
if len(sys.argv) == 1:
    usage()
else:
    for a in sys.argv:
        print a
        if a.startswith("Infile="):
            pos = a.find("=")
            SourceFile = a[pos+1:]
        elif a.startswith("Outfile="):
            pos = a.find("=")
            OutputFile = a[pos+1:]
        elif a == 'Debug':
            Debug = True
        elif a == 'SchemaOnly':
            SchemaOnly = True
        elif a == '-Help' or a == '-H' or a == '-?':
            Help = True
    if Help == True:
        usage()
    r = MySQL2SQLite()
    r.SetUp(SourceFile,OutputFile,Debug,SchemaOnly)
    r.DoWork()
```

Amikor elindítjuk a programot, legalább két változót meg kell adnunk a parancssorban. Ezek az Input és az Output fájlok. Továbbá lehetőséget biztosítunk arra is, hogy a felhasználó nyomon tudja követni a program futását, illetve táblákat adatok nélkül tudjon kimenteni és meg tudjon nézni egy rövid leírást a program használatáról. A „normális” parancssori utasítás így fog kinézni:

```
MySQL2SQLite Infile=Foo
Outfile=Bar
```

ahol a „Foo” a MySQL dump fájl, a „Bar” pedig a létrehozandó SQLite sql fájl.

Emellett a következőképpen is meghívhatjuk:

```
MySQL2SQLite Infile=Foo
Outfile=Bar Debug SchemaOnly
```

Így a debug üzenetek is ki fognak íródni, és KIZÁRÓLAG csak táblák létrehozására kerül majd sor, adatok importálása nélkül.

Végül, ha a felhasználónak segítségre lenne szüksége, akkor a használati utasításokra térünk rá.

Mielőtt továbblépnénk, vizsgáljuk meg, hogy hogyan működik a parancssori argumentumok használata.

```
def usage():
    message = (
       '================================================================\n'
        'MySQL2SQLite - A database converter\n'
        'Author: Greg Walters\n'
        'USAGE:\n'
        'MySQL2SQLite Infile=filename [Outfile=filename] [SchemaOnly] [Debug] [-H-Help-?\n'

        '   where\n'
        '         Infile is the MySQL dump file\n'
        '         Outfile (optional) is the output filename\n'
        '            (if Outfile is omitted, assumed direct to SQLite\n'
        '         SchemaOnly (optional) Create Tables, DO NOT IMPORT DATA\n'
        '         Debug (optional) - Turn on debugging messages\n'
        '         -H or -Help or -? - Show this message\n'
        'Copyright (C) 2011 by G.D. Walters\n'
        '================================================================\n'
       )
    error(message)
    sys.exit(1)

if __name__ == "__main__":
    DoIt()
```

Amikor a felhasználó beírja a program nevét (a terminálba), az operációs rendszer megjegyzi és továbbítja a megadott információkat arra az esetre, ha paraméterek is lennének megadva. Ha nincsenek ilyen paraméterek (avagy argumentumok), akkor ezek száma 1, ami az alkalmazás nevét takarja - a mi esetünkben MySQL2SQLite.py. Ezeket az argumentumokat a sys.arg utasítással érhetjük el. Ha a számuk több egynél, akkor egy ciklus segítségével tudjuk elérni őket. Az argumentumok listájának mindegyik elemén végiglépdelve leellenőrizhetjük őket. Némely program megszabja, hogy milyen sorrendben kell megadni őket. A ciklusos megoldás használatával az argumentumok tetszőle- ges sorrendben megadhatóak lesznek. Ha a felhasználó egyetlen paramétert sem ad meg, vagy a helpet használja, akkor megjelenítjük a használati utasítást. Fent látható a rutinja.

Ha mindezzel és az argumentumok értelmezésével is megvolnánk, példányosítjuk az osztályt, majd meghívjuk a setup rutint, ami kitölt néhány változót és végül elindítja a DoWork rutint. Lássunk is hozzá az osztályhoz.

Itt bevezetjük az osztályt és az __init__ rutint. Létrehozzuk azokat a változókat, amikre a későbbiekben szükségünk lesz. Ne felejtsük el, hogy a DoWork meghívása előtt meg kell hívnunk a Setup rutint. Hozzárendeljük az üres változóinkhoz a helyes értékeket. Vegyük észre, hogy lehetőségünk van a

fájlba írás kihagyására is, ami debuggoláskor jól jöhet. Ezen kívül csak a séma, azaz kizárólag az adatbázis struktúrájának kiírására is ki lehetőségünk van. Ez akkor lehet hasznos, ha egy olyan új projekthez kezdünk hozzá, ahol csak az adatbázisra van szükségünk, adatok nélkül.

A munkát az SQL íDump fájl megnyitásával kezdjük, majd néhány belső változót állítunk be. Ezen felül néhány olyan karakterláncot is megadunk, amikkel némi gépelést fogunk megspórolni magunknak. Ha írunk fájlba, akkor megnyitjuk azt és elkezdjük a teljes folyamatot. Az input minden sorát beolvassuk, feldolgozzuk és nagy valószínűséggel ki is írjuk. Egy végtelen while ciklussal oldjuk meg a sorok beolvasását, és egy break utasítást használunk, ha már nincs más a fájlban. A beolvasáshoz az f.readline() kell, aminek az eredményét a „line” változóba helyezzük. Néhány sort minden további nélkül figyelmen kívül hagyhatunk. Ehhez egy if/elif és egy pass utasítás kell csak.

Ezután végre csinálunk valami komolyabbat is. Ha egy CreateTable utasításunk van,

```
    def SetUp(self, In, Out = '', Debug = False, Schema = 0):
        self.InputFile = In
        if Out == '':
            self.writeFile = 0
        else:
            self.WriteFile = 1
            self.OutputFile = Out
        if Debug == True:
            self.DebugMode = 1
        if Schema == 1:
            self.SchemaOnly = 1
```

Most pedig azzal a DoWork rutinnal fogunk foglalkozni, ami a „lényeges” dolgok végrehajtásáért felelős.

```
    def DoWork(self):
        f = open(self.InputFile)
        print "Starting Process"
        cntr = 0
        insertmode = 0
        CreateTableMode = 0
        InsertStart = "INSERT INTO "
        AI = "auto_increment"
        PK = "PRIMARY KEY "
        IPK = " INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL"
        CT = "CREATE TABLE "
        # Begin
        if self.WriteFile == 1:
            OutFile = open(self.OutputFile,'w')
```

```
    while 1:
        line = f.readline()
        cntr += 1
        if not line:
            break
        # Ignore blank lines, lines that start with
"--" or comments (/*!)
        if line.startswith("--"): #Comments
            pass
        elif len(line) == 1: # Blank Lines
            pass
        elif line.startswith("/*!"): # Comments
            pass
        elif line.startswith("USE"):
            #Ignore USE lines
            pass
        elif line.startswith("CREATE DATABASE "):
            pass
```

```
#===================================
#     BEGIN CLASS MySQL2SQLite
#===================================
class MySQL2SQLite:
    def __init__(self):
        self.InputFile = ""
        self.OutputFile = ""
        self.WriteFile = 0
        self.DebugMode = 0
        self.SchemaOnly = 0
```

akkor elindítjuk a folyamatot. Emlékezzünk, hogy a CT-ét a „Create Table”-vel tettük egyenlővé. Itt beállítjuk a „CreateTableMode”-ot „1”-re, hogy tudjuk mit is kell csinálni. Ezután fogjuk a sorunkat, kivesszük a sortörést és előkészítjük a fájlba való kiírást, majd ha kell, akkor meg is tesszük.

Most pedig meg kell csinálnunk a „create table”-ben lévő sorokat - minden egyes sort külön manipulálva, hogy az SQLite-nak is jó legyen. Sok olyan dolog van, amit az SQLite nem tud lekezelni. Nézzük meg még egyszer a MySQL Create Table utasítását.

Az egyik olyan dolog amit az SQLite semmiképpen sem tud értelmezni, az az utolsó zárójel utáni teljes sor. Egy másik az eggyel előtte levő „Primary Key”-t tartalmazó sor, illetve a második sorban lévő „unsigned” kulcsszó. Ezek lekezelése bele telik majd némi kódolásba, de sikerülni fog.

Először megnézzük, hogy tartalmaz-e a sor „auto increment”-et. Feltételezni fogjuk, hogy ez lesz az elsődleges kulcs sora. Annak ellenére, hogy ez a feltevés az esetek 98.6%-ban helyes, nem lehetünk mindig biztosak ebben. Mindazonáltal az egyszerűség mellet fogjuk letenni a voksunkat. Ezt követően megnézzük, hogy a következő sor „)”-el kezdődik-e. Ez azt jelentené, hogy elértük a „create table” utolsó sorát. Ebben az esetben megfelelően lezárjuk a „newline”-ban lévő utasítást, majd kikapcsoljuk a CreateTableMode változót és végül kiírjuk fájlba (ha szükséges).

Ezt követően felhasználjuk az „auto increment” kulcsszóról szerzet információinkat. Kezdésként eltüntetjük a felesleges szóközöket, majd megkeressük, hogy hol van (feltételezzük a létezését) az „ int(” kifejezés a sorban. Ezt le fogjuk cserélni az „ INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL” utasításra. Az egész számok hossza nem jelent semmit az SQLite számára. Ezt ismét kiírjuk, ha szükség van rá.

Most a „PRIMARY KEY “ kifejezést kell meg- találnunk a sorban. Vegyük

```
    elif line.startswith(CT):
        CreateTableMode = 1
        l1 = len(line)
        line = line[:l1-1]
        if self.DebugMode == 1:
            print "Starting Create Table"
            print line
        if self.WriteFile == 1:
            OutFile.write(line)
```

```
CREATE TABLE `categoriesmain` (
  `idCategoriesMain` int(10) unsigned NOT NULL auto_increment,
  `CatText` char(100) NOT NULL default '',
  PRIMARY KEY  (`idCategoriesMain`)
) ENGINE=InnoDB AUTO_INCREMENT=40 DEFAULT CHARSET=latin1;
```

```
    p1 = line.find(AI)
    if line.startswith(") "):
        CreateTableMode = 0
        if self.DebugMode == 1:
            print "Finished Table Create"
        newline = ");\n"
        if self.WriteFile == 1:
            OutFile.write(newline)
            if self.DebugMode == 1:
                print "Writing Line {0}".format(newline)
```

```
    elif p1 != -1:
        # Line is primary key line
        l = line.strip()
        fnpos = l.find(" int(")
        if fnpos != -1:
            fn = l[:fnpos]
        newline = fn + IPK #+ ",\n"
        if self.WriteFile == 1:
            OutFile.write(newline)
            if self.DebugMode == 1:
                print "Writing Line {0}".format(newline)
```

```
    elif CreateTableMode == 1:
        # Parse the line...
        if self.DebugMode == 1:
            print "Line to process – {0}".format(line)
```

észre azt a kis extra szóközt a sor végén - ez tudatosan került ide. Ha találkoznánk ezzel a sorral, akkor egyszerűen figyelmen kívül fogjuk hagyni.

```
elif line.strip().starts-
with(PK):

    pass
```

Ezután a „ unsigned” kulcsszót nézzük meg (ismét megtartva a szóközt) és lecseréljük „ ”-re.

És ezzel meg is vagyunk a „create table” rutinnal. Most már foglalkozhatunk az „insert” utasításokkal. Az InsertStart változó tartalma az „INSERT INTO “ kifejezés. Ennek ellenőrzésére azért van szükség, mert a MySQL az SQLite-tal ellentétben megenged több beszúrást egyetlen utasítással. Ezért minden adatblokknál külön utasítást kell készítenünk. Az „insertmode” nevű változót „1”-re állítjuk, beolvassuk a „INSERT INTO {Tábla} {mező nevek} VALUES (“ részt egy segédváltozóba (amit röviden előzménynek fogok csak hívni), és továbblépünk.

Megnézzük, hogy csak sémákat kell-e feldolgoznunk. Ebben az esetben nyugodtan figyelmen kívül hagyhatjuk az insert utasításokat. Ha nem, külön ki kell térnünk rájuk.

```
elif self.SchemaOnly == 0:
   if insertmode == 1:
```

Leellenőrizzük, hogy van-e „');” vagy „'),” karaktersorozat a sorban. A „');” esetünkben azt fogja jelenteni, hogy elértük az insert utasítás utolsó sorát.

```
posx = line.find("');")
pos1 = line.find("'),")
l1 = line[:pos1]
```

Az alábbi sor megkeresi az „escape”-elt aposztrófokat és lecseréli őket.

```
line =
line.replace("\\'","''")
```

```
    elif line.startswith(InsertStart):
        if insertmode == 0:
            insertmode = 1
            # Get tablename and field list here
            istatement = line
            # Strip CR/LF from istatement line
            l = len(istatement)
            istatement = istatement[:l-2]
```

```
        elif line.find(" unsigned ") != -1:
            line = line.replace(" unsigned "," ")
            line = line.strip()
            l1 = len(line)
            line = line[:l1-1]
            if self.WriteFile == 1:
                OutFile.write("," + line)
                if self.DebugMode == 1:
                    print "Writing Line {0}".format(line)
```

Különben le kell kezelnünk.

```
        else:
            l1 = len(line)
            line = line.strip()
            line = line[:l1-4]
            if self.DebugMode == 1:
                print "," + line
            if self.WriteFile == 1:
                OutFile.write("," + line)
```

```
        if posx != -1:
            l1 = line[:posx+3]
            insertmode = 0
            if self.DebugMode == 1:
                print istatement + l1
                print "------------------------------"
            if self.WriteFile == 1:
                OutFile.write(istatement + l1+"\n")
```

Vagy, az első részt hozzákapcsoljuk az értékekhez és lezárjuk egy pontosvesszővel.

```
        elif pos1 != -1:
            l1 = line[:pos1+2]
            if self.DebugMode == 1:
                print istatement + l1 + ";"
            if self.WriteFile == 1:
                OutFile.write(istatement + l1 + ";\n")
```

Ha van insert végi záró utasításunk („);”), akkor létre tudjuk hozni az utasítást úgy, hogy az előzményt hozzácsatoljuk az éppen aktuális részhez.

Mindez csak akkor működik, ha az insert utasításban az utolsó érték egy idéző jelek között lévő sztring. Abban az esetben, amikor számra végződik, másképp kell megoldanunk a dolgokat. Szerintem magatoktól is értelmezni tudjátok majd azt, amit az alábbiakban csinálunk.

Végül lezárjuk az input fájlt, és ha külső fájlba írtunk, akkor azokat is.

```
f.close()
if self.WriteFile == 1:
   OutFile.close()
```

Amint elkészültünk a fájl konvertálásával, egyből használhatjuk az SQLite Database Browserrel az adatbázis feltöltésére.

Ennek a kódnak az esetek 90%-ban működnie kell. Még lehetnek olyan dolgok, amelyekkel nem foglalkoztunk különböző okok miatt, ezért is hagytuk benne a debug módot. Több fájlon is teszteltem, és semmi probléma nem merült föl.

Mindenesetre a kód megtalálható a PasteBinen: http://pastebin.com/cPvzNT7T.

Viszontlátásra legközelebb.

Greg Walters a RainyDay Solutions Kft. tulajdonosa, amely egy tanácsadó cég a coloradói Aurórában. Greg 1972 óta foglalkozik programozással. Szeret főzni, túrázni, zenét hallgatni, valamint a szabadidejét családjával tölteni. Weblapja a www.thedesignatedgeek.com címen található meg.

```
else:
    if self.DebugMode == 1:
        print "Testing line {0}".format(line)
    pos1 = line.find("),")
    posx = line.find(");")
    if self.DebugMode == 1:
        print "pos1 = {0}, posx = {1}".format(pos1,posx)
    if pos1 != -1:
        l1 = line[:pos1+1]
        if self.DebugMode == 1:
            print istatement + l1 + ";"
        if self.WriteFile == 1:
            OutFile.write(istatement + l1 + ";\n")
    else:
        insertmode = 0
        l1 = line[:posx+1]
        if self.DebugMode == 1:
            print istatement + l1 + ";"
        if self.WriteFile == 1:
            OutFile.write(istatement + l1 + ";\n")
```

Programozzunk Pythonban – 30. rész

Ebben a hónapban egy újabb GUI tervezőt fedezünk fel, ezúttal a Tkinter-t. A Tkinter-t sokan azért nem használják, mert nincs beépített tervezője. Korábban már bemutattam, hogyan lehet egyszerűen alkalmazásokat készíteni tervező nélkül, most ennek egy módját fogjuk megtekinteni. Page-nek hívják és alapvetően ez egy Python-t támogató Visual TCL változat. A jelenlegi verzió a 3.2-es és megtalálható itt: http://sourceforge.net/projects/page/files/latest/download.

Előfeltételek

Szükség lesz egy TCK/TK 8.5.4-re (vagy újabbra), Python 2.6-ra (vagy újabbra) és pyttk-ra - amit (ha még nincs) itt találsz: http://pypi.python.org/pypi/pyttk. Nagy valószínűséggel ezek mindegyikével rendelkezel már, kivétel talán csak a pyttk.

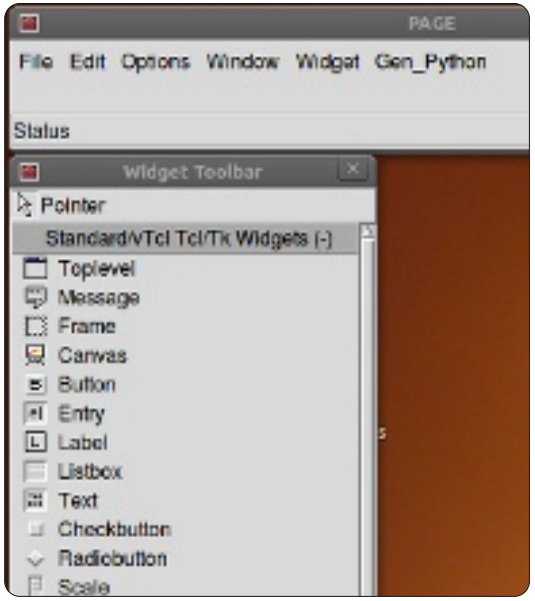
Telepítés

Egyszerű, mint a karikacsapás: ki kell csomagolni a fájlokat egy általunk választott könyvtárba, majd futtassuk itt le a „configure” parancsfájlt. Ezzel létrehozzuk a „page” indító parancsfájlt és a továbbiakban ezt használjuk mindenre.

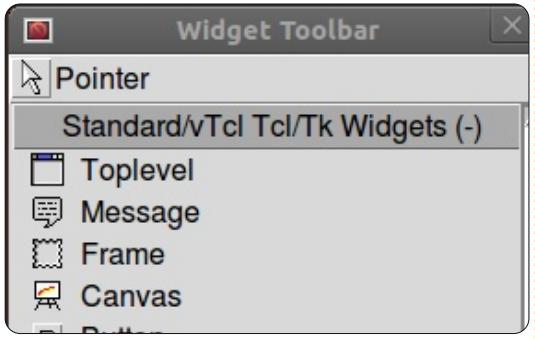
A Page használata

A Page elindítása során három ablakot (form-ot) kapunk: egy „launch pad”, egy eszköztár és egy Attribute Editor.

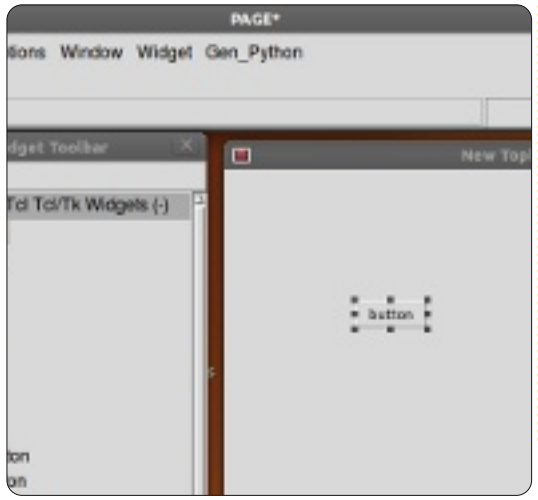
Új projektet az eszköztárban lévő gombra kattintva hozhatunk létre.



Ezzel egy új fő form-ot kapunk, amit a képernyőn szabadon mozgathatunk. Az eszköztárban további elemeket („widgeteket”) találunk, amelyek a fő formon belül szintén tetszőleges módon elhelyezhetők.



Egyelőre csak egy gombot hozzunk létre. Kattints az eszköztárban a Button gombra, majd ezután valahová a fő form-on belül.

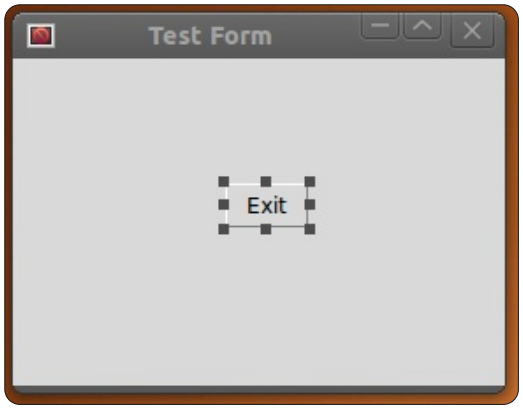


Ezután kattints a launch pad form-ban a Window-ra és válaszd ki az Attribute Editor-t (ha még nem jelent meg magától). Az egyetlen eddig létrehozott gombunk elvileg már ki is van emelve, mozgassuk tehát a form-on belül ahová szeretnénk, majd az egérgomb elengedése után az attribute editorban látnunk kellene az elem helyének megváltozását („x position”, „y position”).

Itt további attribútumokat is beállíthatunk, mint például a gombon (vagy bármilyen másik elemen) megjelenő szöveget, egy elem álnevét (ezen a néven hivatkozunk rá a kódban), a színt és így tovább. Közel az attribute editor aljához találjuk a szöveg mezőt. Ez azt a szöveget tartalmazza, amely a felhasználó előtt megjelenik, esetünkben a gomb elemét. Írjuk át a „button”-t „Exit”-re és vegyük észre, hogy a gombon mostantól ez utóbbi felirat szerepel. Most méretezzük át a form-ot és tegyük a gombot a form közepére.

Ezután a fő formon belül kattintsunk valahová (bárhová, csak ne

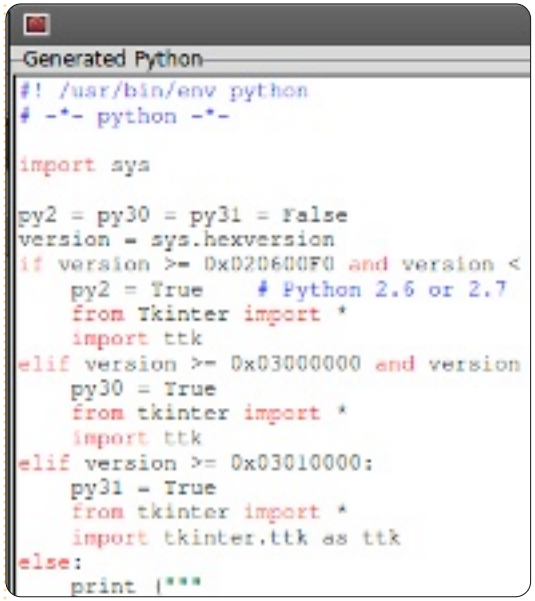
a gombra). Az attribute editor form most a fő form jellegzetességeit mutatja. Keressük meg a „title” mezőt és változtassuk meg „New Toplevel 1”-ről „Test Form”-ra.



Mielőtt elmentenénk a projektet, létre kell hoznunk egy könyvtárat a hozzá tartozó fájloknak, legyen ez a „PageProjects”. Ha megvan, a launch pad ablakban válasszuk a Fájl -> Mentés másként opciót és keressük meg az előbb létrehozott könyvtárat. A párbeszédablak dobozba írjuk azt, hogy TestForm.tcl és nyomjuk meg a mentés gombot. Vegyük észre, hogy a munkánkat TCL fájlként mentettük el, a python fájlt később hozzuk majd létre.

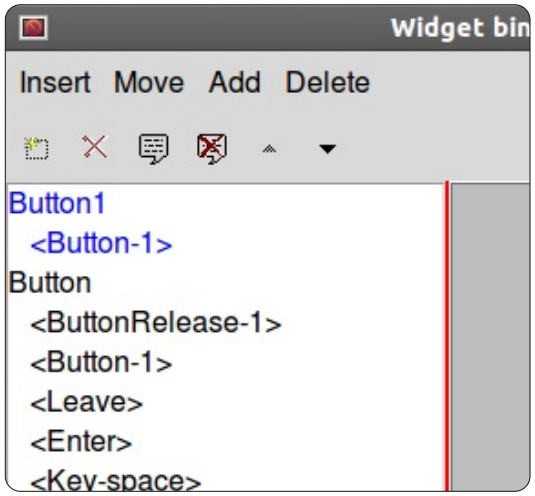
A launch pad-ben keressük meg a Gen_Python menü elemet és kattintsunk rá. Kiválasztva a Generate Python-t egy új form jelenik meg.

A Page (ahogy ezt a név is sugallja) legenerálta a python kódot és az ablakban meg is jelenítette azt számunkra. A form alján találunk három gombot... Save (mentés), Run (futtatás) és Close (bezárás).

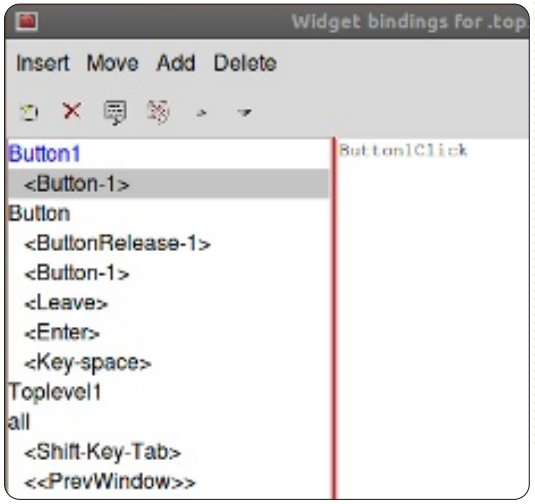


Mentsük el. Ha most belenézünk a PageProjects könyvtárba, találunk ott egy python fájlt (TestForm.py). Most kattintsunk a futtatás gombra. A projekt néhány másodpercen belül elindul. A gomb most még semmihez sincs hozzákötve, így ha meg is nyomjuk, nem csinál majd semmit. Zárjuk be a form-ot egyszerűen az ablak sarkában lévő „X”-szel. Zárjuk be a Python Console ablakot is.

Visszatérve a fő form-hoz, jelöljük ki az Exit gombot és kattintsunk rá a jobb gombbal. Válasszuk a „Bindings...”-et (kötések), ami alatt gombok gyűjteményét találjuk.



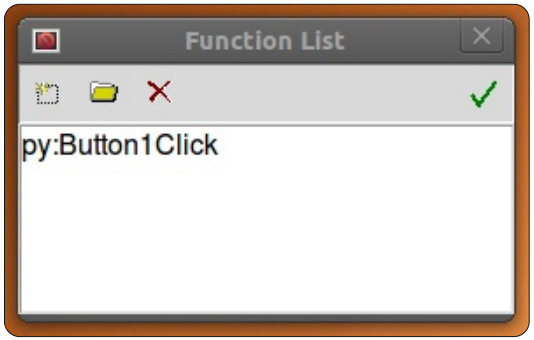
Új kötéseket a bal oldalon lévő elsővel tudunk létrehozni. Kattintsunk a „Button-1”-re, ezzel beállíthatjuk, hogy az egér bal gombjának megnyomására mi történjen. A



jobb oldali ablakba írjuk be azt, hogy „Button1Click”.

Mentsük el és generáljuk a python kódot újra. Görgessünk le a Python Console-ban a fájl aljára. A „class Test_Form” fölötti kódban megjelent az új függvény, amit az előbb létrehoztunk. Vegyük észre, hogy ez itt most csak egyszerűen átadódik. Menjünk lejjebb és látni fogjuk a kód azon részét, amely létrehozza és vezérli a gombunkat. A program lényegében mindent megcsinált helyettünk. Persze azt még meg kell mondanunk a gombnak, hogy mit csináljon. Zárjuk be a Python Console-t és folytassuk.

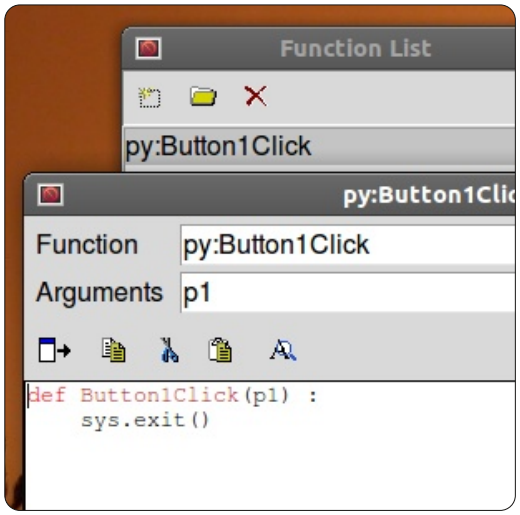
A launchpad-en kattintsunk a Window-ra majd válasszuk a Function List-et. Itt fogjuk megírni, hogy hogyan záródjon be az ablak.



Bal oldalon az első gomb az Add button (gomb hozzáadása). Kattintsunk rá és a függvény dobozba írjuk ezt: „py:Button1Click”, az argumentum dobozba pedig: „p1” és

végül az alsó dobozban változtassuk meg a szöveget erre:

```
def Button1Click(p1):
        sys.exit()
```



Kattintsunk a pipára és készen is vagyunk ezzel.

Most össze kellene kötnünk ezt a rutint a gombbal. Válasszuk ki a gombot a form-ban, jobb klikk és „Bindings...”. Ahogy azt már korábban is tettük, kattintsunk az eszköztár távoli bal gombjára és válasszuk a Button-1-et. Ez a bal egérgomb lenyomásának megfelelő esemény. A jobb oldali szövegdobozba írjuk be: „Button1Click”. Ügyeljünk rá, hogy valóban az előbb létrehozott függvényre hivatkozzunk. Jobb oldalon kattintsunk a pipára.

Mentsük el és generáljuk újra a python kódot.

Az aljához közel, de a Test_Form class-on KÍVÜL az alábbi kódrészletet kell látnunk:

```
def Button1Click(p1) :

sys.exit()
```

Az osztály utolsó sora pedig így néz ki:

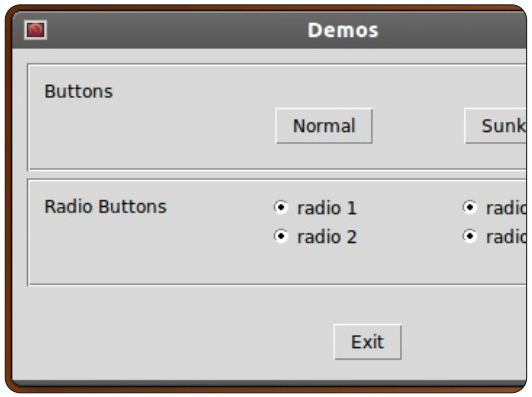
```
self.Button1.bind('<Button-
1>',Button1Click)
```

Ezek után a kódot futtatva és az Exit gombra kattintva a form elvileg megfelelően be fog záródni.

Menjünk tovább

Bonyolítsuk egy kicsit a dolgokat. Létre fogunk hozni egy demot, amelyben bemutatjuk az elérhető elemeket. Először zárjuk be a Page-t és indítsuk újra. Hozzunk létre egy új Toplevel form-ot. Adjunk hozzá két keretet (frame), egyiket a másik fölé és növeljük úgy, hogy azok lefedjék csaknem a form teljes szélességét. Helyezzünk el a felső keretben egy címkét és az attribute editort használva változtassuk meg a szöveget „Buttons:”-ra. Ezután adjunk hozzá két gombot a vízszintes sík mentén. A bal oldali szövege legyen „Normal”, a jobb oldalié „Sunken” (süllyesztett). A sunken gombnál változtassuk meg a kiemelést (relief) „sunken”-re és legyen a neve btnSunken. A „Normal” gombot nevezzük „btnNormal”-nak. Mentsük el a projektet „Demos.tcl” néven.

Helyezzünk el egy „Radio Buttons” címkét az alsó keretben és négy rádiógombot, ahogy azt a mellékelt ábra is mutatja. Végül az alsó keret alá helyezzük el az Exit gombot is.



Mielőtt a kötéseket beállítanánk, hozzuk létre a saját kattintás függvényünket. Nyissuk meg a Function List-et és készítsük el a btnNormalClicked és a btnSunkenClicked nevű függvényeket. Az argumentum dobozban p1 szerepeljen. Valahogy így nézzen ki a kód:

```
def btnNormalClicked(p1):

print "Normal Button Clicked"

def btnSunkenClicked(p1) :

print "Sunken Button Clicked"
```

Most pedig hozzuk létre a kötéseket. Az egyes gomboknál kattintsunk előbb jobb gombbal, válasszuk ki a „Bindings...”-et majd a hozzáadást és ahogy az előbb, kössük hozzá a függvényünkhöz. A normal gombra ez „btnNormalClicked”, a süllyesztettre pedig btnSunkenClicked. Mentsük el és generáljuk le a kódot. Ha most teszteljük le a programot a Python Console „Run” opciójával, akkor nem látunk semmit, de amikor bezárod az alkalmazást, a kijelzőn megjelennek a visszajelzések. A Page-nél ez normális és ha egyszerűen csak a parancssorból indítod, ahogy általában szokás, akkor a dolgok az elvárásoknak megfelelően fognak működni.

És most a rádiógombok. Két „klaszterbe” csoportosítottuk őket. Az első kettő (Radio 1 és Radio 2)

az elsőbe, míg a maradék kettő a második klaszterbe kerül. Kattints az Attribute Editor-ban a Radio1-re és állítsd be az értéket 0-ra, a változó pedig legyen „rcb1”. A Radio 2 esetén a megfelelő mezőkbe 1 és „rbc1” kerüljön. Ugyanez vonatkozik a 3-as és 4-es rádiógombra is, de a változó legyen „rbc2”. Ha szeretnéd, a rádiógombok megnyomására kiírathatsz valamit a terminál ablakba, de most nem ez a fontos, hanem hogy a klaszterek működnek. A Radio1-re kattintva a Radio2 kiválasztása törlődik és ez nincs hatással a Radio3 és 4-re.

Végül hozzunk még létre egy függvényt az Exit gombra és kössük össze a kettőt, ahogy azt az első példa során is tettük.

A most következő kód egy korábbi, Tkinter alkalmazásról szóló cikk alapján megérthető. Ha ez így zavaros, lapozz vissza néhány számnyit a Full Circle magazinban és találsz róla egy részletes leírást.

Láthatod, hogy az alap szintű tervezési folyamatot a Page használata LÉNYEGESEN leegyszerűsíti, mintha az egészet magunknak kellene írnunk. Amit itt bemutattam, az csak a felszín egy része, a Page ennél sokkal többet is tud, amit a következő számban egy sokkal valódibb példán keresztül szeretnék bemutatni.

```
def set_Tk_var():
# These are Tk variables passed to Tkinter and must
# be defined before the widgets using them are created.
global rbc1
rbc1 = StringVar()
global rbc2
rbc2 = StringVar()
def btnExitClicked(p1) :
sys.exit()
def btnNormalClicked(p1) :
print "Normal Button Clicked"
def btnSunkenClicked(p1) :
print "Sunken Button Clicked"
```

A python kód elérhető az alábbi linken: http://patbin.com/qq0YVgTb.

Még egy megjegyzés, mielőtt befejezném erre a hónapra. Talán feltűnt, hogy néhány számban nem volt python-os írás. Ennek oka, hogy a feleségemnél tavaly rákot diagnosztizáltak. Próbáltam kézben tartani a dolgokat, de néhány dolog így is kicsúszott. Az egyik ilyen a régi domain/web oldalam, a ~~www.thedesignatedgeek.com~~, amit elfelejtettem megújítani. A domaint közben eladták alólam, így minden ott tárolt anyagot igyekszem átköltöztetni a www.thedesignatedgeek.net címre. Elnézést kérek mindenkitől.

Legközelebb is találkozunk!

Greg Walters a RainyDay Solutions, LLC tanácsadó cég (Aurora, Colorado) tulajdonosa és 1972 óta programozik. Szeret főzni, túrázni, a zenét és az idejét a családjával tölteni. Honlapja: www.thedesignatedgeek.net.

ubuntu uk podcast

Az Ubuntu Podcast lefedi a legfrissebb híreket és kiadásokat amik általában érdekelhetik az Ubuntu Linux felhasználókat és a szabadszoftver rajongókat. Az műsor felkelti a legújabb felhasználók és a legöregebb fejlesztők érdeklődését is. A beszélgetésekben szó van az Ubuntu fejlesztéséről, de nem túlzottan technikai. Szerencsések vagyunk, hogy gyakran vannak vendégeink, így első kézből értesülünk a legújabb fejlesztésekről, ráadásul olyan módon ahogyan mindenki megérti! Beszélünk továbbá az Ubuntu közösségről is és a benne zajló dolgokról is.

A műsort a nagy-britanniai Ubuntu közösség tagjai szerkesztik. Mivel az Ubuntu viselkedési kódexnek megfelelően készítik, bárki meghallgathatja.

A műsor minden második hét keddjén élőben hallgatható (brit idő szerint), másnap pedig letölthető.

podcast.ubuntu-uk.org

Programozzunk Pythonban – 31. rész

Írta: Greg D. Walters

A legutóbbi írásom alapján most már talán van sejtésed arról, hogyan kell használni a Page-et. Ha mégsem, kérlek, olvasd el az előző havi cikkemet. A mai alkalommal egy GUI fájllista alkalmazás készítéséről lesz szó. A cél egy olyan GUI alkalmazás létrehozása, amely rekurzív módon végigjárja a mappákat, és megadott kiterjesztéssel rendelkező fájlokat keres. Az eredményt fa nézetben jelenítjük meg. Olyan médiafájlokat fogunk például keresni, mint az „.avi”, „.mkv”, „.mv4”, „.mp3” és „.ogg”.

Ez alkalommal a tervezés rész szövege talán kissé tömörnek fog tűnni. Most csak megmutatom a fő irányokat arra nézve, hogy hová tegyük a widgeteket, valamint megadom a szükséges attribútumokat és értékeket, valahogy így:

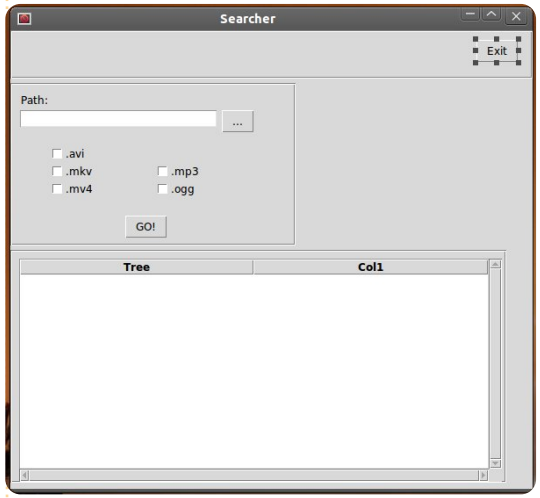
```
Widget

Attribute: Value
```

Szövegrészeket csak akkor fogok idézni, ha az szükséges. Például a gombok egyikén a „...” szöveg jelenjen meg.

Az alkalmazás GUI-ja tehát valahogy így fog kinézni:

Van tehát egy fő formunk, egy



kilépés gombunk és egy szövegbeviteli mezőnk egy olyan gombbal, amely egy mappa párbeszédablakot hív meg, 5 jelölőnégyzet a kiterjesztéstípusok megválasztásához, egy „GO!” gomb a folyamat tényleges indításához és egy fa nézet a kimenetünk megjelenítéséhez.

Kezdjük hát el. Indítsd el a Page-et, és hozz létre egy új top level widgetet. Az Attribute Editort használva állítsd be az alábbi tulajdonságokat:

```
Alias: Searcher
Title: Searcher
```

Gyakran készíts mentéseket, a fájl elmentésekor válaszd a „Searcher” opciót. Ne feledd, a Page egy .tcl kiterjesztésű fájlt hoz létre, és amikor végül legenerálod a python kódot, azt ugyanabba a mappába menti el.

Ezután hozzunk létre egy keretet, amely a fő keret tetejéről indul. Használjuk az alábbi beállításokat:

```
Width: 595
Height: 55
x position: 0
y position: 0
```

Adjunk hozzá egy Exit gombot ehhez a kerethez:

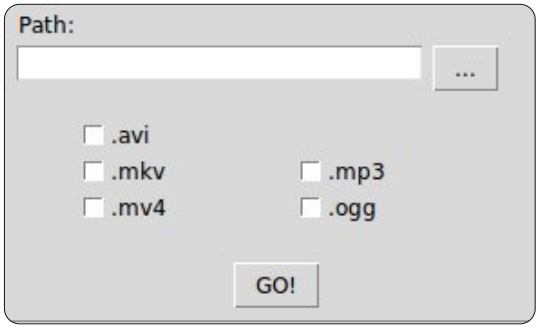
```
Alias: btnExit
Text: Exit
```

Tegyük ezt a keret közepébe, vagy a keret jobb oldalára. Én az X 530 és Y 10 koordinátákat választottam.

Hozzunk létre egy újabb keretet.

```
Width: 325
Height: 185
y position: 60
```

A keret tehát valahogy így fog kinézni:



Adjunk hozzá egy címkét. A szöveg attribútumát állítsuk „Path”-ra és mozgassuk a keret bal felső sarkához.

Ugyanebben a keretben hozzunk létre egy új widgetet.

```
Alias: txtPath
Text: FilePath
Width: 266
Height: 21
```

Most hozzunk létre egy gombot a widget jobb oldalán.

```
Alias: btnSearchPath
Text: "..." (no quotes)
```

Ezután készítsünk öt (5) darab check gombot. Az alábbi módon rendezzük el őket:

```
x
x x
x x
```

A három bal szélső video-, a két jobb oldali pedig audiofájlok kezelésére szolgál majd. Először a három bal oldalit állítjuk be, majd a két jobb oldalit is:

```
Alias: chkAVI
Text: ".avi" (no quotes)
Variable: VchkAVI

Alias: chkMKV
Text: ".mkv" (no quotes)
Variable: VchkMKV

Alias: chkMV4
Text: ".mv4" (no quotes)
Variable: VchkMV4

Alias: chkMP3
Text: ".mp3" (no quotes)
Variable: VchkMP3

Alias: chkOGG
Text: ".ogg" (no quotes)
Variable: VchkOGG
```

Végül még ugyanebben a keretben valahol az öt jelölőnégyzet alatt a keret közepére igazítva hozzunk létre egy gombot.

```
Alias: btnGo
Text: GO!
```

Most pedig hozzunk létre egy újabb keretet az utoljára használt alatt.

```
Width: 565
Height: 265
```

Én az X 0 és Y 250 helyre raktam. Előfordulhat, hogy a fő formot át kell majd méretezned, hogy az egész keret látszódjon. Ezen a kereten belül egy olyan widgetet alkotunk, amely lapozó fa nézetet biztosít számunkra.

```
Width: 550
Height: 254
X Position: 10
Y Position: 10
```

Így. Megterveztük hát a saját GUI-nkat. Már csak a függvények létrehozása és a gombokkal történő összekapcsolás van hátra.

A Függvénylista ablakban kattints a New gombra (bal oldalon), új függvények szerkesztésére itt van lehetőség. Változtassuk meg a függény mezejét „py: xxx"-ről „py:btnExitClick()"-re, az argumentum mezőt pedig „p1"-re. Az alsó többsoros beviteli mezőben szerepeljen ez a szöveg:

```
def btnExitClick(p1):

    sys.exit()
```

Vegyük észre, hogy itt nem használtunk bekezdést (a Page ezt automatikusan megcsinálja a python fájl generálásánál). Hozzunk létre egy újabb függvényt „btnGoClick" névvel. Átadott (passed) paraméterként adjuk hozzá: „p1". Ide majd még visszatérünk.

Végül hozzunk létre egy „btnSearchPath" függvényt. A pass-szal ismét nem foglalkozzunk. A gombokat még össze kell kötnünk a létrehozott függvényekkel.

Jobb klikk az általunk létrehozott kilépés gombra, és válasszuk a kötést (bind). Egy nagy doboz fog felugrani: kattintsunk a New binding gombra, majd a Button-1-re, és változtassuk meg a jobb oldali mezőben a „TODO" szöveget „btnExitClick"-re. NE használjuk a () jeleket itt.

A GO gombot kössük össze a btnGoClick-el és a „..." gombot a btnSearchPathClick-kel.

Mentsük el a GUI-t, és generáljunk python kódot.

Most már csak létre kell hoznunk a kódot, ami „összeragasztja" a GUI-t.

A generált kódot nyissuk meg a kedvenc szövegszerkesztőnkkel. Nézzük meg alaposabban, mit csinált a Page.

A fájl tetején egy standard python header és egy (a sys könyvtárra vonatkozó) import utasítás áll. Ami ezután következik, az kissé zavaros (legalábbis elsőre). A kód megnézi, hogy milyen verziójú pythonnal szeretnénk futtatni az alkalmazást, és ennek alapján importálja a megfelelő tkinter programkönyvtárakat. Ha csak nem 3.x pythont használsz, lényegében nem szükséges foglalkozni ezzel a résszel.

A 2.x kódrészletet fogjuk úgy módosítani, hogy további tkinter modulokat is importáljunk.

A következő rutin a „vp_start_gui()", ami a program fő részét képezi. Beállítja a gui-t, a szükséges változókat, és ciklikusan meghívja a tkintert. A kódban talán feltűnik a „w = None" sor, ennek valójában nem kellene ott lennie.

Ezután két olyan rutin következik (create_Searcher és destroy_Searcher), amelyek a fő ciklust szokták helyettesíteni, ha az alkalmazást programkönyvtárként hívjuk meg. Ezzel most nem kell foglalkoznunk.

A „set_Tk_var" rutin következik. A widget létrehozása előtt először a tkinter változókat definiáljuk. A következő három rutint a függvény editorral létrehozott funkcióink, valamint egy „init()" funkció alkotja.

Futtassuk most le a programot. Vegyük észre, hogy a kijelölő gombok alapból be vannak szürkítve. Mi ezt nem szeretnénk, szóval szükség van még némi kódolásra. A jelölőnégyzeteken kívül csak az Exit gombunk működik egyelőre.

Zárjuk be a programot.

Most a GUI definíciókat tartalmazó osztályt fogjuk megvizsgálni, ez a „class Searcher". Itt vannak megadva a widgetek és azok helye a saját űrlapunkon. Mostanra ez már valószínűleg ismerős.

Két további osztály jön létre a görgethető fanézetet támogató kód elhelyezéséhez. Nem szükséges hozzányúlnunk, ezt a Page készítette nekünk.

Ugorjunk vissza a kód tetejére, és változtassunk rajta. Importálnunk kell néhány további könyvtár modult, ezért az „import sys" utasítást egészítsük ki:

```
import os

from os.path import join,
getsize, exists
```

Most keressük meg azt a részt, ahol a „py2 = True" kifejezés szerepel. Itt történik a tkinter importálások kezelése Python 2.x esetén. Az „import ttk" alatt az alábbiakat kell írnunk ahhoz, hogy a FileDialog könyvtár támogatott legyen. Továbbá a tkFont modul importálására is szükség van.

```
import tkFileDialog

import tkFont
```

Hozzá kell adnunk a „set_Tk_var()" rutinhoz néhány újabb változót. A rutin végére írjuk oda:

```
global exts, FileList

exts = []

FileList=[]
```

Itt két globális változót hozunk létre (exts és FileList), amelyekre a kód későbbi részében fogunk még hivatkozni. Mind a kettő egy lista, az „exts" a felhasználó által GUI-ban választott kiterjesztéseket tartalmazza, a „FileList" pedig a keresési feltételeknek megfelelő fájlokat tartalmazza.

A „btnExitclick"-et a Page már létrehozta nekünk, így most csak a „btnGoClick" rutinnal foglalkozunk. Kommenteljük ki a „pass" utasítást, és adjuk a kódhoz, hogy valahogy így nézzen ki...

```
def btnGoClick(p1) :

#pass

BuildExts()

fp = FilePath.get()

e1 = tuple(exts)

Walkit(fp,e1)

LoadDataGrid()
```

Ezt a rutint hívjuk meg akkor, ha a felhasználó megnyomja a „GO!" gombot. Meghívjuk a „BuildExts" névre hallgató rutint is, amely a felhasználó által választott kiterjesztések listáját hozza létre. Ezután az AskDirectory párbeszédablakban megadott útvonalat kapjuk meg és rendeljük hozzá az fp változóhoz. Létrehozunk egy rendezett n-est a kiterjesztéslistából, amit a fájlok ellenőrzésénél fogunk használni. Meghívjuk a „Walkit" rutint, átadva neki a célmappát és a kiterjesztéslistát, és végül a „LoadDataGrid" hívásával zárjuk a sort.

Most szedjük ki a „btnSearchPathClick"-et: kommenteljük ki az eredetit, és helyette az alábbiak szerepeljenek:

```
def btnSearchPathClick(p1) :

    #pass

    path = tkFileDialog.ask
directory() #**self.file_opt)

    FilePath.set(path)
```

Az init rutin következik:

```
def init():
```

```
    #pass

    # Fires AFTER Widgets and
Win

dow are created...

    global treeview

    BlankChecks()

    treeview = w.Scrolledtre-
eview1

    SetupTreeview()
```

Létrehozunk egy „treeview" nevezetű globált, és meghívjuk azt a rutint, amely eltávolítja a szürke jelzéseket a jelölőnégyzetekből, valamint a „treeview" változót a formunk görgethető fanézetéhez rendeli. Végül meghívjuk a „SetupTreeview"-t, amely az oszlopok fejléceit állítja be.

A BlankChecks rutinra is szükségünk lesz:

```
def BlankChecks():

    VchkAVI.set('0')

    VchkMKV.set('0')

    VchkMP3.set('0')

    VchkMV4.set('0')

    VchkOGG.set('0')
```

Annyi történik, hogy megfelelően beállítjuk a változókat (ami automatikusan beállítja a jelölőnégyzeteket) „0" értékre. Ha a jelölőnégyzetre kattintunk, a változó értéke automatikusan frissül majd. Ha a változót a saját kódunk által módosítjuk, a jelölőnégyzet erre is reagál. A folytatásban a felhasználó kattintásai által kiválasztott kiterjesztéslista létrehozásával foglalkozunk.

Emlékezz vissza a 35. számban megjelent 9. írásomra: olyan kódot írtunk, amellyel az MP3 fájlainkat katalogizáltuk. Ennek egy rövidített változatát fogjuk most használni. Ha kérdések merülnének fel ezzel kapcsolatban, javaslom, nézz bele a 35. FCM számba.

A SetupTreeview meghívása következik. Létrehozunk egy „ColHeads" változót, amely a fa nézet fejlécét tartalmazza, mindezt egy listaként, majd beállítjuk minden oszlopra. Az oszlop szélességét a fejléc méretéhez igazítjuk.

Végül létre kell hoznunk a „LoadDataGrid"-et, amely biztosítja a fa nézethez szükséges adataink beolvasását. Minden sor egy külön bejegyzés a FileList lista változónkban. Az oszlopok szélességét (újra) az oszlopokban szereplő adatok méretéhez igazítjuk.

Ennyi lenne. Futtassuk le, és nézzük meg, mit csináltunk. Számoljunk azzal, hogy amennyiben sok fájllal dolgozunk, úgy tűnhet, hogy a program nem reagál. Ezzel még érdemes kezdeni valamit. Létre fogunk hozni egy olyan rutint, amely erre az időre megváltoztatja a kurzor alakját „watch" stílusúra, így ha egy időigényes dolgot csinálunk, azt a felhasználó is láthatja.

```
def BuildExts():
     if VchkAVI.get() == '1':
          exts.append(".avi")
     if VchkMKV.get() == '1':
          exts.append(".mkv")
     if VchkMP3.get() == '1':
          exts.append(".mp3")
     if VchkMV4.get() == '1':
          exts.append(".mv4")
     if VchkOGG.get() == '1':
          exts.append(".ogg")
```

```
def Walkit(musicpath,extensions):
     rcntr = 0
     fl = []
     for root, dirs, files in os.walk(musicpath):
          rcntr += 1 # This is the number of folders we have walked
          for file in [f for f in files if f.endswith(extensions)]:
               fl.append(file)
               fl.append(root)
               FileList.append(fl)
               fl=[]
```

```
def SetupTreeview():
     global ColHeads
     ColHeads = ['Filename','Path']
     treeview.configure(columns=ColHeads,show="headings")
     for col in ColHeads:
          treeview.heading(col, text = col.title(),
               command = lambda c = col: sortby(treeview, c, 0))
     ## adjust the column's width to the header string
     treeview.column(col, width =
tkFont.Font().measure(col.title()))
```

A „set_Tk_var" rutin végére írjuk az alábbiakat:

```
    global busyCursor,pre
BusyCursors,busyWidgets
    busyCursor = 'watch'
    preBusyCursors = None
    busyWidgets = (root, )
```

Beállítjuk a globális változókat, és hozzárendeljük őket a widget(ek)hez (a busyWidgetsen belül), ami(k)re hivatkozni akarunk, esetünkben a gyökérhez, amely a mi teljes ablakunk. Vegyük észre, hogy ez is egy rendezett n-es.

Készítünk még két további rutint a kurzor beállítására (set, unset). Először nézzük a set beállítást: „busyStart". A „LoadDataGrid" után írjuk be a jobbra, középen látható kódot.

Előbb ellenőrizzük, hogy az értéket átadtuk-e a „newcursor"-nak. Ha nem, alapbeállításként a busyCursor-t használjuk. Ezután végigfutunk a busyWidget listán, és beállítjuk a kurzort úgy, ahogy szeretnénk. A jobbra, lent található kódot írjuk be.

Itt lényegében a busyWidget listában szereplő widgetek esetében állítjuk be a kurzort úgy, ahogy azt az alapbeállítások megkövetelik.

Mentsük el és futtassuk le a programot. Ha egy hosszú fájllistát olvasunk be, akkor a kurzor alakja most már meg fog változni.

Ez az alkalmazás tulajdonképpen nem sok mindent csinál, de azt azért megmutatja, hogyan lehet hatékonyan használni a Page-et. Egy GUI tervezése és elkészítése tehát nem feltétlenül nehéz és fájdalmas feladat.

```
def LoadDataGrid():
    global ColHeads
    for c in FileList:
        treeview.insert('','end',values=c)
        # adjust column's width if necessary to fit each value
        for ix, val in enumerate(c):
            col_w = tkFont.Font().measure(val)
            if treeview.column(ColHeads[ix],width=None)<col_w:
                treeview.column(ColHeads[ix], width=col_w)
```

```
def busyStart(newcursor=None):
    global preBusyCursors
    if not newcursor:
        newcursor = busyCursor
    newPreBusyCursors = {}
    for component in busyWidgets:
        newPreBusyCursors[component] = component['cursor']
        component.configure(cursor=newcursor)
        component.update_idletasks()
        preBusyCursors = (newPreBusyCursors, preBusyCursors)
```

```
def busyEnd():
    global preBusyCursors
    if not preBusyCursors:
        return
    oldPreBusyCursors = preBusyCursors[0]
    preBusyCursors = preBusyCursors[1]
    for component in busyWidgets:
        try:
            component.configure(cursor=oldPreBusyCursors[component])
        except KeyError:
            pass
        component.update_idletasks()
```

A tcl fájl elérhető a pastebinről (http://pastebin.com/AA1kE4Dy), akárcsak a python kód:

http://pastebin.com/VZm5un3e.

Viszlát legközelebb.

Közreműködnél?

Az olvasóközönségtől folyamatosan várjuk a magazinban megjelenítendő új cikkeket! További információkat a cikkek irányvonalairól, ötletekről és a kiadások fordításairól a http://wiki.ubuntu.com/UbuntuMagazine wiki oldalunkon olvashatsz.
Cikkeidet az alábbi címre várjuk: articles@fullcirclemagazine.org
A **magyar fordítócsapat** wiki oldalát itt találod: https://wiki.ubuntu.com/UbuntuMagazine/TranslateFullCircle/Hungarian
A magazin eddig megjelent **magyar fordításait** innen töltheted le: http://www.fullcircle.hu
Ha **email**-t akarsz írni a magyar fordítócsapatnak, akkor erre a címre küldd: fullcirclehu@gmail.com

Ha **hírt** szeretnél közölni, megteheted a következő címen: news@fullcirclemagazine.org

Véleményed és Linux-os **tapasztalataidat** ide küldd: letters@fullcirclemagazine.org

Hardver és szoftver **elemzéseket** ide küldhetsz: reviews@fullcirclemagazine.org

Kérdéseket a 'Kérdések és Válaszok' rovatba ide küldd: questions@fullcirclemagazine.org

Az én asztalom képeit ide küldd: misc@fullcirclemagazine.org

... vagy látogasd meg **fórumunkat**: www.fullcirclemagazine.org

A FULL CIRCLE-NEK SZÜKSÉGE VAN RÁD!

Egy magazin, ahogy a Full Circle is, nem magazin cikkek nélkül. Osszátok meg velünk véleményeiteket, desktopjaitok kinézetét és történeteiteket. Szükségünk van a Fókuszban rovathoz játékok, programok és hardverek áttekintő leírására, a Hogyanok rovatban szereplő cikkekre (K/X/Ubuntu témával); ezenkívül, ha bármilyen kérdés, javaslat merül fel bennetek, nyugodtan küldjétek a következő címre: articles@fullcirclemagazine.org

A Full Circle Csapata

Szerkesztő - Ronnie Tucker
ronnie@fullcirclemagazine.org
Webmester - Rob Kerfia
admin@fullcirclemagazine.org
Kommunikációs felelős - Robert Clipsham
mrmonday@fullcirclemagazine.org
Podcast - Robert Catling
podcast@fullcirclemagazine.org

Full Circle Magazin
Magyar Fordítócsapat

Koordinátor:
Pércsy Kornél
Fordítók:
Palotás Anna
Tömösközi Máté Ferenc

Lektor:
Balogh Péter

Szerkesztő/Korrektor:
Heim Tibor

Nagy köszönet a Canonicalnek és a fordítócsapatoknak világszerte, továbbá **Thorsten Wilms**-nek a jelenlegi Full Circle logóért.