



Full Circle

AZ UBUNTU LINUX KÖZÖSSÉG FÜGGETLEN MAGAZINJA

Programozói sorozat – Különkiadás



Programozói sorozat
Különkiadás

PROGRAMOZZUNK PYTHONBAN 4. Kötet

Full Circle

AZ UBUNTU LINUX KÖZÖSSÉG FÜGGETLEN MAGAZINJA



Programozzunk Pythonban
22. rész 3. oldal



Programozzunk Pythonban
23. rész 11. oldal



Programozzunk Pythonban
24. rész 18. oldal

Üdvözöllek egy újabb, „egyetlen témáról szóló külökiadásban”

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

Most ez a „**Programozzunk Pythonban**” 22–26. részének az újabb kiadása (a magazin 48–52. 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 megpróbálsz emulálni/utánozni a külökiadásokban lévő ismertetőket. Előfordulhat, hogy a szoftver későbbi verziói vannak meg neked, vagy érhetőek el a kiadásod tárolóiban.

Jó szórakozást!



Programozzunk Pythonban
25. rész 24. oldal



Programozzunk Pythonban
26. rész 32. oldal



Minden szöveg- és képanyag, amelyet a magazin tartalmaz, a Creative Commons Nevezd meg! - Így add tovább! 3.0 Unported License alatt kerül kiadásra. Ez annyit jelent, hogy átdolgozhatod, másolhatod, terjesztheted és továbbadhatod a cikkeket a következő feltételekkel: jelezned kell eme szándékodat a szerzőnek (legalább egy név, e-mail cím vagy url eléréssel), valamint fel kell tüntetni a magazin nevét („Full Circle magazin”).

és az url-t, ami a www.fullcirclemagazine.org (úgy terjeszd a cikkeket, hogy ne sugalmazzák azt, hogy te készítetted őket, vagy a te munkád van benne). Ha módosítasz, vagy valamit átdolgozol benne, akkor a munkád eredményét ugyanilyen, hasonló vagy ezzel kompatibilis licenz alatt leszel köteles terjeszteni.

A Full Circle magazin teljesen független a Canonicaltól, az Ubuntu projektek támogatójától. A magazinban megjelenő vélemények és állásfoglalások a Canonical jóváhagyása nélkül jelennek meg.



Helyesbítés

A múlt hónapi 21. részben azt mondtam, hogy a mentést „PlaylistMaker.glade” néven tegyék, de a kódban már „playlistmaker.glade”-ként volt hivatkozva.

Gondolom, észrevettétek, hogy az egyikben vannak nagybetűk, a másikon pedig nincsenek. A kód csak akkor működik, ha mind a hívásban, mind a fájlnevében egyformán használjuk.

Hogy megfelelően tudjunk a munkához látni, szükségünk lesz az előző havi playlistmaker.glade és a playlistmaker.py fájlokra. Ha nem lennének meg, akkor a részletekért lapozd fel az előző havi számot. Még mielőtt nekiülnénk a kódnak, nem árt, ha megnézzük, hogy valójában mi egy playlist fájl. Több fajtájuk is van, mindegyiknél más-más a kiterjesztéssel. Amelyikkel mi foglalkozunk, az a *.m3u. Ez a legegyszerűbb alakjában egy „#EXTM3U”-val kezdődő szöveges fájl, ami után minden egyes lejátszandó dal fájlának bejegyzése található – beleértve a

teljes elérési utat. Továbbá létezik egy bővítése is, ami minden bejegyzéshez tartalmazza a dal hosszát, a dal albumának nevét, a sorszámát és magát a dal nevét. Ezenlőre figyelmen kívül hagyjuk ezt a kiegészítést és csak az alap verzióra koncentrálunk.

Itt van egy példa M3U fájl:

```
#EXTM3U
Adult Contemporary/Chris
Rea/Collection/02 - On The
Beach.mp3
Adult Contemporary/Chris
Rea/Collection/07 - Fool
(If You Think It's
Over).mp3
Adult Contemporary/Chris
Rea/Collection/11 - Looking
For The Summer.mp3
```

Minden elérési út relatív a playlist fájl helyéhez.

Nos, lássunk is hozzá! Itt van jobbra a múlt havi kódunk eleje.

Most létre kell hoznunk az eseménykezelő rutint minden egyes előforduló eseményhez. Vegyük észre, hogy a _MainWindow_destroy és az on_tbtnQuit_clicked már

```
#!/usr/bin/env python
import sys
from mutagen.mp3 import MP3
try:
    import pygtk
    pygtk.require("2.0")
except:
    pass
try:
    import gtk
    import gtk.glade
except:
    sys.exit(1)
```

a következő az osztály definíció

```
class PlaylistCreator:
    def __init__(self):
        self.gladefile = "playlistmaker.glade"
        self.wTree = gtk.glade.XML(self.gladefile, "MainWindow")
```

és a main függvény

```
if __name__ == "__main__":
    plc = PlaylistCreator()
    gtk.main()
```

Itt van a dictionary-nk, amit az __init__ rutin után kell elhelyezni.

```
def SetEventDictionary(self):
    dict = {"on_MainWindow_destroy": gtk.main_quit,
            "on_tbtnQuit_clicked": gtk.main_quit,
            "on_tbtnAdd_clicked": self.on_tbtnAdd_clicked,
            "on_tbtnDelete_clicked": self.on_tbtnDelete_clicked,
            "on_tbtnClearAll_clicked": self.on_tbtnClearAll_clicked,
            "on_tbtnMoveToTop_clicked": self.on_tbtnMoveToTop_clicked,
            "on_tbtnMoveUp_clicked": self.on_tbtnMoveUp_clicked,
            "on_tbtnMoveDown_clicked": self.on_tbtnMoveDown_clicked,
            "on_tbtnMoveToBottom_clicked": self.on_tbtnMoveToBottom_clicked,
            "on_tbtnAbout_clicked": self.on_tbtnAbout_clicked,
            "on_btnGetFolder_clicked": self.on_btnGetFolder_clicked,
            "on_btnSavePlaylist_clicked": self.on_btnSavePlaylist_clicked}
    self.wTree.signal_autoconnect(dict)
```

készen vannak, és még 10 hátra van. Most csak helyet hagyunk ki

nekik.

A csonka rutinokat mindjárt módosítjuk. Ez viszont elég ahhoz, hogy elinduljon az alkalmazásunk és tesztelni tudjunk pár dolgot. Ehhez azonban még egy sort el kell helyeznünk az `__init__`-ben. A `self.wTree` után írjuk be a következőt:

```
self.SetEventDictionary()
```

Most, ha már futatjuk az alkalmazást, láthatjuk az ablakunkat, és rá tudunk kattintani a kilépéshez az eszköztár Quit gombjára. Mentjük el a kódot „playlistmaker-1a.py” néven és futtassuk. Ne felejtsük el ugyanabba a mappába menteni, mint a múltkori glade fájlt, vagy másoljuk a glade-et a forrásunk könyvtárába.

Mindezen felül még létre kell hoznunk néhány változót későbbi használatra. Ezeket az `__init__` `SetEventDictionary` hívása után kellene begépelnünk.

```
self.CurrentPath = ""  
self.CurrentRow = 0  
self.RowCount = 0
```

Most létrehozzuk azt a felugró ablakot létrehozó rutint, amivel információkat közölhetünk a felhasználóval. Van egy beépített függvény család erre, de egy saját rutint fogunk létrehozni,

hogy megkönnyítsük a dolgunkat. Ez a `gtk.MessageDialog` rutin lesz és a szintaxisa a következőképpen alakul:

```
gtk.MessageDialog(  
parent, flags, MessageType, Buttons, message)
```

Pár dolgot meg kell még beszélnünk, mielőtt tovább lépnénk. Az üzenet típusa az alábbiak közül kerülhet ki:

GTK_MESSAGE_INFO - Információs szöveg
GTK_MESSAGE_WARNING - Nem végzetes üzenet
GTK_MESSAGE_QUESTION - Döntést igénylő kérdés
GTK_MESSAGE_ERROR - Végzetes hiba

És a gombok típusai:

GTK_BUTTONS_NONE - nincs gomb
GTK_BUTTONS_OK - egy OK gomb
GTK_BUTTONS_CLOSE - egy Bezárás gomb
GTK_BUTTONS_CANCEL - egy Mégsem gomb
GTK_BUTTONS_YES_NO - Igen és Nem gombok
GTK_BUTTONS_OK_CANCEL - OK és Mégsem gombok

Alapból a következő kódot,

```
def on_tbtnAdd_clicked(self,widget):  
pass  
def on_tbtnDelete_clicked(self,widget):  
pass  
def on_tbtnClearAll_clicked(self,widget):  
pass  
def on_tbtnMoveToTop_clicked(self,widget):  
pass  
def on_tbtnMoveUp_clicked(self,widget):  
pass  
def on_tbtnMoveDown_clicked(self,widget):  
pass  
def on_tbtnMoveToBottom_clicked(self,widget):  
pass  
def on_tbtnAbout_clicked(self,widget):  
pass  
def on_btnGetFolder_clicked(self,widget):  
pass  
def on_btnSavePlaylist_clicked(self,widget):  
pass
```

vagy ennek egy változatát használunk dialógusok létrehozásához, megjelenítéséhez, olvasásához és eltakarításához.

```
dlg = gtk.MessageDialog(  
None,0,gtk.MESSAGE_INFO,  
gtk.BUTTONS_OK,"This is a  
test message...")  
response = dlg.run()  
dlg.destroy()
```

Mindazonáltal, ha egy olyan üzenetet akarsz megjeleníteni, amit egy-két alkalomnál többször láthat a felhasználó, akkor ez IGEN SOK gépelést jelent. Az alapszámba az, hogy ha egy kódrészletet többször írsz le, akkor azt ki kell

emelni egy függvénybe. Gondoljunk erre úgy, mintha kb. 10 alkalommal szeretnénk megjeleníteni valamit az alkalmazásban, ez 10 x 3 (azaz 30) sor kódot jelent. Egy függvény segítségével hívásával (az előbbi példában) ez mindössze 10 + 3-ra (azaz 13) redukálódna. Minél többször hívunk egy adott dialógust, annál kevesebb kódot kell írunk, nem is beszélve arról, hogy sokkal olvashatóbb is lesz. A függvényünk segítségével bármelyik üzenet típust meghívhatjuk a paraméterek változtatásával.

Ez egy nagyon egyszerű függvény, amit a következőképpen

hívnánk meg:

```
self.MessageBox("info", "The  
button QUIT was clicked")
```

Vegyük észre, hogy ha a MESSAGE_QUESTION-t használjuk, akkor két különböző választ kaphatunk (Az „Igen”-t vagy a „Nem”-et). Bármelyik gombra is kattint a felhasználó, annak eredményét a kódban visszakapjuk. Egy kérdés párbeszéd ablak használatához valami ilyesmit írunk:

```
response = self.Message-  
Box("question", "Are you sure  
you want to do this now?")
```

```
if response == gtk.RESPON-  
SE_YES:
```

```
    print "Yes was clicked"
```

```
elif response == gtk.RESPON-  
SE_NO:
```

```
    print "NO was clicked"
```

A visszaadott értéket le tudjuk ellenőrizni. Az itt jobbra látható módon cseréljük le a „pass” hívásokat mindegyik eseménykezelőben.

Ez még nem végleges, de már jelzi, hogy hogyan működnek a gombok. Mentsük a kódot „playlistmaker-1b.py” néven, és futtas-

```
def MessageBox(self, level, text):  
    if level == "info":  
        dlg = gtk.MessageDialog(None, 0, gtk.MESSAGE_INFO, gtk.BUTTONS_OK, text)  
    elif level == "warning":  
        dlg = gtk.MessageDialog(None, 0, gtk.MESSAGE_WARNING, gtk.BUTTONS_OK, text)  
    elif level == "error":  
        dlg = gtk.MessageDialog(None, 0, gtk.MESSAGE_ERROR, gtk.BUTTONS_OK, text)  
    elif level == "question":  
        dlg = gtk.MessageDialog(None, 0, gtk.MESSAGE_QUESTION, gtk.BUTTONS_YES_NO, text)  
    if level == "question":  
        resp = dlg.run()  
        dlg.destroy()  
        return resp  
    else:  
        resp = dlg.run()  
        dlg.destroy()
```

suk. Most egy olyan függvényt fogunk létrehozni, ami beállítja a widgetek referenciáit. Ez a rutin csak egyszer fog meghívódni, és így sokkal kezelhetőbb és olvashatóbb lesz a kódunk. Gyakorlatilag olyan helyi változókat akarunk létrehozni, amik a glade ablak widgetjeire hivatkoznak – tehát bármikor el tudjuk érni őket, ha szükségünk van rájuk. Tegyük a függvényt a SetEventDictionary után.

Vegyük észre, hogy van egy olyan dolog, amit nem használunk a rutinunkban. Ez pedig a treeview widget. A hivatkozását akkor hozzuk majd

```
def on_tbtnAdd_clicked(self, widget):  
    self.MessageBox("info", "Button Add was clicked...")  
def on_tbtnDelete_clicked(self, widget):  
    self.MessageBox("info", "Button Delete was clicked...")  
def on_tbtnClearAll_clicked(self, widget):  
    self.MessageBox("info", "Button ClearAll was clicked...")  
def on_tbtnMoveToTop_clicked(self, widget):  
    self.MessageBox("info", "Button MoveToTop was clicked...")  
def on_tbtnMoveUp_clicked(self, widget):  
    self.MessageBox("info", "Button MoveUp was clicked...")  
def on_tbtnMoveDown_clicked(self, widget):  
    self.MessageBox("info", "Button MoveDown was clicked...")  
def on_tbtnMoveToBottom_clicked(self, widget):  
    self.MessageBox("info", "Button MoveToBottom was clicked...")  
def on_tbtnAbout_clicked(self, widget):  
    self.MessageBox("info", "Button About was clicked...")  
def on_btnGetFolder_clicked(self, widget):  
    self.MessageBox("info", "Button GetFolder was clicked...")  
def on_btnSavePlaylist_clicked(self, widget):  
    self.MessageBox("info", "Button SavePlaylist was clicked...")
```

létre, amikor őt magát is beállítjuk. Utolsó megjegyzésem a rutin vé-

gén lévő sorra vonatkozna. Az állapotsor használatához arra a

környezeti azonosítójával (context id) kell hivatkoznunk. Erre a későbbiekben még szükségünk lesz.

Következőnek az „about” dialógust megjelenítő függvényt hozzuk létre. Ismét csak van nekünk ehhez egy GTK-s rutin. Helyezzük el a jobbra lent látható kódot a MessageBox függvény után.

Mentsük el a programot, és próbáljuk ki. Egy középen megjelenő felugró ablakot kellene látnunk, ami megjeleníti az eddig beállított dolgokat. Ennél van még több olyan attribútum is, amit állítgatni tudunk, (ezeket a <http://www.pygtk.org/docs/pygtk/class-gtkaboutdialog.html> oldalon találhatjuk meg), de én ezeket tartom feltétlenül szükségeseknek.

Mielőtt tovább lépnénk, még meg kell beszélnünk az elkövetkezőket. Az alapvető ötlet az az, hogy amikor a felhasználó rákattint az „Add” gombra, akkor egy felugró ablakban fájlokat tud elhelyezni a lejátszási listában, majd a treeview widget megjeleníti azok adatait. Ez után lehet még több fájlt is elhelyezni, törölni egy bejegyzést vagy mindet, egy bejegyzést fel vagy le, előre vagy hátra mozgatni. Végül beállíthatják a fájlmentési helyét, megadhatunk egy mentési nevet „m3u” kiterjesztés-

```
def SetWidgetReferences(self):
    self.txtFilename = self.wTree.get_widget("txtFilename")
    self.txtPath = self.wTree.get_widget("txtPath")
    self.tbtnAdd = self.wTree.get_widget("tbtnAdd")
    self.tbtnDelete = self.wTree.get_widget("tbtnDelete")
    self.tbtnClearAll = self.wTree.get_widget("tbtnClearAll")
    self.tbtnQuit = self.wTree.get_widget("tbtnQuit")
    self.tbtnAbout = self.wTree.get_widget("tbtnAbout")
    self.tbtnMoveToTop = self.wTree.get_widget("tbtnMoveToTop")
    self.tbtnMoveUp = self.wTree.get_widget("tbtnMoveUp")
    self.tbtnMoveDown = self.wTree.get_widget("tbtnMoveDown")
    self.tbtnMoveToBottom = self.wTree.get_widget("tbtnMoveToBottom")
    self.btnGetFolder = self.wTree.get_widget("btnGetFolder")
    self.btnSavePlaylist = self.wTree.get_widget("btnSavePlaylist")
    self.sbar = self.wTree.get_widget("statusbar1")
    self.context_id = self.sbar.get_context_id("Statusbar")
```

és helyezzük el a hívást közvetlenül az `__init__ self.SetEventDictionary()` sora után:

```
self.SetWidgetReferences()
```

sel és rákattinthatnak a mentés gombra. Annak ellenére, hogy ez elég egyszerűnek tűnik, a háttérben sok dolog megy végbe. Az összes trükk a treeview-t érinti, szóval beszéljük meg azt. Mivel ez elég részletes lesz, ezért nem árt odafigyelni olvasáskor, mivel így sok hibától megkímélhetjük a későbbiekben magunkat.

A treeview lehet egy egyszerű oszlopokba rendezett táblázatra vagy adatbázis kivonatra hasonlító adatszerkezet, vagy lehet egy

```
def ShowAbout(self):
    about = gtk.AboutDialog()
    about.set_program_name("Playlist Maker")
    about.set_version("1.0")
    about.set_copyright("(c) 2011 by Greg Walters")
    about.set_comments("Written for Full Circle Magazine")
    about.set_website("http://thedesignedgeek.com")
    about.run()
    about.destroy()
```

Most kommenteljük ki (vagy akár törölhetjük is) az `on_tbtnAbout_clicked` messagebox hívását, és írjuk be az alábbi, `ShowAbout` függvényt meghívó kódot:

```
def on_tbtnAbout_clicked(self,widget):
    #self.MessageBox("info","Button About was clicked...")
    self.ShowAbout()
```

bonyolultabb, fájl-könyvtár szerkezet szülőkkel és gyermekekkel, ahol a szülő egy könyvtár, a gyermekek pedig a könyvtárban lévő fájlok. De akár még ennél komplexebb is lehet. Ebben a projektben az első megoldást választjuk. A listában három oszlop lesz. Az egyik a zene fájl nevét, a másik a kiterjesztését (mp3, ogg, wav, etc.) és a harmadik az elérési útját fogja tartalmazni. Mindezt egy sztringbe rakva (elérési út, fájl név, kiterjesztés) megkapjuk a létrehozandó playlist egy bejegyzését. Természetesen további oszlopokat is megadhatnánk, de egyelőre csak ezzel a hárommal fogunk foglalkozni.

A treeview egyszerűen egy grafikus tároló, ami tartalmaz és megjelenít egy modellt. A modell egy valós „eszköz”, ami tárolja és manipulálja az adatainkat. A treeview-val két különböző előre definiált modellt használunk, de természetesen sajátot is létre tudunk hozni. Mindazonáltal az esetek 98%-ban ez a két modell pont elég lesz. A két típus a GTKListStore és a GTKTreeStore. Ahogy nevükből is láthatjuk, a ListStore listákhoz, a TreeStore pedig fákhoz használható. A mi programunkban a GTKListStore-t fogjuk használni. Az alapvető lépések a következők:

- Hozzuk létre a TreeView widget

referenciáját.

- Adjuk hozzá az oszlopokat.
- Állítsuk be a használni kívánt renderer típusát.
- Hozzuk létre a ListStore-t.
- Állítsuk be a TreeView model tulajdonságát a modellünkre.
- Töltsük fel adattal.

A harmadik lépés az oszlop által használt renderer típusa, ami az adatok megjelenítéséhez kell, ami egyszerűen csak az adatok kirajzolásáért felelős rutin. Sok fajta, a GTK részét képező cella renderer van, de a legtöbbet használtak a GtkCellRendererText és a GtkCellRendererToggle.

Akkor hát hozzuk is létre a TreeView widgetet beállító függvényt. Hívjuk ezt SetupTreeView-nak. Először definiálunk néhány változót az oszlopokhoz, majd beállítjuk a TreeView referenciáját, hozzáadjuk az oszlopokat, beállítjuk a ListStore-t és végül a modellt. A függvény kódja itt következik, melyet a SetWidgetReferences függvény után helyezhetünk el.

A cFName, a cFType és a cFPath változók az oszlopok számait adják meg. Az sFName, sFType és sFPath a megjelenített oszlopok neveit tartalmazzák. A hetedik változó

```
def SetupTreeView(self):
    self.cFName = 0
    self.cFType = 1
    self.cFPath = 2
    self.sFName = "Filename"
    self.sFType = "Type"
    self.sFPath = "Folder"
    self.treeview = self.wTree.get_widget("treeview1")
    self.AddPlaylistColumn(self.sFName, self.cFName)
    self.AddPlaylistColumn(self.sFType, self.cFType)
    self.AddPlaylistColumn(self.sFPath, self.cFPath)
    self.playlist = gtk.ListStore(str, str, str)
    self.treeview.set_model(self.playlist)
    self.treeview.set_grid_lines(gtk.TREE_VIEW_GRID_LINES_BOTH)
```

beállítja a treeview widget referenciáját a glade fájlban megadottak szerint.

Következőnek meghívunk az oszlopokhoz egy rutint, amit mindjárt megírunk. Ezután definiáljuk a GTKListStore-unkat három szövegmezővel, és végül beállítjuk a modell tulajdonságokat GTKListStore-ra a TreeView widgetnél. Most készítsük el az AddPlaylistColumn függvényt. Helyezzük el a SetupTreeView után.

Mindegyik oszlop ezzel a függvénnyel készül. Átadjuk az oszlop nevét (ami mindegyik oszlop legfelső sorában megjelenik) és a columnID-t. Ebben az esetben, az általunk korábban beállított változók (sFName és cFName) kerülnek ide. Ezután létrehozunk egy oszlopot a Treeview widgetünkben, aminek

megadjuk a címkéjét, a cellák renderelésének módját és az oszlop azonosítóját. Ezután beállítjuk az oszlopot átméretezhetőre (resizable), átadjuk a sort id-t és végül hozzácsapjuk az oszlopot a TreeViewhoz.

Másoljuk be ezt a két függvényt a kódunkba. Én a SetWidgetReferences után raktam őket, de bárhová elhelyezhetők a PlaylistCreator osztályban. A függvény használatához írjuk be a következő sort az __init__ SetWidgetReferences() hívása után.

```
self.SetupTreeView()
```

Mentsük el és futtassuk. Azt fogjuk tapasztalni, hogy három oszlopunk van fejlécekkel a TreeView widgetünkben.

Olyan sok dolog van még hátra. Szükségünk van egy olyan módszerre, ami beolvassa a zene fájlok neveit a felhasználótól és eltárolja a TreeView egyes soraiban. Létre kell hoznunk a Delete, a ClearAll, a mozgató függvények, a mentés rutin és a fájl elérési útjának rutinját, illetve egy-két aprócska dologgal profibbá varázsolhatjuk az alkalmazásunkat. Kezdjük az Add rutinnal, úgyis ez a legelső eszköztár gomb.

Amikor a felhasználó az Add gombra kattint, szeretnénk, ha a szokásos fájl megnyitása ablak jönne fel, ami megengedi a többszörös kijelölést is. Amint a felhasználó befejezte a válogatást, jó lenne ezt az adatot megszerezni és hozzáadni a treeview-hoz. Tehát az első logikus dolog a File Dialog elkészítése lenne. Ismét csak, a GTK rendelkezésünkre bocsát egy hagyományos fájl dialógust. Ezt akár be is írhatnánk az on_tbtnAdd_clicked eseménykezelőbe, de készítsünk egy különálló osztályt ehhez. Ha már úgyis itt vagyunk, megcsinálhatnánk azt is, hogy ne csak az OPEN dialógust, hanem a SELECT dialógust is lekezelje. Mint korábban a MessageBox függvénynél, kimenthetjük ezt is abba a snippet fájlba, amiben mindenféle hasznos rutinokat tárolunk későbbi felhasználásra.

Egy új, FileDialog nevű osztály definiálásával kezdünk, aminek csak egy ShowDialog metódusa lesz. Ez a függvény két paramétert vár, az egyik a „which”, ami megadja, hogy egy fájl megnyitás vagy egy mappa kiválasztás dialógust hozunk létre. A másik a CurrentPath, amely a dialógus nézetének alapértelmezett elérési útja. Hozzuk létre ezt a main kódja előtt, a forrás alapján.

```
class FileDialog:
```

```
    def ShowDialog(self, which, CurrentPath):
```

A kód első része egy IF kell hogy legyen:

```
    if which == 0: # file chooser
        ...
    else:          # folder chooser
        ...
```

Mielőtt továbblépni, nézzük meg, hogy hogyan hívjuk és használjuk a fájl/mappa dialógust. Az ablak szintaxisa a következő:

```
gtk.FileChooserDialog(title, parent, action, buttons, backend)
```

```
def AddPlaylistColumn(self, title, columnId):
    column = gtk.TreeViewColumn(title, gtk.CellRendererText(), text=columnId)
    column.set_resizable(True)
    column.set_sort_column_id(columnId)
    self.treeview.append_column(column)
```

```
dialog = gtk.FileChooserDialog("Select files to add...", None,
    gtk.FILE_CHOOSER_ACTION_OPEN,
    (gtk.STOCK_CANCEL, gtk.RESPONSE_CANCEL,
    gtk.STOCK_OPEN, gtk.RESPONSE_OK))
```

és egy dialog objektumot ad vissza. Az első sorunk (a which == 0 alatt) itt fentebb látható. Ahogy láthatjuk, a title „Select your files to add...” és a szülő None. Egy fájl megnyitás típusú ablakot (action) választottunk egy Cancel és egy Open gombbal, illetve mindkettő rendelkezik alapértelmezett ikonnal. A visszatérési érték a felhasználó döntésétől függően gtk.RESPONSE_CANCEL vagy gtk.RESPONSE_OK lesz. A mappa kiválasztó hívása az Else ágban hasonló.

Gyakorlatilag az egyetlen dolog ami változott az előzőekhez képest, az a title (lásd lent) és az action. Az osztályunk kódja a következő oldalon található.

```
dialog = gtk.FileChooserDialog("Select Save Folder..", None,
    gtk.FILE_CHOOSER_ACTION_SELECT_FOLDER,
    (gtk.STOCK_CANCEL, gtk.RESPONSE_CANCEL,
    gtk.STOCK_OPEN, gtk.RESPONSE_OK))
```

Ezek beállítják az alapértelmezett választ OK-ra és engedélyezik a többszörös kijelölést ahhoz, hogy a felhasználó (vajon mit csinálhasson?) több fájl tudjon hozzáadni. Ha ezt nem tennénk meg, akkor – mivel a set_select_multiple alapból False-ra van állítva – a dialógus egyszerre csak egy fájl kijelölését engedélyezné. A következő sorok beállítják az aktuális elérési utat és megjelenítik a tényleges ablakot. Mielőtt begépelni a kódot, hadd magyarázzam el, hogy miért foglalkozunk az elérési úttal. Minden alkalommal amikor egy dialógust jelenítünk meg a path beállítása nélkül, az alapértelmezett mappa az alkalmazásunké lesz. Tegyük fel, hogy a felhasználó zenéi a /media/music_

files/ könyvtárban vannak és rendre műfajok, szerzők, illetve albumok szerint vannak osztályozva. Továbbá feltételezzük, hogy a felhasználó a /home/user2/playlistmaker-be telepítette programunkat. Minden alkalommal, amikor megjelenítjük az ablakot, az a /home/user2/playlistmaker-t nyitná meg, és így elég gyorsan frusztrálóvá válna. Jobb megoldás lenne, ha az utolsó megjelenített mappából indulnánk ki. Értjük már? Rendben. Akkor itt van a következő pársor.

Itt leellenőrizzük a visszaküldött válaszokat. Ha a felhasználó az „Open” gombra – ami egy `gtk.RESPONSE_OK`-ot küld – kattintott, akkor a kiválasztott fájl vagy fájlok elérési útjait kapjuk meg, majd beállítjuk az aktuális mappát, töröljük a dialógust és visszaadjuk az adatokat a hívó rutinnak. A másik esetben, ha a felhasználó a „Cancel” gombra kattint, akkor egyszerűen csak töröljük a dialógust. A kiíratást csak azért raktam bele, hogy lássuk a gombnyomás tényleg működik. Benne is hagyhatjátok vagy ki is vehetitek. Vegyük észre, hogy amikor az Open gombos részből visszatérünk, két értéket adunk vissza. Az egyik „fileselection”, ami a kiválasztott fájlok listája, a másik `CurrentPath`.

Ahhoz, hogy a rutin csináljon valamit, helyezzük el a következő sort az `on_tbtnAdd_click` rutin után:

```
fd =
FileDialog()

selectedfiles,
self.CurrentPath =
fd.ShowDialog(
0,self.CurrentPath)
```

Itt megszeresszük a két visszatérési értéket. Egyelőre csak helyezzük el a következő kódot, ami megmutatja, hogy a visszaadott információ hogyan néz ki.

```
for f in selectedfiles:

    print "User selected %s"
    % f

print "Current path is %s" %
self.CurrentPath
```

Amikor futtatjuk a programot, kattintsunk az „Add” gombra. Így láthatjuk a fájl dialógust. Most navigáljunk el valahova, ahol van néhány fájlunk és válasszuk ki őket. A

```
class FileDialog:
    def ShowDialog(self,which,CurrentPath):
        if which == 0: #file chooser
            #gtk.FileChooserDialog(title,parent,action,buttons,backend)
            dialog = gtk.FileChooserDialog("Select files to add...",None,

                                           gtk.FILE_CHOOSER_ACTION_OPEN,
                                           (gtk.STOCK_CANCEL, gtk.RESPONSE_CANCEL,
                                            gtk.STOCK_OPEN, gtk.RESPONSE_OK))

        else:          #folder chooser
            dialog = gtk.FileChooserDialog("Select Save Folder..",None,

                                           gtk.FILE_CHOOSER_ACTION_SELECT_FOLDER,
                                           (gtk.STOCK_CANCEL, gtk.RESPONSE_CANCEL,
                                            gtk.STOCK_OPEN, gtk.RESPONSE_OK))

        dialog.set_default_response(gtk.RESPONSE_OK)
        dialog.set_select_multiple(True)
```

A következő két sor az IF/ELSE utasításokon kívül lesz:

Ctrl gomb lenyomásával és kattintással egyenként több fájlt is kijelölhetünk, a Shifttel pedig sok egymás után lévőt. Kattintsunk az „Open” gombra és figyeljük meg a

terminálban az eredményt. Felhívnam a figyelmet arra, hogy ha most kattintanánk a „Cancel” gombra, akkor egy hiba üzenetet kapnánk. Ez azért van, mert a kód

```
if CurrentPath != "":
    dialog.set_current_folder(CurrentPath)
response = dialog.run()
```

Ez után le kell kezelnünk az ablak választát.

```
if response == gtk.RESPONSE_OK:
    fileselection = dialog.get_filenames()
    CurrentPath = dialog.get_current_folder()
    dialog.destroy()
    return (fileselection,CurrentPath)
elif response == gtk.RESPONSE_CANCEL:
    print 'Closed, no files selected'
    dialog.destroy()
```

feltételezi, hogy nincsenek fájlok kijelölve. Ez egyelőre még ne nagyon izgasson minket – egy kicsit később foglalkozunk majd vele. Csak kíváncsi voltam arra, hogy mit kapunk az „Open” megnyomása-kor. Még egy dolog, amit meg kell csinálnunk, az egy szűrő hozzáadása a fájl-megnyitó ablakhoz. Mivel azt várjuk, hogy a felhasználó zene fájlokat fog megnyitni, ezért (1) meg kell neki adni a zene fájlok, és (2) minden fájl megjelenítésének lehetőségét is (ha szükség lenne rá). Ezt az ablak filefilter tulajdonságával tehetjük meg. A következő kódnak a `which == 0` részen belül, közvetlenül a dialógus beállítása után kell kerülnie.

```
filter = gtk.FileFilter()
filter.set_name("Music Files")
filter.add_pattern("*.mp3")
filter.add_pattern("*.ogg")
filter.add_pattern("*.wav")
dialog.add_filter(filter)
filter = gtk.FileFilter()
filter.set_name("All files")

filter.add_pattern("")
dialog.add_filter(filter)
```

Két csoportot állítunk be: egyiket a zenének (`filter.set_name("Music Files")`), a másikat az összes fájlnek. Mintákat használunk a fájl típusok meghatározásához. Három ilyen mintánk van, de nyugodtan tehetünk

még hozzájuk, vagy vehetünk el belőlük. Azért raktam a zene szűrőt előre, mert azt feltételezzük, hogy főleg ezeket akarja a felhasználó használni. Tehát a lépések a következők:

- Egy szűrő változó definiálása.
- Név beállítása.
- Minta hozzáadása.
- A szűrő dialógushoz kötése.

Annyi szűrőnk lehet, amennyit csak akarunk. Vegyük észre azt is, hogy ha egyszer egy filtert az ablakhoz rendeltünk, akkor újrahasznosíthatjuk a változóját.

Még az `on_tbtnAdd_clicked` rutinnál, kommenteljük ki az utolsó sorainkat, és írjuk be a következőt.

```
self.AddFilesToTreeview(selectedfiles)
```

tehát a rutinunk a következő képpen néz ki (lásd jobbra):

Amikor megszerezzük a dialógus választ, a kiválasztott fájlok listáját ennek a rutinnak adjuk át. Itt beállítjuk a számlálót (hány fájlt helyezünk el), majd parzoljuk a listát. Emlékeztünk, hogy minden bejegyzés egy teljes fájlnevet tartalmaz elérési úttal és kiterjesztéssel. Ennek alapján fel szeretnénk darabolni elérési útra, névre és kiterjesztésre. Először megkeressük legutolsó pontot és feltételezzük, hogy itt kezdődik a kiterjesztés és a pozícióját az `exStarthoz` rendeljük. Ez-

```
def on_tbtnAdd_clicked(self,widget):
    fd = FileDialog()
    selectedfiles,self.CurrentPath =
fd.ShowDialog(0,self.CurrentPath)
    self.AddFilesToTreeview(selectedfiles)
```

Most létre kell hoznunk egy olyan függvényt, ahova a hívást rakjuk. Helyezzük ezt a függvényt az `on_btnSavePlaylist_clicked` rutin után.

```
def AddFilesToTreeview(self,FileList):
    counter = 0
    for f in FileList:
        extStart = f.rfind(".")
        fnameStart = f.rfind("/")
        extension = f[extStart+1:]
        fname = f[fnameStart+1:extStart]
        fpath = f[:fnameStart]
        data = [fname,extension,fpath]
        self.playlist.append(data)
        counter += 1
    self.RowCount += counter
    self.sbar.push(self.context_id,"%s files added for
a total of %d" % (counter,self.RowCount))
```

után megkeressük a legutolsó `/`-t a fájlnev kezdetének meghatározásához. Ezt követően feltördeljük a sztringet kiterjesztés, fájlnev és elérési útra. A kapott értékeket a `"data"` listába helyezzük el és hozzáillesztjük a lejátszási lista `ListStore`-jához. Mivel készen vagyunk, növeljük a számlálót. Végül a `RowCount` változót is inkrementáljuk – ami a `ListStore`-ban lévő sorok számát tárolja – és kiíratunk egy üzenetet az állapotsorra.

Most már futtathatjuk az alkalmazást, és megnézhetjük a `TreeVi-`

`ew` tartalmát.

Mint mindig, a teljes kód megtalálható a <http://paste-bin.com/JtrhuE71> címen.

Következő alkalommal véglegesítjük az alkalmazásunkat, befejezzük a hiányzó rutinokat, stb.



Greg Walters a RainyDay Solutions LLC tulajdonosa, amely egy tanácsadó cég a coloradói Aurorában. 1972 óta foglalkozik programozással. Szeret főzni, túrázni, zenét hallgatni, valamint a szabadidejét családjával tölteni.



Most befejezzük a lejátszáslista-készítő programunkat. Legutóbb már a nagyját megcsináltuk, de egy pár dolog befejezetlenül maradt. A playlistet nem tudjuk elmenteni, a mozgató funkciók sincsenek készen, nem tudjuk kiválasztani a file rögzítési útvonalát, meg hasonlókat. Egy pár dolgot viszont még meg kell csinálnunk, mielőtt nekiállnánk kódokat írni. Először is kell találnunk egy képet az alkalmazáshoz, ami az “about” dobozban fog szerepelni. Erre akkor is szükségünk lesz, amikor az alkalmazást minimális méretre csökkentjük. Körülnézhettünk az /usr/share/icons mappában, de akár a webről is lementhetünk egy nekünk tetsző képet, vagy akár mi magunk is csinálhatunk egyet. Ha megvan a kép, tegyük be a code mappába. Tegyük mellé a .glade file-t, továbbá a forráskódot, amit múlt hónapban készítettünk. Nevezzük el logo.png-nek. Most nyissuk meg a .glade file-t (amit szintén a múlt hónapban készítettünk) és eszközöljünk benne pár változtatást.

Először a MainWindow-ban klikkeljünk a General fülre és gördítsünk le addig, amíg meg nem találjuk az Icon-t. A böngésző funkcióval keressük meg a korábban ki-

választott ikonunkat és válasszuk ki. A szövegdobozban a “logo.png”-nek kell szerepelnie. Utána a hierarchia dobozban válasszuk a tree-view1-et, klikkeljünk a “signal” fülre és a GtkTreeView | cursor-changed résznél adjunk hozzá egy handler-t az “on_treeview1_cursor_changed”-hez. Ne felejtsük el (a múlt hónapban említettem már), hogy ha azt akarjuk, hogy működjön a változtatás, ezt is végre kell hajtunk. Végül ismét a hierarchia dobozban válasszuk ki a txtFilename-t, és klikkeljünk a “signal” fülre. Gördítsünk le, míg meg nem találjuk a “GtkWidget”-et; gördítsünk tovább egészen a “key-press-event”-ig. Adjunk hozzá egy handler-t az “on_txtfilename_key_press_event”-hez. Mentsük el az egészet és zárjuk be.

Most értünk az egésznek a befejező részéhez. Ott vagyunk, ahol az előző alkalommal a kód használatával dolgoztunk.

Először is módosítani kell a kó-

```
elif response == gtk.RESPONSE_CANCEL:
    print 'Closed, no files selected'
    dialog.destroy()
```

Láthatjuk, hogy semmit nem helyeztünk vissza. Ez okozta ezt a problémát. Ahhoz, hogy kijavítsuk, írjuk az alábbi kódot a dialog.destroy() sor után:

```
Return ([], "")
```

Ezzel el fog tűnni a hibaüzenet. Most adjuk hozzá a szövegdoboz-handler-t, amit a glade-ben készítettünk. Adjuk a szótárhoz az alábbi sort:

```
"on_txtFilename_key_press_event": self.txtFilenameKeyPress,
```

Tudjuk, hogy ez lesz a keypress event működtetője. Most megcsináljuk magát a funkciót:

```
def txtFilenameKeyPress(self, widget, data):
    if data.keyval == 65293: # The value of the return key
        self.SavePlaylist()
```

dot a FileDialog-ban. Ne felejtsük el, hogy ha a Cancel-re klikkelünk, rögtön hibaüzenetet kapunk. Ezt most fogjuk kijavítani. A rutinfelet adat végén az alábbi kódot kapjuk:

Valószínűleg már sejthető, hogy ez fogja érzékelni a lenyomott billentyűk értékét, amikor a txtFilename box-ban dolgozunk, majd összehasonlítja a billentyűk értékét a 65293-as értékkel, mely az Enter

billentyű kódja. Ha az értékek megfelelőek, akkor magától létrejön a SavePlaylist funkció. Még csak rá sem kell klikkelnünk a gombra.

Most pedig jöjjön egy újabb kód. Kezdjük a ClearAll gombbal az eszköztáron. Ha ráklikkelünk, törlődik a ListStore és a gyökérkönyvtár-megjelenítés. Ez egy egysoros parancs, amit az "on_tbtnClearAll_clicked" rutinba tehetünk.

```
def
on_tbtnClearAll_clicked(self,
widget):
```

```
self.playList.clear()
```

Ezzel egyszerűen kiadtuk a playList ListStore-nak a parancsot a törlésre. Ez igazán nem volt nehéz. Most jön az eszköztáron a Delete gomb. Ez már nehezebb lesz, de ha belemélyedünk, érthetőbbé válik.

Először is azt kell megnéznünk, hogyan tudunk kijelölni a gyökérkönyvtár-widget-ben és a ListStore-ban. Ez bonyolult, ezért lassan fogunk haladni. Ahhoz, hogy a ListStore-ból visszahozzuk az adatokat, szükségünk van egy gtk.TreeSelection-re. Ez egy segédobjektum, ami a gyökérkönyvtár-nézetben segíti a kijelölést. Ennek a segítségével fogjuk visszaszerezni a modelltipust, továbbá egy iterátort, ami a kijelölt sorokat tartalmazza.

Most nyilván arra gondol a kedves Olvasó, hogy "Mi a keserves bú-

bánat az az iterátor?". Pedig már korábban használtuk, legfeljebb nem tudtuk, hogy az az. Gondoljunk az AddFilesToTreeview funkciót követő kódsorra, amit a múlt hónapban csináltunk.

Nézzük meg a "for" után következő részt. Ott használtunk egy iterátort hogy a FileList nevű listán végigmenjünk. Ebben az esetben az iterátor lényegében egyenként végigmegy az összes részen, és minden egyes részt újra visszahelyez. Most is egy iterátort fogunk csinálni és beleírjuk a gyökérkönyvtár-nézetben kiválasztott sorokat. Majd ezt fogjuk listaként használni. Tehát az on_tbtnDelete_clicked kódja a következő:

Az első sor hozza létre a TreeSelection objektumot. Ezzel jelöljük ki a sorokat (ami most csak egy lesz, mert a jelenlegi példát nem állítottuk be többszörös kijelölésre). betöltjük az iters nevű listára, majd járunk el úgy, mint a .clear metódusnál. A változtatható RowCount-ot is csökkentjük, majd a file-ok számát tegyük ki a status bar-ra.

Most pedig, mielőtt a mozgató műveletekhez érnénk, foglalkozunk kicsit a save-file-path-al. Ismét a FileDialog-ot fogjuk használni, mint korábban. Készítünk egy kó-

```
def AddFilesToTreeview(self,FileList):
    counter = 0
    for f in FileList:
        extStart = f.rfind(".")
        fnameStart = f.rfind("/")
        extension = f[extStart+1:]
        fname = f[fnameStart+1:extStart]
        fpath = f[:fnameStart]
        data = [fname,extension,fpath]
        self.playList.append(data)
        counter += 1
```

```
def on_tbtnDelete_clicked(self,widget):
    sel = self.treeview.get_selection()
    (model,rows) = sel.get_selected_rows()
    iters=[]
    for row in rows:
        iters.append(self.playList.get_iter(row))
    for i in iters:
        if i is not None:
            self.playList.remove(i)
            self.RowCount -= 1
    self.sbar.push(self.context_id,"%d files in list." %
(self.RowCount))
```

```
def on_btnGetFolder_clicked(self,widget):
    fd = FileDialog()
    filepath,self.CurrentPath = fd.ShowDialog(1,self.CurrentPath)
    self.txtPath.set_text(filepath[0])
```

dot az "on_btnGetFolder_clicked"-hez:

A korábbiakhoz képest az egyetlen különbség a kód utolsó sora. A FileDialog-ból betesszük az útvonalat a szövegdobozba, amit már korábban a set_text-tel beállítottunk.

Ne feledjük, hogy a kapott adat benne van a listában, még akkor is, ha csak egyetlen sort jelöltünk ki. Ezért használjuk a "filepath[0]"-t.

Most jön a fájlmentés művelete. Ezt nyugodtan megcsinálhatjuk még a mozgató műveletek előtt. Csinálunk egy SavePlaylist funkciót. Ehhez először meg kell néznünk, hogy van-e valami a txtPath szövegdobozban. Utána nézzük meg, hogy szerepel-e valamilyen filenév a txtFilename dobozban. Mindkét esetben a szövegdozoz.get_text() módját használjuk.

Most hogy már tudjuk, hogy van egy útvonalunk (fp) és filenevünk (fn), megnyithatjuk a file-t, kinyomtathatjuk az m3u fejléct és továbbléphetünk a playList-re. Az útvonalat elmentettük a 2. oszlopba, a file neve a 0. oszlopban van, a kiterjesztés az elsőben. Most egyszerűen csak csinálunk hozzá egy string-et, beleírjuk a file-ba és bezzárjuk.

Most nekiláthatunk a mozgató funkcióknak. Kezdjük a Move To Top-pal. Ugyanúgy csináljuk, mint amikor a törlés funkciót készítettük: kijelölés, majd sorok kijelölése. Utána át kell lépnünk a sorokon, hogy két változatot kapjunk. Path1-nek és path2-nek fogom most őket nevezni. A path2 ebben az esetben 0-ra lesz állítva, ez lesz a "cél" sor. A path1 a felhasználó

```
def SavePlaylist(self):
    fp = self.txtPath.get_text()      # Get the filepath from the text box
    fn = self.txtFilename.get_text()  # Get the filename from the filename text box
```

Most ellenőrizzük az értékeket...

```
if fp == "":
    self.MessageBox("error", "Please provide a filepath for the playlist.")
elif fn == "":
    self.MessageBox("error", "Please provide a filename for the playlist file.")
else:
    # Otherwise we are good to go.
```

```
plfile = open(fp + "/" + fn, "w")  # Open the file
plfile.writelines('#EXTM3U\n')     # Print the M3U Header
for row in self.playList:
    plfile.writelines("%s/%s.%s\n" % (row[2], row[0], row[1])) #Write the line data
plfile.close                       # Finally close the file
```

Végül egy felugró ablakban informáljuk a felhasználót, hogy a file-t elmentette.

```
self.MessageBox("info", "Playlist file saved!")
```

Most pedig egy hívókédot illesztünk a rutinba az "on_btnSavePlaylist_clicked" esemény handlerében.

```
def on_btnSavePlaylist_clicked(self, widget):
    self.SavePlaylist()
```

Mentsük el a kódot és teszteljük. A lejátszási listában immár tudunk menteni, és az egésznek kb. úgy kell kinéznie, ahogy azt a múlt hónapban mutattam.

által kijelölt sor. Végül a model.move_before() metódussal tegyük a kiválasztott sort a 0. sorba, így gyakorlatilag mindent eggyel lejjebb toltunk. A kódot közvetlenül az "on_tbtnMoveToTop_clicked" rutinba írjuk.

```
def on_tbtnMoveToTop_clicked(self, widget):
    sel = self.treeview.get_selection()
    (model, rows) = sel.get_selected_rows()
    for path1 in rows:
        path2 = 0
        iter1 = model.get_iter(path1)
        iter2 = model.get_iter(path2)
        model.move_before(iter1, iter2)
```

A MoveToBottom funkcióval szinte pontosan ugyanúgy járunk el, mint a MoveToTop esetében, a különbség az, hogy a model.move_before() helyett a model.move_after() parancsot használjuk, és a path2-t nem nullára, hanem self.RowCount-1-re állítjuk. Mindjárt meg fogjuk érteni, miért van változó RowCount-unk. Tudjuk, hogy a számok nulla alapúak, ezért kell RowCount-1-et használnunk.

Nézzük, hogyan kell a MoveUp rutint csinálni. Ez is eléggé hasonló az előző kettőre. Most viszont a path1-et vesszük, ami a kijelölt sor, a path2-nek meg jelöljük ki a -1-ik sort. Aztán AMENNYIBEN a path2 (a célsor) nagyobb vagy egyenlő nullával, akkor a model.swap() metódust használjuk.

Ugyanígy megy a MoveDown funkció is. Most viszont meg kell nézni, hogy a path2 KISEBB vagy egyenlő-e a self.RowCount-1 értékével.

Most pedig eszközölünk pár változtatást a lejátszási listánkon. Múlt hónapban mutattam a playlist file alapformátumát.

Viszont azt is mondtam, hogy van egy kiterjesztett formátuma is. Ebben van egy extra sor, amit hozzá lehet adni a file-hoz (ez az adott zeneszámról tartalmaz extra információt). A formátuma így néz ki:

```
#EXTINF:[Length of song in seconds],[Artist Name] – [Song Title]
```

Lehet, hogy az Olvasó kíváncsi, miért tartalmazzák a műveletek már az elejétől fogva a mutagén könyvtárat, holott még sosem használtuk őket. Hát, most fogjuk. Mint tudjuk, a mutagén könyvtár az ID3 tag eléréséhez szükséges (az mp3 file-okon belül). A Full Circle magazin 35. számában van erről egy hosszabb értekezés (a sorozat 9. részében). Most egy olyan funkciót fogunk készíteni, ami felismeri az mp3 file-t, a dal címét, az előadó nevét és a dal hosszát (mp-ekben). Erre a három dologra van szükségünk a kiterjesztett információs vonalhoz. Tegyük a funkciót a Show-About funkció után a PlaylistCreatorban.

```
#EXTM3U
```

```
Adult Contemporary/Chris Rea/Collection/02 - On The Beach.mp3
```

```
Adult Contemporary/Chris Rea/Collection/07 - Fool (If You Think It's Over).mp3
```

```
Adult Contemporary/Chris Rea/Collection/11 - Looking For The Summer.mp3
```

```
def on_tbtnMoveToBottom_clicked(self,widget):
    sel = self.treeview.get_selection()
    (model,rows) = sel.get_selected_rows()
    for path1 in rows:
        path2 = self.RowCount-1
        iter1=model.get_iter(path1)
        iter2 = model.get_iter(path2)
        model.move_after(iter1,iter2)
```

```
def on_tbtnMoveUp_clicked(self,widget):
    sel = self.treeview.get_selection()
    (model,rows) = sel.get_selected_rows()
    for path1 in rows:
        path2 = (path1[0]-1,)
        if path2[0] >= 0:
            iter1=model.get_iter(path1)
            iter2 = model.get_iter(path2)
            model.swap(iter1,iter2)
```

```
def on_tbtnMoveDown_clicked(self,widget):
    sel = self.treeview.get_selection()
    (model,rows) = sel.get_selected_rows()
    for path1 in rows:
        path2 = (path1[0]+1,)
        iter1=model.get_iter(path1)
        if path2[0] <= self.RowCount-1:
            iter2 = model.get_iter(path2)
            model.swap(iter1,iter2)
```

Ismétlés a tudás anyja: menjünk végig a kódon. Először töröljük ki a három visszatérő változót, így ha bármi történne, ezek üresen maradnak. Most belehelyezzük a meg-

vizsgálandó mp3-file nevét. Aztán behúzzuk a kulcsokat egy (igen, eltalálta a kedves Olvasó) iterátorba és végigmegyünk rajta, mégpedig úgy, hogy két specifikus tag-et ke-

resünk. Ezek pedig a TPE1 (az előadó neve) és a TIT2 (a dal címe). Ha a kulcs nem létezik, akkor kapunk egy hibaüzenetet, ezért minden hívókódot beburkolunk egy "try|except" utasítással. Majd kijelöljük a dal hosszát az audio.info.length attribútumban és nyomunk az egész kócerájra egy entert.

Most pedig a SavePlayList funkciót módosítjuk, hogy támogassa a kiterjedt információs vonalat. Ha ott vagyunk, ellenőrizzük, hogy létezik-e a filenév. Ha igen, jelöljük meg egy zászlóval a felhasználót és lépünk ki a feladatból. A felhasználó számára még úgy tehetjük egyszerűbbé a dolgokat (miután semmi más fíletípus nem támogatott az esetünkben), hogy az m3u kiterjesztést automatikusan az útvonalhoz és a filenévhez csatoljuk, ha az nem létezik. Először is írjunk egy importáló sort a kód elejére, hogy mozgathassuk az os.path-et a rendszer és a mutagén import között.

Akár az AddFilesToTreeview funkcionál, itt is az "rfind" metódust használjuk a legutolsó pont ('.') filenév-beli (fn) pozíciójának megtalálásához. Ha nincs, akkor az enter billentyű értékét állítsuk -1-re. Ha megnéztük, hogy az enter

értéke tényleg -1, csatoljuk a kiterjesztést és tegyük vissza a filenevet a szövegdobozba.

```
if os.path.exists(fp +
"/" +
fn):

self.MessageBox("error", "The file already exists. Please select another.")
```

Most pedig az egész funkciót összecsomagoljuk egy IF|ELSE záradékkal, így ha a file már létezik, egyszerűen kiesünk a műveletből.

```
def GetMP3Info(self, filename):
    artist = ''
    title = ''
    songlength = 0
    audio = MP3(filename)
    keys = audio.keys()
    for key in keys:
        try:
            if key == "TPE1":          # Artist
                artist = audio.get(key)
        except:
            artist = ''
        try:
            if key == "TIT2":          # Song Title
                title = audio.get(key)
        except:
            title = ''
    songlength = audio.info.length    # Audio Length
    return (artist, title, songlength)
```

```
import os.path
```

Utána jelöljük ki a meglévő SavePlaylist funkciót, mert azt most eltávolítjuk.

```
def SavePlaylist(self):
    fp = self.txtPath.get_text()      # Get the file path from the text box
    fn = self.txtFilename.get_text()  # Get the filename from the text box
    if fp == "": # IF filepath is blank...
        self.MessageBox("error", "Please provide a filepath for the playlist.")
    elif fn == "": # IF filename is blank...
        self.MessageBox("error", "Please provide a filename for the playlist file.")
    else: # Otherwise
```

Idáig minden olyan, mint korábban. Most kezdődnek azonban a változások.

```
extStart = fn.rfind(".") # Find the extension start position
if extStart == -1:
    fn += '.m3u' #append the extension if there isn't one.
    self.txtFilename.set_text(fn) #replace the filename in the text box
```

Az ellenőrzéshez az `os.path.exists` (filename)-t használjuk.

A kód többi része nagyrészt ugyanaz, mint korábban, de azért nézzük csak át.

A 2. sor nyitja a file-t, amit meg fogunk írni. A 3. sor illeszti be az `m3u` fejléct. A 4. sor kalauzol végig a `playList` `ListStore`-ján. Az 5. sor alkotja meg a file nevét a `ListStore` 3 oszlopa alapján. A 6. sor kéri a `GetMP3Info`-t és tárolja az értékeket a változóiban. A 7. sor ellenőrzi, hogy van-e érték a három változóban. Ha igen, a 8. sorba írjuk a kiterjesztés információs sorát. A 9. sor a filenevet írja, ahogy korábban. A 10. sor végre-valahára bezárra a file-t, a 11. pedig küldi a `popup`-ablakot, amely tudatja az

```
else:
    plfile = open(fp + "/" + fn,"w") # Open the file
    plfile.writelines('#EXTM3U\n')   #Print the M3U header
    for row in self.playList:
        fname = "%s/%s.%s" % (row[2],row[0],row[1])
        artist,title,songlength = self.GetMP3Info(fname)
        if songlength > 0 and (artist != '' and title != ''):
            plfile.writelines("#EXTINF:%d,%s - %s\n" % (songlength,artist,title))
            plfile.writelines("%s\n" % fname)
    plfile.close # Finally Close the file
    self.MessageBox("info","Playlist file saved!")
```

userrel, hogy készen van a procedúrával.

Mentsük el a kódot és teszteljük le.

Egyetlen dolog van még, amit hozzá kell adnunk, mégpedig a vezérlőelemek buboréksúgói, melyek akkor jönnek elő, ha a felhasználó az egérrel rájuk navigál. Ettől az

egész annyira profi kinézetű lesz. Csináljuk is meg a funkciót hozzá.

Azt a widget-referenciát használjuk hozzá, amit korábban beállítottunk, majd beírjuk a buboréksúgó szövegét a (a kedves Olvasó már nyilván tudja) `set_tooltip_text` attribútummal. Már csak hívó kód hiányzik hozzá. Menjünk vissza az `__init__` routine-ba a `self.SetWidgetReferences` sorba és

írjuk be:

```
self.SetupToolTips()
```

Végül, de természetesen nem utolsósorban a korábban kiválasztott logónkat be akarjuk helyezni az `About` dobozba. Mint mindenhez, ehhez is van attribútum. A `ShowAbout`-hoz adjuk hozzá ezt a sort:

```
def SetupToolTips(self):
    self.tbtnAdd.set_tooltip_text("Add a file or files to the playlist.")
    self.tbtnAbout.set_tooltip_text("Display the About Information.")
    self.tbtnDelete.set_tooltip_text("Delete selected entry from the list.")
    self.tbtnClearAll.set_tooltip_text("Remove all entries from the list.")
    self.tbtnQuit.set_tooltip_text("Quit this program.")
    self.tbtnMoveToTop.set_tooltip_text("Move the selected entry to the top of the list.")
    self.tbtnMoveUp.set_tooltip_text("Move the selected entry up in the list.")
    self.tbtnMoveDown.set_tooltip_text("Move the selected entry down in the list.")
    self.tbtnMoveToBottom.set_tooltip_text("Move the selected entry to the bottom of the list.")
    self.btnGetFolder.set_tooltip_text("Select the folder that the playlist will be saved to.")
    self.btnSavePlaylist.set_tooltip_text("Save the playlist.")
    self.txtFilename.set_tooltip_text("Enter the filename to be saved here. The extension '.m3u' will be added for you if you don't include it.")
```

```
about.set_logo(gtk.gdk.pix-  
buf_new_from_file("lo-  
go.png"))
```

És ennyi. Most már van egy teljes mértékben működő programunk, ami jól is néz ki és fantasztikusan is működik, ahogy lejátszási listát készít a zenefájleja-inkból.

A teljes forráskód (a .glade-fájlal együtt, amit múlt hónapban csináltunk) a pastebin-en található: <http://pastebin.com/tQJizcwT>

Remélem, kellemesen töltötték az időt és örömmel használják majd az újonnan tanultakat.



Greg Walters a RainyDay Solutions LLC nevű tanácsadó cég tulajdonosa. Aurorában, Colorado államban él és 1972 óta foglalkozik programozással. Hobbija a főzés, a kirándulás, a zenehallgatás és a családja. Weboldala: www.thedesignedgeek.com.

EXTRA! EXTRA! Olvasd el egyben!



Inkscape Speciális kiadás 1.

Ez egy különleges kiadása a Full Circle magazin 60–67. kiadásában megjelent részeknek.

http://fullcircle.hu/wp-content/plugins/download-monitor/download.php?id=issueIS01_hu.pdf

A Full Circle Speciális Kiadása megjelent!



LibreOffice Speciális kiadás 1.

A LibreOffice cikksorozat, a magazin 46–52. számaiban megjelent részei.

http://fullcircle.hu/wp-content/plugins/download-monitor/download.php?id=issueL001_hu.pdf



WOW! Nehéz elhinni, hogy a 24. cikkél tartunk. Már két éve tanuljuk a Pythont és egy elég hosszú út áll mögöttünk.

Ez alkalommal két témával fogunk foglalkozni. Az első a nyomtatással való nyomtatás, a második az RTF (Rich Text Format) fájlok készítése.

Általános nyomtatás Linux alatt

Nos, akkor kezdjünk a nyomtatással. Az ötlet, hogy ezzel a témával foglalkozzunk Gord Campbell leveléből jött. Valójában Linux alatt egész egyszerű a legtöbb nyomtatási feladat elintézése, mi több könnyebb mint azon a másik operációs rendszeren aminek neve „WIN-nel” kezdődik – és ezzel az OS-sel nem is foglalkozunk.

Mindaddig amíg csak egyszerű szöveget akarunk nyomtatni félkövérek, dőlték és betűtípusok, stb. nélkül, addig könnyű. Itt van egy egyszerű alkalmazás, ami közvetle-

nül a nyomtatóra ír...

```
import os

pr = os.popen('lpr', 'w')

pr.write('print test from linux via python\n')

pr.write('Print finished\n')

pr.close()
```

Az egész elég egyértelmű, ha egy kicsit elengedjük a fantáziánkat. A fenti kódban az „lpr” a nyomtató spoolja. Az egyetlen előfeltétel, hogy legyen egy beállított és futó ‘lpd’. Több mint valószínű, hogy amikor egy nyomtatót használunk Ubuntu alatt, akkor ezt már megtették helyetted. Az ‘lpd’ többnyire egy „bűvös átalakító”, ami automatikusan átkonvertálja a különböző dokumentumokat a nyomtató által értelmezett adattá. Mi az ‘lpr’ eszköze/objektumra fogunk nyomtatni. Gondoljunk erre úgy mint egy fájlra. Megnyitjuk a fájlt. Ehhez be kell importálnunk az os-t. A második sorban megnyitjuk az ‘lpr-t’ írásra – közben hozzárendeljük az objektumot a ‘pr’ változóhoz. Ezután egy ‘pr.write’ hívást

hajtunk végre a kinyomtatandó dologgal. Végül (5. sor) bezárjuk a fájlt, így küldve el az adatokat a nyomtatónak.

Egy szöveges fájlt is létre tudunk hozni, majd a következőképpen kinyomtatni...

```
import os

filename = 'dummy.file'

os.system('lpr %s' % filename)
```

Ebben az esetben is az lpr objektumot használjuk, de az ‘os.system’ utasítás úgy tesz, mintha a terminálba írtunk volna be egy parancsot.

Játszozzatok el vele egy kicsit.

PyRTF

Most foglalkozzunk az RTF fájlokkal. Az RTF formátumot (olyan ez, mint a PIN number, ahol a PIN a Personal Identification Number rövidítése, így a jelölés valójában a Personal-Identification-Number

WOW! Nehéz elhinni, hogy a 24. cikkél tartunk. Már két éve tanuljuk a Pythont és egy elég hosszú út áll mögöttünk.

Number-t takarja...) eredetileg a Microsoft készítette 1987-ben, a szintaxisára pedig nagy hatással volt a TeX betűszedő nyelv. A PyRTF egy csodálatos könyvtár, amely megkönnyíti az RTF fájlok írását. Előre ki kell találnod, hogy a fájl megjelenését hogyan képezed el, de a végeredmény bőségesen kompenzálja majd ezt a plusz munkát.

Először is, le kell töltened a PyRTF csomagot. Ehhez menj a <http://pyrtf.sourceforge.net> oldalra és töltsd le a PyRTF-0.45.tar.gz csomagot. Mentsd el valahová és a kicsomagoláshoz használd az archívum kezelőt. Nyiss egy terminált és lépj be abba a könyvtárba, ahová a kicsomagolt fájlokat tetted. A csomag telepítéséhez írd

be ezt: „sudo python setup.py install”. Találsz majd itt egy mappát példafájlokkal, ez még a későbbiekben hasznos lehet, ha az egyszerű szöveg nyomtatásánál valami bonyolultabbat szeretnél csinálni.

Na, szóval akkor kezdjünk is hozzá a már jól megszokott módon, egy egyszerű kis program csonkkal.

Mielőtt tovább mennénk, beszéljünk meg gyorsan mi történt. A második sor importálja a PyRTF könyvtárat. Felhívnam rá a figyelmet, hogy más import formátumot használunk, mint általában. Ez a megoldás a könyvtárból mindent importál.

A fő munka-rutinunk a MakeExample. A program vázát már elhelyeztük. Az OpenFile rutin paraméterként megkapja a létrehozandó fájl nevét, a végére pedig hozzáfűzi a .rtf kiterjesztést és „write” módba helyezi, majd visszaadja a fájlkezelőnek.

Korábban már beszéltünk az if __name__ rutinról. Emlékeztetőül csak annyit, hogy ha egy programot önállóan futtatunk, a __name__ belső változó a „__main__”-re van állítva. Ha ezt a rutint egy másik programból importálva hívjuk meg, akkor a kódrészlet ezen része egyszerűen csak figyelmen kívül lesz hagyva.

Most a példa kedvéért létrehozunk egy Renderer objektumot, meghívjuk a MakeExample rutint és megkapjuk a visszatérő object docot. Ez után az OpenFile rutint használva kiírjuk a fájlt (.doc-ba).

Most jön a mi jó öreg MakeExample rutinunk. Cseréljük le a „pass statement”-et a lent látható kódra.

Nézzük meg, hogy mit is hoztunk össze. Az első sorban egy Document, majd egy style sheet, illetve végül egy section példányt hozunk létre és csatolunk a dokumentumhoz. A section-re gondolkodunk úgy mint egy könyv fejezetére. Ezután létrehozunk egy paragrafust Normal stílussal. A PyRTF készítője ezt 11 pontos Arial betűtípusnak állította be. Ezután elhelyezzük a paragrafusban a kívánt szöveget és hozzáadjuk a section-

```
#!/usr/bin/env python
from PyRTF import *
```

```
def MakeExample():
    pass
```

```
def OpenFile(name) :
    return file('%s.rtf' % name, 'w')
```

```
if __name__ == '__main__' :
    DR = Renderer()
    doc = MakeExample()
    DR.Write(doc, OpenFile('rtftesta'))
    print "Finished"
```

hoz, majd visszaadjuk a doc dokumentumot.

Ez elég könnyű. Ismét nagyon körültekintően kell megtervezni a kimenetet, de semmi sem túl körülményes.

Mentsük a programot „rtftesta.py” néven és futtassuk. Amikor elkészült, használjuk az openoffi-

ce-t (vagy LibreOffice-t) a fájl megnyitásához és leellenőrzéséhez.

Most csináljunk egy-két trükkös dolgot. Először elhelyezünk egy címsort. A PyRTF fejlesztője ismét csak egy előre definiált Header1 nevű stílust készített számunkra. Ezt fogjuk felhasználni a címsorunkhoz. A doc.Sections.append sor és a p = Paragraph sor közé

```
doc = Document()
ss = doc.StyleSheet
section = Section()
doc.Sections.append(section)

p = Paragraph(ss.ParagraphStyles.Normal)
p.append('This is our first test writing to a RTF file. '
        'This first paragraph is in the preset style called normal '
        'and any following paragraphs will use this style until we change it.')
section.append(p)

return doc
```

helyezzük el a következőket:

```
p = Paragraph(ss.ParagraphStyles.Heading1)

p.append('Example Heading 1')

section.append(p)
```

Változtassuk meg az rtf fájl nevét "rtftestb-re" Valahogy így...

```
DR.Write(doc, OpenFile('rtftestb'))
```

Mentsük ezt rtftestb.py néven és futassuk. Most már van egy címsorunk. Biztos vagyok benne, hogy most azon töröd a fejed, hogy még mi mindent tudunk csinálni. Itt van a készítő által létrehozott stílusok listája.

Normal, Normal Short, Heading 1, Heading 2, Normal Numbered, Normal Numbered 2. Van még egy List stílus is, amit rátok hagyok, hogy kipróbáljátok. Ha még többre

Nézzük meg, hogy hogyan tudjuk megváltoztatni a betűtípusokat, betűméreteket és tulajdonságokat...

```
p = Paragraph(ss.ParagraphStyles.Normal)
p.append( 'It is also possible to provide overrides for elements of a style. ',
        'For example you can change just the font ',
        TEXT(' size to 24 point', size=48),
        ' or',
        TEXT(' typeface to Impact', font=ss.Fonts.Impact),
        ' or even more Attributes like',
        TEXT(' BOLD',bold=True),
        TEXT(' or Italic',italic=True),
        TEXT(' or BOTH',bold=True,italic=True),
        '.' )
section.append(p)
```

vagytok kíváncsiak, akkor a stílusok definícióját megtalálhatjátok a telepített disztribúció Elements.py fájlában.

Ugyan ezek a stílusok sok mindenre jók, de szükségünk lehet a felkínáltakon kívül másokra is. Nézzük meg, hogy hogyan tudjuk megváltoztatni a betűtípusokat, betűméreteket és tulajdonságokat (félkövér, dőlt, stb.) saját magunk. A paragrafusunk után, de még a dokumentum objektum visszaadása előtt írjuk be a következő kódot, és változtassuk meg a kimeneti fájl nevét rtftestc-re. Mentsük a programot rtftestc.py néven és futtasuk. A dokumentum új részének így kellene kinéznie...

Arra is lehetőségünk van, hogy felüldefiniáljuk a stílusok elemeit. Például átállíthatjuk csak a méretet

24 pontra vagy a betűtípust Impactra esetleg még több attribútumot is, mint például félkövér, vagy dőlt vagy MINDKETTŐ.

Most mit is csináltunk? Az első sor létrehoz egy új paragrafust. Ezután úgy folytatjuk, ahogy eddig is, a szöveg elhelyezésével. Nézzük meg a negyedik sort (TEXT(' size to 24 point', size = 48),). A TEXT módosító használatával megmondjuk a PyRTF-nek, hogy a mondat közepén valami mást csináljon, ami ebben az esetben a betűméret 24 pontosra állítása a 'size = ' utasítással. Várjunk csak! A méret itt 48, de amit írunk az 24 pont és a tényleges kimenet is 24 pontos szöveg lesz. Mi is történik itt? Nos, a size parancs fél-pontokkal működik. Tehát, ha 8 pontos betűket akarunk, akkor size = 16-ot kell használnunk. Érthető?

Következőnek a betűtípust állítjuk át a 'font =' paranccsal. Ismét, minden ami a TEXT parancson belül egyszeres idézőjelek között van, az fog módosulni és semmi más.

Ok. Ha mindezt megértettük, akkor mit tudunk még csinálni?

A szöveg színét is be tudjuk állítani a TEXT utasításon belül. Például így:

```
p = Paragraph()

p.append('This is a new paragraph with the word ',

        TEXT('RED',colour=ss.Colours.Red),

        ' in Red text.')

section.append(p)
```

Vegyük észre, hogy a paragraph stílusát nem kellett újra Normal-nak megadni. Ha egyszer már beállítottuk, akkor az úgy is marad, egészen addig amíg újra meg nem változtatjuk. Vegyük észre azt is, hogy ha az Egyesült Államokban élünk, a „colour” szó nyelvtanilag „megfelelő” alakját kell használnunk.

Ezek lennének az előre definiált színek: Black, Blue, Turquoise, Green, Pink, Red, Yellow, White, BlueDark, Teal, GreenDark, Violet, RedDark, YellowDark, GreyDark and Grey.

Álljon itt egy lista az előre megadott betűtípusokról is (a jelölésben kell őket beállítani)...

Arial, ArialBlack, ArialNarrow, BitstreamVeraSans, BitstreamVeraSerif, BookAntiqua, BookmanOldStyle, BookmanOldStyle, Castellar, CenturyGothic, ComicSansMS, CourierNew, FranklinGothicMedium, Garamond, Georgia, Haettenschweiler, Impact, LucidaConsole, LucidaSansUnicode, MicrosoftSansSerif, PalatinoLinotype, Monotype-Corsiva, Papyrus, Sylfaen, Symbol, Tahoma, TimesNewRoman, TrebuchetMS és Verdana.

```
p = Paragraph(ss.ParagraphStyles.Courier)
p.append('Now we are using the Courier style at 8 points. '
        'All subsequent paragraphs will use this style automatically. '
        'This saves typing and is the default behaviour for RTF documents.',LINE)
section.append(p)
p = Paragraph()
p.append('Also notice that there is a blank line between the previous paragraph ',
        'and this one. That is because of the "LINE" inline command.')

section.append(p)
```

Most biztos arra gondolsz, hogy ez mind szép és jó, de hogyan készíthetjük el a saját stílusainkat? Nagyon egyszerű, menj az eddig elkészített fájl elejére és a header sor előtt írd be a következő parancsokat.

```
result = doc.StyleSheet

NormalText = TextStyle(TextPropertySet(result.Fonts.CourierNew,16))

ps2 = ParagraphStyle('Courier',NormalText.Copy())

result.ParagraphStyles.append(ps2)
```

Mielőtt kipróbálnánk mit alkotunk, értsük meg előbb. Result néven létrehozunk egy új stíluslapot. A betűtípust és méretet a 8 pontos Courier New-ra állítjuk a második sorban és „regisztráljuk” a stílust Courier néven. Ne feledd, a betűméret megadásához 16-ot kell ír-

nunk, mivel a betűméretet félpont egységben kell megadni.

Most, mielőtt a rutin végén elérnénk a return sort, adjunk hozzá egy új bekezdést a Courier stílust használva.

Most már van egy új stílusod, amit bármikor használhatsz. A fent felsorolt listából bármelyik betűtípus a rendelkezésedre áll és létrehozatsz további stílusokat is. Egyszerűen csak másold át a stílus kódot és kedved szerint cseréld ki a betűtípusra és méretre vonatkozó információkat. Megtehetjük akár még ezt is...

```
NormalText = TextStyle(TextPropertySet(result.Fonts.Arial,22,bold=True,colour=ss.Colors.Red))

ps2 = ParagraphStyle('ArialBoldRed',NormalText.Copy())
```

```
result.ParagraphStyles.append(ps2)
```

Add még hozzá az alábbiakat is...

```
p = Paragraph(ss.ParagraphStyles.ArialBoldRed)

p.append(LINE,'And now we are using the ArialBoldRed style.',LINE)

section.append(p)
```

és nyomtass ArialBoldRed stílusban.

Táblázatok

Column Header 1	Column Header 2	Column Header 3
Row 1 data 1	Row 1 data 2	Row 1 data 3
Row 2 data 1	Row 2 data 2	Row 2 data 3

Sok esetben a táblázat az egyetlen módja annak, hogy az adatokat megfelelő módon jelenítsük meg egy dokumentumban. A táblázatok készítése egy egyszerű szövegben nehézkes, de NÉHÁNY esetben kifejezetten könnyű PyRTF-ben. Ezt az állítást egy kicsit később szeretném megmagyarázni.

Nézzünk meg egy átlagos OpenOffice/LibreOffice táblázatot, minden egy oszlopban végződik. Mutatok egy egyszerű példát.

A sorok balról jobbra, az oszlopok lefelé orientáltak. Egyszerű koncepció.

Indítsunk egy új alkalmazást, legyen a neve rtfTable-a.py. Kezdsnek másoljuk bele a standard programkódunkat és onnan építsük tovább.

Ez a kód szinte ugyanaz, mint amivel korábban már volt dolgunk. Most a TableExample rutint fogjuk kivesézni. Lényegében a PyRTF szerzője által biztosított példa-program egy részét használtam fel. Cseréld le a rutinban az alábbi kódra a pass statement-et...

```
doc = Document()
```

```
ss = doc.StyleSheet
```

```
section = Section()
```

```
doc.Sections.append(section)
```

Ez ugyanaz, mint ami korábban már előfordult.

```
table = Table(TabPS.DEFAULT_
WIDTH * 7,
```

```
TabPS.DEFAULT_WIDTH * 3,
```

```
TabPS.DEFAULT_WIDTH * 3)
```

Ez a sor (igen, valóban egy sorról van szó, csak az átláthatóság érdekében van így tördelve) hozza létre az alap táblázatot. Egy 3 oszlopos táblázatot készítünk, ahol az első 7, a második és a harmadik oszlop pedig 3 tab széles. A tab-okkal nem kell külön foglalkoznunk, a szélesség twip-ekben is megadható. Egy pillanat az egész.

```
c1 = Cell(Paragraph('Row One,
Cell One'))
```

```
c2 = Cell(Paragraph('Row One,
Cell Two'))
```

```
c3 = Cell(Paragraph('Row One,
Cell Three'))
```

```
table.AddRow(c1,c2,c3)
```

Itt most az első sor celláiba írandó adatokat állítjuk be.

```
#!/usr/bin/env python
```

```
from PyRTF import *
```

```
def TableExample():
    pass
```

```
def OpenFile(name):
    return file('%s.rtf' % name, 'w')
```

```
if __name__ == '__main__':
    DR = Renderer()
    doc = TableExample()
    DR.Write(doc, OpenFile('rtftable-a'))
    print "Finished"
```

```
c1 = Cell(Paragraph(ss.ParagraphStyles.Heading2, 'Heading2 Style'))
```

```
c2 = Cell(Paragraph(ss.ParagraphStyles.Normal, 'Back to Normal Style'))
```

```
c3 = Cell(Paragraph('More Normal Style'))
```

```
table.AddRow(c1,c2,c3)
```

Ez a kódrészlet adja meg a második sorba írandó adatokat. A stílust akár külön minden cellában is megváltoztathatjuk.

```
c1 = Cell(Paragraph(ss.ParagraphStyles.Heading2, 'Heading2 Style'))
```

```
c2 = Cell(Paragraph(ss.ParagraphStyles.Normal, 'Back to Normal Style'))
```

```
c3 = Cell(Paragraph('More Normal Style'))
```

```
table.AddRow(c1,c2,c3)
```

Ez pedig megadja az utolsó sort.

```
section.append(table)
```

```
return doc
```

Ez hozzácsatolja a táblázatot az adott részhez és visszakapjuk a dokumentumot, lehet nyomtatni.

Mentsd el és futtasd az alkalmazást. Észre fogod venni, hogy minden olyan lett, mint amit szeretnél volna, egyedül csak a táblázat széle fog hiányozni. Ez bonyolítja a helyzetet, ezért gyorsan javítsuk ki.

Ismét csak egy meglévő példa programkódot fogok használni, amit a PyRTF alkotója készített.

Mentsd el a fájlt `rtftable-b.py` néven. Most a `TableExample` rutinban töröld ki mindent a `'doc.Sections.append(section)'` és `'return doc'` rész között és helyette írd be ezt...

```
thin_edge = BorderPS(
width=20, style=BorderPS.SINGLE )
```

```
thick_edge = BorderPS(
width=80, style=BorderPS.SINGLE )
```

```
thin_frame = FramePS(
thin_edge, thin_edge,
thin_edge, thin_edge )
```

```
thick_frame = FramePS(
thick_edge, thick_edge,
thick_edge, thick_edge )
```

```
mixed_frame = FramePS(
thin_edge, thick_edge,
thin_edge, thick_edge )
```

Most állítottuk be a határokat és a kereteket megszabó paramétereiket.

```
table = Table( TabPS.DEFAULT_WIDTH * 3, TabPS.DEFAULT_WIDTH * 3,
TabPS.DEFAULT_WIDTH * 3 )
```

```
c1 = Cell( Paragraph( 'R1C1' ), thin_frame )
```

```
c2 = Cell( Paragraph( 'R1C2' ), )
```

```
c3 = Cell( Paragraph( 'R1C3' ), thick_frame )
```

```
table.AddRow( c1, c2, c3 )
```

Az első sorban az első oszlop (vékony keret) és a harmadik oszlop (vastag keret) cellái határokat kaptak maguk köré.

```
c1 = Cell( Paragraph( 'R2C1' ), )
```

```
c2 = Cell( Paragraph( 'R2C2' ), )
```

```
c3 = Cell( Paragraph( 'R2C3' ), )
```

```
table.AddRow( c1, c2, c3 )
```

A második sorban egyik cella sem kap határokat.

```
c1 = Cell( Paragraph( 'R3C1' ), mixed_frame )
```

```
c2 = Cell( Paragraph( 'R3C2' ), )
```

```
c3 = Cell( Paragraph( 'R3C3' ), mixed_frame )
```

```
table.AddRow( c1, c2, c3 )
```

A harmadik sor első és harmadik oszlopában ismét csak vegyes keretekkel találkozunk.

```
section.append( table )
```

Nos, most már mindened megvan ahhoz, hogy kód által RTF dokumentumokat készíts.

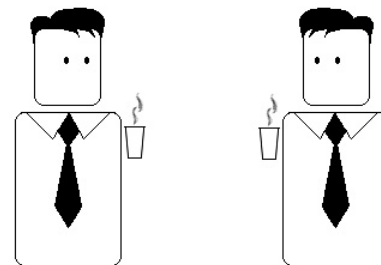
Találkozzunk legközelebb is!

A forráskódok – ahogy általában – most is megtalálhatók [pastebin-en](http://pastebin.com/3Rs7T3D7). Az első csomag a <http://pastebin.com/3Rs7T3D7> címen érhető el, ami az `rtftest.py` (a-e) részt foglalja össze. A második részben az `rtftable.py` (a-b) található, letölthető a <http://pastebin.com/XbaE2uP7> linkről.

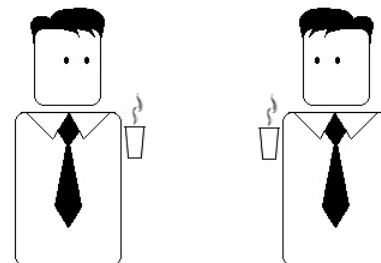


Greg Walters a *RainyDay Solutions Kft.* tulajdonosa, amely egy tanácsadó cég a coloradói Aurórában. Greg 1972 óta foglalkozik programozással. Szeret főzni, túrázni, zenét hallgatni, valamint a szabadidejét családjával tölteni. Weblapja a címen www.thedesiginateedgeek.com található meg.

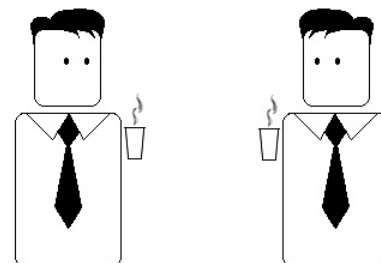
A tény, hogy elértük az 50. kiadást újra definiálja az online kiadást.



Mindent megváltoztatt.



Mondtam, hogy ne vegyél iPhone-t.



by Richard Redei



Mivel sokan jelezték, hogy nagyon tetszetek a GUI programozós cikkek, ezért válaszul úgy döntöttem, hogy megnézzünk egy másik, *Tkinter*-nek nevezett GUI toolkitet is. Igazából Python alatt ez lenne a GUI programozás „hivatalos” módja. A Tkinter már régóta létezik és elég rossz a híre amiatt, hogy „régimódinak” néz ki. Mivel ez azonban nemrég megváltozott, így erre könnyen rá fogunk cáfolni a későbbiekben.

FIGYELEM – Minden bemutatott kód Python 2.x verziójú. Egy leendő cikkben meg fogjuk beszélni, hogy hogyan lehet a tkintert Python 3.x alatt is használni. Ha MINDENKÉPPEN Python 3.x-el kell dolgoznotok, akkor változtassátok az import utasításokat „from tkinter import *-ra”.

Egy kis történelem meg néhány háttérinformáció

A Tkinter a „Tk interface” szavakból jön. A Tk egy különálló programozási nyelv, és a Tkinter segítségével használhatjuk ennek

GUI eszközeit. Több olyan widget is létezik, ami alpból megtalálható a Tkinter modulban. Néhány ezek közül: Toplevel (pl. főablak) tároló, gombok (Buttons), Címkek (Labels), keretek (Frames), Szövegmezők (Text Entry), jelölő dobozok (CheckButtons), rádió gombok (RadioButtons), háttér (Canvas), többsoros beviteli mező (Multiline Text entry), stb. Ezen kívül még olyan modulok is léteznek, amelyek további szolgáltatásokat adnak a Tkinterhez. Ez alkalommal néhány widgetre fogunk csak koncentrálni: a Toplevelre (továbbiakban a gyökér ablakként hivatkozunk rá), a Frame-re, a Labelre és a Button-ra. A következő cikkben más widgeteket is meg fogunk vizsgálni.

Gyakorlatilag van egy Toplevel konténerünk amiben a többi widgetet tároljuk. Ez a gyökér, vagy más néven a főablak. A gyökérablakon belül helyezzük el azokat a widgeteket, amiket a programunkban használni akarunk. Mindegyik widgetnek – a Toplevel kivételével – van egy szülője. A szülőnek nem feltétlenül kell a gyökér ablaknak lennie, mert egy

másik widget is betöltheti ezt a pozíciót. Erről bővebben a következő számban fogunk beszélni. Egyenlőre minden widgetnek a főablak lesz az őse.

Ahhoz, hogy elhelyezzük és megjelenítsük a gyermek widgeteket, egy „geometry managementnek” (geometria kezelő) nevezett dolgot kell használnunk. Ezen keresztül helyezzük el a dolgainkat a gyökér ablakon. A legtöbb programozó a három féle geometria kezelő valamelyikét használja csak: a Packert, a Gridet vagy a Place-t. Véleményem szerint a Packer egy nagyon ügyetlen módszer, ezért rátok hagyom, hogy kipróbáljátok. A Place segítségével szerfelett pontosan tudjuk elhelyezni a widgeteket, de igen bonyolult lehet a használata. Emiatt a Place módszert egy későbbi cikkben fogjuk majd megnézni. Kizárásos alapon így most csak a Gridre fogunk koncentrálni.

A Gridre gondoljunk úgy, mint egy táblázatra. Vannak sorai és oszlopai. Az oszlopok függőlegesek, a sorok pedig vízszintesek.

	COLUMNS - >				
ROWS	0,0	1,0	2,0	3,0	4,0
	0,1	1,1	2,1	3,1	4,1
	0,2	1,2	2,2	3,2	4,2
	0,3	1,3	2,3	3,3	4,3

Példának itt egy 5 oszlopos és 4 soros táblázat a cellák koordinátáival együtt.

Ebben az esetben a grid lesz a szülő, a gyermekei pedig az egyes cellákba fognak kerülni. Első pillantásra azt gondolhatnánk, hogy ez egy igen behatárolt módszer. Azonban a widgeteket több sorba és/vagy oszlopba is elhelyezhetjük.

Első példa

Az első példánk egy SZUPER egyszerű (csak négy sorból álló) program lesz.

```
from Tkinter import *

root = Tk()

button = Button(root, text = "Hello FullCircle").grid()

root.mainloop()
```

Nos, mi is van itt? Az első sor

beimportálja a Tkinter modult. Ezt követően root néven inicializáljuk a Tk objektumot (a Tk a Tkinter része). Itt a negyedik sor:

```
button = Button(root, text =  
"Hello FullCircle").grid()
```

Létrehozunk egy button nevű gombot, aminek a szülője a root és a szövege a „Hello FullCircle”, majd ezt belehelyezzük a gridbe. Végül meghívjuk az ablak fő ciklusát. A mi szempontunkból ez igen egyszerű, de a háttérben igen sok minden történik. Viszont szerencsére most ezzel egyáltalán nem kell foglalkoznunk.

Futtassuk a programot és nézzük meg, hogy mit készítettünk. Az én gépem a főablak a képernyő bal alsó sarkában jelent meg. Nálatok esetleg máshol is megjelenhet. A gombot megnyomva semmi sem történik. A következő példában ezt fogjuk kiküszöbölni.

Második példa

Most egy App nevű osztályt hozunk létre. Ez az osztály tartalmazza majd az ablakunkat. Vágjunk is bele.

```
from Tkinter import *
```

```
class App:  
    def __init__(self, master):  
        frame = Frame(master)  
        self.lblText = Label(frame, text = "This is a label widget")  
        self.btnQuit = Button(frame, text="Quit", fg="red", command=frame.quit)  
        self.btnHello = Button(frame, text="Hello", command=self.SaySomething)  
        frame.grid(column = 0, row = 0)  
        self.lblText.grid(column = 0, row = 0, columnspan = 2)  
        self.btnHello.grid(column = 0, row = 1)  
        self.btnQuit.grid(column = 1, row = 1)
```

Ez a Tkinter import utasítása.

Definiáljuk az osztályt, majd az `__init__` rutinban beállítjuk a widgetjeinket és elhelyezzük őket a griden.

Az `__init__` rutin első sora létrehozza a keretet (frame), ami majd az összes többi widgetünk szülője lesz. A keret őse pedig a gyökér ablak (azaz a Toplevel widget). Ezután definiálunk egy címkét és két gombot. Nézzük meg a címke létrehozásának sorát.

```
self.lblText = Label(frame,  
text = "This is a label  
widget")
```

Létrehozuk a Label widget objektumból származó címke widgetet `self.lblText` néven. Beállítjuk a szülőjét (frame) és a megjelenítendő szöveget (`text = "this is a label`

widget”). És végeztünk.

Természetesen ennél többet is megtehetünk, de egyenlőre ennyire van csak szükségünk. Következőnek beállítjuk a két gombot:

```
self.btnQuit = Button(frame,  
text="Quit", fg="red",  
command=frame.quit)
```

```
self.btnHello =  
Button(frame, text="Hello",  
command=self.SaySomething)
```

Elnevezzük a widgeteket, beállítjuk a szülőiket és a kiíratandó szövegeket. A `btnQuit`nek van egy `fg` nevű attribútuma is, amit „red”-re állítottunk. Könnyű kitalálni, hogy ez az előtér, azaz a szöveg színét állítja pirosra. Az utolsó attribútum a gombra kattintás callback metódusának beállítása. A `btnQuit` esetében ez a `frame.quit` utasítás, ami leállítja a programot. Ez egy beépített tagfüggvény,

ezért nem nekünk kell megírni. A `btnHello`nál egy `self.SaySomething`nek nevezett rutint használunk. Ezt viszont már meg kell csinálnunk, de előtte néhány dolgot még megnézzünk.

El kell helyeznünk a widgeteinket a griden. Ennek a sorai:

```
frame.grid(column = 0, row = 0)
```

```
self.lblText.grid(column = 0, row = 0, columnspan = 2)
```

```
self.btnHello.grid(column = 0, row = 1)
```

```
self.btnQuit.grid(column = 1, row = 1)
```

Először hozzárendeljük a gridet a frame-hez, majd a grid attribútumon keresztül beállítjuk a widgetek pozícióit. Figyeljük meg a címke (`self.lblText`)

columnspan részét. Ez azt mondja meg, hogy a címkénk két oszlopot fog elfoglalni. Azonban, mivel összesen csak két oszlopunk van, ez gyakorlatilag az egész alkalmazást le fogja fedni. Mostmár létrehozhatjuk a callback tagfüggvényünket:

```
def SaySomething(self):  
  
    print "Hello to FullCircle  
    Magazine Readers!!"
```

Ez egyszerűen kiírja a terminál ablakba a "Hello to FullCircle Magazine Readers!!" üzenetet.

Végül példányosítjuk a Tk osztályt – illetve az App osztályunkat – és futtatjuk a fő ciklust.

```
root = Tk()  
  
app = App(root)  
  
root.mainloop()
```

Próbáljuk ki. Most már tényleg csinál is valamit. De megint elég kényelmetlen helyen jelenik meg az ablak. A következő példában ezt a kis hibát fogjuk orvosolni.

Harmadik példa

Mentsük el az előző alkalmazást example3.py néven. Egyetlen sor kivételével minden meg fog egyezni. A program végén, a példányosításkor találjuk meg az új részt:

```
root = Tk()  
  
root.geometry('150x75+550+150')  
  
app = App(root)  
  
root.mainloop()
```

Ez a sor az ablak kezdeti méretét 150 pixel szélesre és 75 pixel magasra állítja be. Továbbá azt is megadjuk, hogy az ablak bal felső sarka az x szerinti 550-es (bal és

```
class Calculator():  
    def __init__(self, root):  
        master = Frame(root)  
        self.CurrentValue = 0  
        self.HolderValue = 0  
        self.CurrentFunction = ''  
        self.CurrentDisplay = StringVar()  
        self.CurrentDisplay.set('0')  
        self.DecimalNext = False  
        self.DecimalCount = 0  
        self.DefineWidgets(master)  
        self.PlaceWidgets(master)
```

jobb), illetve az y szerinti 150-es (fent és lent) pozíción legyen.

Hogy honnan jönnek ezek a számok? Nos, egy ésszerűnek tűnő értéket állítottam amíg elfogadható eredményt nem kaptam. Ez egy elég körülményes módszer, de még így is jobb, mint ha egyáltalán nem csináltunk volna semmit.

Negyedik példa – Egy egyszerű számológép

Most nézzünk meg valami bonyolultabbat. Ez alkalommal egy négy funkciós számológépet fogunk elkészíteni. A „négy funkció” a következő négy műveletet jelenti: Összeadás, Kivonás, Szorzás és Osztás. A program az alábbi, szöveges formátumban megadott módon fog kinézni.

Vágjunk is a közepébe és majd menet közben elmagyarázom a kódot.

A geometry utasítás kivételével mindennek érthetőnek kellene lennie. Emlékezzünk arra, hogy vegyünk valami ésszerű értéket és

```
-----  
|                               0 |  
-----  
| 1 | 2 | 3 | + |  
-----  
| 4 | 5 | 6 | - |  
-----  
| 7 | 8 | 9 | * |  
-----  
| - | 0 | . | / |  
-----  
|                               = |  
-----  
|                               CLEAR |  
-----
```

```
from Tkinter import *
```

```
def StartUp():  
    global val, w, root  
    root = Tk()  
    root.title('Easy Calc')  
    root.geometry('247x330+469+199')  
    w = Calculator(root)  
    root.mainloop()
```

bütyköljük egy kicsit, majd haladjunk tovább.

Az osztály definíciójával és az __init__ tagfüggvény beállításával kezdünk.

A következő három változót használjuk:
– CurrentValue – A számológépbe betáplált aktuális értéket tartalmazza.

- **HolderValue** – Egy funkciógomb megnyomása előtti értéket raktározza el.
- **CurrentFunction** – Ez egy egyszerű „könyvjelző”, ami megadja, hogy melyik művelettel kell dolgoznunk.

Ezután definiáljuk a **CurrentDisplay** változót és hozzárendeljük a **StringVar** objektumhoz. Ez egy olyan speciális objektum, ami a Tkinter eszközenszer része.

Bármelyik widgethez is rendeljük hozzá, az automatikusan frissíteni fogja a widgetben található értéket. A mi esetünkben arra fogjuk használni, hogy eltárolja az általunk a **display** címkén keresztül megjeleníteni kívánt dolgokat. Mielőtt a widgethez tudnánk rendelni, előbb példányosítanunk kell, ezután használjuk a beépített **'set'** függvényt. Következőnek egy **DecimalNext** nevű, logikai típusú és egy **DecimalCount** változót definiálunk, majd meghívjuk a **DefineWidgets** függvényt, ami létrehozza az összes widgetet, illetve meghívjuk még a **PlaceWidget** rutint is, ami igazából elhelyezi őket a főablakban.

```
def DefineWidgets(self, master):
```

```
self.lblDisplay = Label(master, anchor=E, relief = SUNKEN, bg="white", height=2, textvariable=self.CurrentDisplay)
```

Már készítettünk korábban címkéket. Most azonban egy pár attribútumot is használni fogunk. Vegyük észre, hogy nem vettük igénybe a **'text'** tulajdonságot. Itt rendeljük hozzá a címkét a szülőjéhez (**master**), állítjuk be a szöveg anchorját (avagy a mi szempontunkból az igazítást). Ezzel arra utasítjuk a címkét, hogy minden szöveget keletre, azaz jobbra igazítsa a widgeten belül. Van **justify** attribútum is, de az csak több soros szövegeknél alkalmazható. Az **anchor** tulajdonságnak a következő beállításai vannak: **N** (É), **NE** (ÉK), **E** (K), **SE** (DK), **SW** (DNy), **W** (Ny), **NW** (ÉNy) és végül az alapértelmezett **CENTER**. Gondoljunk ezekre úgy, mint az iránytű jeleire. Rendes körülmények között mi csak az **E**-t (jobb), a **W**-t (bal) és a **Center**t használjuk.

Következőnek a **relief**-t, avagy a címke szövegének kinézetét állítjuk be. A „megadható” értékek a **FLAT**, a **SUNKEN**, a **RAISED**, a **GROOVE** és a **RIDGE**. Ha semmi

```
self.btn1 = Button(master, text = '1', width = 4, height=3)
self.btn1.bind('<ButtonRelease-1>', lambda e: self.funcNumButton(1))
self.btn2 = Button(master, text = '2', width = 4, height=3)
self.btn2.bind('<ButtonRelease-1>', lambda e: self.funcNumButton(2))
self.btn3 = Button(master, text = '3', width = 4, height=3)
self.btn3.bind('<ButtonRelease-1>', lambda e: self.funcNumButton(3))
self.btn4 = Button(master, text = '4', width = 4, height=3)
self.btn4.bind('<ButtonRelease-1>', lambda e: self.funcNumButton(4))
```

mást nem adunk meg, akkor a **FLAT** az alapértelmezett. Nyugodtan próbáljátok ki a többi beállítást is.

Ezután, hogy jól elkülönüljön az ablak többi részétől a háttér színét (**bg**) fehérnek adjuk meg. A magasságot **2**-re állítjuk (ami két sort és nem két pixelt jelent), és végül hozzárendeljük a nemrég elkészített változót (**self.CurrentDisplay**) a **textvariable** attribútumhoz. Így a címke szövege követni fogja a **self.CurrentDisplay** változását.

Most létrehozunk néhány gombot.

Itt csak négy gombbal fogunk foglalkozni, mivel – ahogy látható is – a kód szinte teljesen ugyanolyan. Korábban ebben a cikkben már készítettünk gombokat, de azért nézzük meg az itt lévő kódot egy kicsit közelebbről.

Definiáljuk a szülőt (**master**),

majd beállítjuk a gomb szövegét, szélességét és magasságát is. Figyeljük meg, hogy a szélesség karakterekben, a magasság pedig sorokban van számolva. Ha valamilyen grafikával dolgoznánk a gombon, akkor ugyanezek pixelben lennének megadva. Ez egy kicsit zavaró lehet addig, amíg hozzá nem szokunk. Ezután beállítjuk a **bind** tulajdonságot. Amikor korábban a gombokkal foglalkoztunk, akkor a **'command='** attribútummal adtuk meg a kattintáskor végrehajtandó függvényt. Most ehhez a **'bind'**-ot fogjuk használni. Ez majdnem ugyanaz, de sokkal egyszerűbb használni és átadni adatokat a statikus call-backeknek. Vegyük észre, hogy a **bind** kioldójaként a **'<ButtonRelease-1>'**-t használjuk. Ebben az esetben azt akarjuk, hogy a felhasználó kattintása ÉS az egérgomb felengedése után hívódjon meg a callback. Végül megadjuk magát a használni kívánt callback függvényt is.

A szorgalmasabbak (azaz gyakorlatilag mindannyiőtök), fel fognak fedezni egy új dolgot. Ez pedig a 'lambda e:' hívás.

A Lambdát Pythonban olyan név nélküli függvények definiálására használjuk, amelyeket kifejezések helyén adhatunk meg. Ez által lehetőségünk van arra, hogy több egybefüggő részt egy sorban helyezzünk el. Gondoljunk rá úgy, mint egy apró függvénykére. A példában a callback nevét, az átadandó értéket és az esemény jelét (e:) állítjuk be. A Lambda kifejezésekkel egy későbbi cikk

ben részletesebben is foglalkozni fogunk. Egyenlőre kövessük a példát.

Az első négy gombot már megadtam. Másoljátok és illesszétek be a fenti kód 0-ás és 5-9-es gombjait. Mindegyik ugyan úgy néz ki, a neveik és a callback értékeinek kivételével.

Az egyedüli dolgok amiről még nem volt szó, a columnspan és a sticky attribútumok. Mint ahogy már tudjuk, egy widget több oszlopba vagy több sorba is elhelyezhető. Ebben az esetben a címke widgetet mind a négy oszlopban

```
self.btnDash = Button(master, text = '-',width = 4,height=3)
self.btnDash.bind('<ButtonRelease-1>', lambda e: self.funcFuncButton('ABS'))
self.btnDot = Button(master, text = '.',width = 4,height=3)
self.btnDot.bind('<ButtonRelease-1>', lambda e: self.funcFuncButton('Dec'))
```

A btnDash a begépelt érték ellentettjét adja vissza, azaz a 523 -523, a -523 pedig 523 lesz. A btnDot egy tizedes pontot helyez el. Ezek a példák és a lentiek a funcFuncButton callbacket használják.

```
self.btnPlus = Button(master,text = '+', width = 4, height=3)
self.btnPlus.bind('<ButtonRelease-1>', lambda e: self.funcFuncButton('Add'))
self.btnMinus = Button(master,text = '-', width = 4, height=3)
self.btnMinus.bind('<ButtonRelease-1>', lambda e:
self.funcFuncButton('Subtract'))
self.btnStar = Button(master,text = '*', width = 4, height=3)
self.btnStar.bind('<ButtonRelease-1>', lambda e: self.funcFuncButton('Multiply'))
self.btnDiv = Button(master,text = '/', width = 4, height=3)
self.btnDiv.bind('<ButtonRelease-1>', lambda e: self.funcFuncButton('Divide'))
self.btnEqual = Button(master, text = '=')
self.btnEqual.bind('<ButtonRelease-1>', lambda e: self.funcFuncButton('Eq'))
```

Itt van a négy matek függvényes gomb.

```
self.btnClear = Button(master, text = 'CLEAR')
self.btnClear.bind('<ButtonRelease-1>', lambda e: self.funcClear())
```

Végül pedig a törlés gombja. Ez természetesen kiüríti a tárolt változókat és törli a kijelzőt. Most helyezzük el a widgeteket a PlaceWidget rutinban. Először inicializáljuk a gridet, majd elkezdjük belepakolni a widgeteket. Itt van a rutin első fele.

```
def PlaceWidgets(self, master):
    master.grid(column=0,row=0)
    self.lblDisplay.grid(column=0,row=0,columnspan = 4,sticky=EW)
    self.btn1.grid(column = 0, row = 1)
    self.btn2.grid(column = 1, row = 1)
    self.btn3.grid(column = 2, row = 1)
    self.btn4.grid(column = 0, row = 2)
    self.btn5.grid(column = 1, row = 2)
    self.btn6.grid(column = 2, row = 2)
    self.btn7.grid(column = 0, row = 3)
    self.btn8.grid(column = 1, row = 3)
    self.btn9.grid(column = 2, row = 3)
    self.btn0.grid(column = 1, row = 4)
```

keresztül „széthúzzuk”. Ez a „columnspan” tulajdonság feladata. Van még egy „rowspan” nevű is. A „sticky” attribútum megmondja a widgetnek, hogy hova illessze az oldalait. Gondoljunk erre úgy, mint a widget griden belüli elhelyezkedésének módjára. Itt van a többi gomb is.

Mielőtt még továbblépnénk, nézzük meg, hogy mi történik akkor, amikor a felhasználó egy gombra kattint.

Tegyük fel, hogy az ügyfél a $563 + 127$ -et akarja beütni. Rendre megnyomja vagy leüti az 5-ös, a 6-os, a 3-as, a „+”, az 1-es, a 2-es, a 7-es, majd végül az „=” gombot. Hogyan kezeljük le ezt a kódban?

A számgombok callbackjét már beállítottuk a funcNumButton rutinra. A megvalósításra két lehetséges mód is van. Lehetőségünk van a bevitt értékek stringként való tárolására úgy, hogy csak a számolásakor konvertáljuk számmá, vagy már eleve számként kezeljük őket. Mi most az utóbbit fogjuk megvalósítani. Ehhez meg fogjuk tartani a már a „self.CurrentValue-ban” lévő értéket (kezdésnek 0).

```
self.btnDash.grid(column = 0, row = 4)
self.btnDot.grid(column = 2, row = 4)
self.btnPlus.grid(column = 3, row = 1)
self.btnMinus.grid(column = 3, row = 2)
self.btnStar.grid(column = 3, row = 3)
self.btnDiv.grid(column=3, row = 4)
self.btnEqual.grid(column=0,row=5,columnspan = 4,sticky=NSEW)
self.btnClear.grid(column=0,row=6,columnspan = 4, sticky = NSEW)
```

```
def funcNumButton(self, val):
    if self.DecimalNext == True:
        self.DecimalCount += 1
        self.CurrentValue = self.CurrentValue + (val * (10**(-self.DecimalCount)))
    else:
        self.CurrentValue = (self.CurrentValue * 10) + val
    self.DisplayIt()
```

Amikor egy számot kapunk, a tároltat megszorozzuk 10-el és hozzáadjuk az új értéket. Tehát, ha a felhasználó beüti az 5, 6 és 3 számokat, mi a következőt csináljuk...

A felhasználó az 5-re kattint – $0 * 10 + 5$ (5)

A felhasználó a 6-ra kattint – $5 * 10 + 6$ (56)

A felhasználó a 3-ra kattint – $56 * 10 + 3$ (563)

Természetesen a címkében mindig megjelenítjük a „self.CurrentValue” változót.

Ezután a „+” gomb lesz lenyomva. Ekkor fogjuk a „self.CurrentValue” tartalmát és átrakjuk a

„self.HolderValue-ba”, majd töröljük a „self.CurrentValue” értékét 0-ra, illetve megismételjük mindezt az 1, 2 és 7 gombokkal. Amikor a felhasználó az „=” gombra kattint, összeadjuk a „self.CurrentValue” és „self.HolderValue” változók tartalmát és megjelenítjük az eredményt. Mielőtt továbblépnénk, mindkét változót töröljük.

Most már nekiláthatunk a callbackek definiálásának:

A „funcNumButton” rutin megkapja a gombnyomáskor átadott értéket. Az egyetlen, a fenti példától eltérő dolog, az a tizedes pont („.”) megnyomásának esete. Lentebb láthatjuk, hogy egy logikai tí-

pusú változót használunk a pont lenyomásának rögzítésére, melyet a következő kattintáskor fogunk lekezelni. Ez lenne az „if self.DecimalNext == True:” sor. Menjünk is végig rajta.

A felhasználó a „32.4” begépeléséhez megnyomja a 3-at és a 2-őt, majd a tizedes pontot, végül a 4-et. A 3-at és a 2-őt a „funcNumButton”-on keresztül kezeljük le. Megnézzük, hogy a self.DecimalNext igaz-e (ami ebben az esetben hamis addig amíg a „.”-ot meg nem nyomjuk). Ha nem, akkor egyszerűen megszorozzuk a tárolt értéket 10-el és hozzáadjuk a bejövő számot. Amikor a felhasználó leüti a

„-ot, a „funcFuncButton” a „Dec” értékkel lesz meghívva. Ekkor csak annyit teszünk, hogy beállítjuk a „self.DecimalNext” változót True-ra. Amikor a 4-esre kattintunk a self.DecimalCount igaz lesz. És most jön a trükk. Először megnöveljük a self.DecimalCount értékét. Ez mondja meg, hogy hány tört hellyel dolgozunk. Következőnek vesszük a bejövő számot és megszorozzuk (10^{**} -self.DecimalCount)-al. A $**$ operátorral a hatványozás műveletét hajtjuk végre. Például a $10^{**}2$ 100-al, a $10^{**}-2$ pedig 0.01-al tér vissza. Valójában ez a művelet kerekítési hibától szenved, de a mi egyszerű számológépünk esetében a legtöbb értelmes tört számnál még használható lesz. Rátok hagyom egy hatékonyabb függvény létrehozását. Gondoljatok erre úgy, mint az e havi házi feladatra.

A „funcClear” rutin egyszerűen csak törli a két tároló változót, majd beállítja a kijelzőt.

```
def funcClear(self):  
  
    self.CurrentValue = 0  
  
    self.HolderValue = 0  
  
    self.DisplayIt()
```

```
def funcFuncButton(self, function):  
    if function == 'Dec':  
        self.DecimalNext = True  
    else:  
        self.DecimalNext = False  
        self.DecimalCount = 0  
        if function == 'ABS':  
            self.CurrentValue *= -1  
            self.DisplayIt()
```

Az ABS függvény egyszerűen veszi az aktuális értéket és megszorozza -1-el.

```
        elif function == 'Add':  
            self.HolderValue = self.CurrentValue  
            self.CurrentValue = 0  
            self.CurrentFunction = 'Add'
```

Az összeadás művelet átmásolja a „self.CurrentValue” változót a „self.HolderValue”-ba, törli a „self.CurrentValue”-t és beállítja a „self.CurrentFunction”-t „Add”-ra. A kivonás, a szorzás és az osztás műveletek ugyanezt csinálják, csak a „self.CurrentFunction” értéke lesz más.

```
        elif function == 'Subtract':  
            self.HolderValue = self.CurrentValue  
            self.CurrentValue = 0  
            self.CurrentFunction = 'Subtract'  
        elif function == 'Multiply':  
            self.HolderValue = self.CurrentValue  
            self.CurrentValue = 0  
            self.CurrentFunction = 'Multiply'  
        elif function == 'Divide':  
            self.HolderValue = self.CurrentValue  
            self.CurrentValue = 0  
            self.CurrentFunction = 'Divide'
```

Az „Eq” (egyenlőség) függvényénél történik a lényeg. A következő kódot most már könnyen értelmezni tudjátok.

```
        elif function == 'Eq':  
            if self.CurrentFunction == 'Add':  
                self.CurrentValue += self.HolderValue  
            elif self.CurrentFunction == 'Subtract':  
                self.CurrentValue = self.HolderValue - self.CurrentValue  
            elif self.CurrentFunction == 'Multiply':  
                self.CurrentValue *= self.HolderValue  
            elif self.CurrentFunction == 'Divide':  
                self.CurrentValue = self.HolderValue / self.CurrentValue  
            self.DisplayIt()  
            self.CurrentValue = 0  
            self.HolderValue = 0
```

Most pedig jöjjenek a műveletek. A 'Dec'-et már tisztáztuk az első „if” utasításban. Ha az „else”-re lépünk, és ha bármi más a függvény, akkor töröljük a „self.DecimalNext” és a „self.DecimalCount” változókat.

A DisplayIt egyszerűen csak beállítja a megjelenítő címke tartalmát. Emlékezzünk, hogy a címkének azt az utasítást adtuk, hogy „figyelje” a „self.CurrentDisplay” változó értékét. Bármikor, amikor az új értéket vesz fel, a címke automatikusan követni fogja a változásokat. A „.set” metódust az érték módosítására használjuk.

```
def DisplayIt(self):  
  
    print('CurrentValue = {0} -  
    HolderValue = {1}'.format(self.CurrentValue, self.HolderValue))
```

```
self.CurrentDisplay.set(self.CurrentValue)
```

Végül itt vannak még a kezdő sorok.

```
if __name__ == '__main__':  
    StartUp()
```

Most már futtathatjuk és kipró-

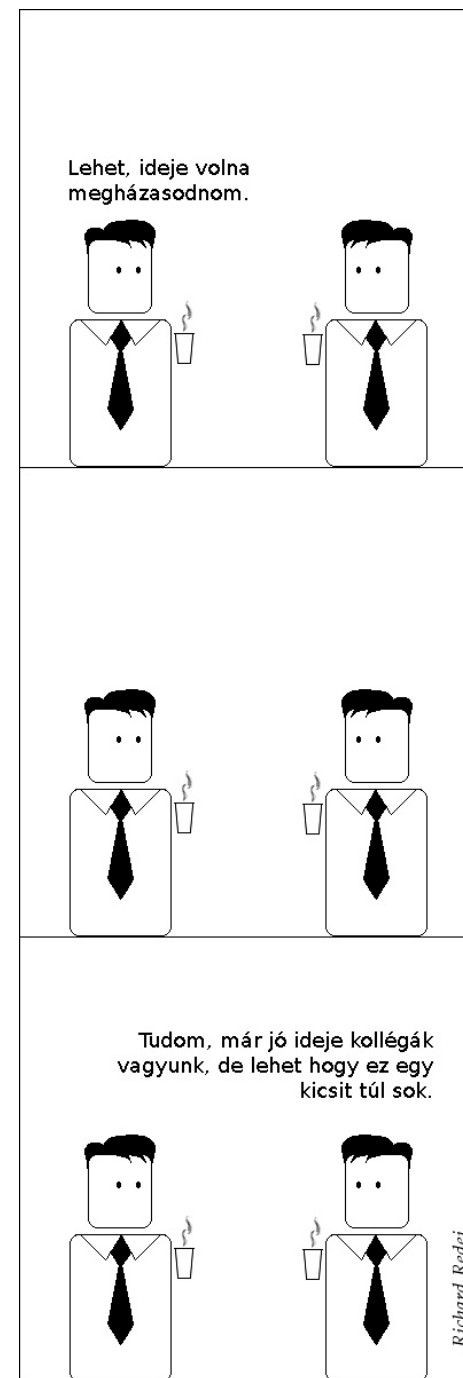
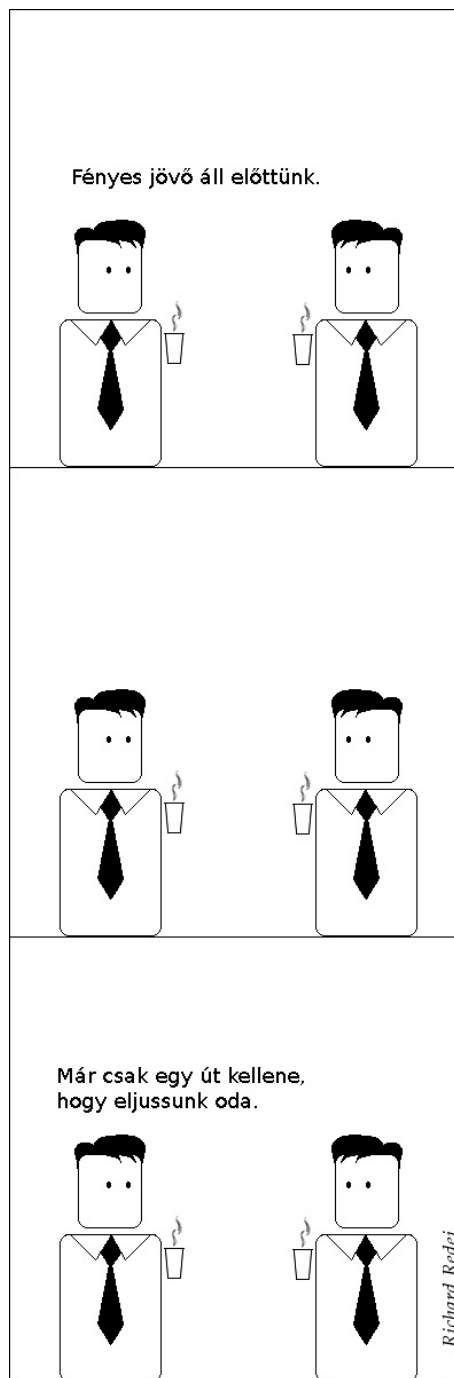
bálhatjuk a programot.

Mint mindig, a kódokat a Pastebin-en találjátok meg. Az 1., 2. és 3. példákat itt: <http://pastebin.com/mBAS1Umm> a Calc.py-t pedig itt: <http://pastebin.com/LbMibF0u>

A következő hónapban folytatjuk a tkintert és megnézzünk még egy rakat új widgetet. Az azt követő cikkekben pedig a tkinter PAGE nevű GUI szerkesztőjét próbáljuk ki. Addig is érezzétek magatokat jól a Tkinterrel!



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 címen www.thedesigntededgeek.com található meg.





Múltkor a tkInterről és annak négy widgetjéről, a TopLevelről, a Frame-ről, a Button-ról és a Labelről volt szó. Előző hónapban azt is megígértem, hogy megmutatom hogyan lehet olyan widgeteket létrehozni, amelyeknek nem a TopLevel az őse.

Ebben a cikkben beszélni fogunk még egy kicsit a Frame-ekről, a Buttonokról és a Labelekről, majd bemutatom a jelölőnégyzeteket (checkbox), a rádió gombokat (radio button), a szöveg mezőket (textbox, Entry widget), a függőleges görgetősávval rendelkező listákat (listbox) és végül az üzenet ablakokat (messagebox). Mielőtt belevágunk, vizsgáljuk meg ezeket a widgeteket.

A jelölőnégyzetek olyan többszörös választó widgetek, amelyeknek két, bejelölt vagy üres (azaz bekapcsolt vagy kikapcsolt) állapota van. Általában arra használjuk őket, hogy felajánljunk különböző választási lehetőségeket, amik közül néhányat vagy az összeset be lehet jelölni, illetve akár teljesen fi-

gyelmen kívül is hagyhatjuk mindet. Eseményt is hozzárendelhetünk a bejelölés pillanatához, vagy a nekünk megfelelő időpontban lekérdezhetjük a widget állapotát.

A rádió gombok egyszeres választást tesznek lehetővé. Ezeknek is két, ki- vagy bekapcsolt állapota van. Ezen felül olyan csoportokba vannak szervezve, amelyeknél logikailag egyszerre csak egy jelölhető be. Lehet továbbá több rádiógomb csoportunk is, amik - ha megfelelően vannak leprogramozva - nem fogják egymást befolyásolni.

A listák az elemek olyan felsorolása, amelyek közül a felhasználó választhat. Legtöbbször csak egy kiválasztott elemre van szükségünk, de lehetőségünk van több elem bejelölésének az engedélyezésére is. A vízszintes vagy függőleges görgetősáv segítségével a felhasználó könnyen szemügyre veheti a felajánlott elemeket.

A projektünk egy fő ablakból és hét darab fő keretből fog állni, amelyek csoportokba szervezve jelenítik meg a widgetjeinket:

- Az első keret nagyon kezdetleges lesz. Több különböző, a megjelenítési beállításokat mutató címkét fog tartalmazni.

- A második szintén elég egyszerű. A fenti beállításoknak a gombjait fogja tartalmazni.

- Ebben a keretben két jelölődobozunk van, melyeket az itt lévő másik gombbal tudunk átkapcsolni. Ezek amikor átállítódnak, elküldik az állapotukat (1 vagy 0) a terminál ablakba.

- Következőnek két, három rádió gombot tartalmazó csoportunk lesz, melyek üzenetet küldenek a terminálnak a rájuk való kattinásakor. A két csoport különálló.

- Ebben a már ismerős szöveg-dobozokból lesz néhány, de lesz egy olyan gomb is, amivel letilthatjuk/engedélyezhetjük az egyiket. Letiltott állapotban nem tudunk bele írni.

- Két gombot és egy olyan listát fog tartalmazni, ami üzenetet küld a terminálnak, ha egy elemét kiválasztották. Az egyik gomb törölni fogja a listát, a másik pedig töltelék

```
# widgetdemo1.py
# Labels
from Tkinter import *
```

```
class Demo:
    def __init__(self, master):
        self.DefineVars()
        f = self.BuildWidgets(master)
        self.PlaceWidgets(f)
```

értékekkel rakja tele.

- Az utolsó keret egy sor olyan gombot fog tartalmazni, melyek különböző típusú üzenő ablakokat jelenítenek meg.

Akkor lássunk is neki. A neve legyen "widgetdemo1.py". Bizonyosodjunk meg arról, hogy elmentettük, mert a projectet apró részletekben fogjuk elkészíteni és a későbbiekben ezekre a darabokra fogunk építkezni. Mindegyik darab a keretek valamelyikét fogja érinteni. Fel fog tűnni, hogy néhány megjegyzést is elhelyeztem, így tudni fogjátok, hogy hol mi történik. Következzen az első néhány sor.

Az első két sorban (megjegyzés) meg van adva az alkalmazás

neve és az érdeklődésünk tárgya. A harmadik sor egy import utasítás. Ezután definiáljuk az osztályunkat. A következő sor a már ismerős `__init__` rutine-unk - az újaknak mondom, hogy ez a kód az osztály példányosításakor fut le. A Toplevelt, azaz a gyöker ablakot fogjuk neki átadni. Az (eddig) utolsó három sorban, három különböző rutint hívunk meg. Az első (Define-Vars) néhány később használt változót állít be. A második (BuildWidgets) a widgetek előállításáért felel, a következőben (PlaceWidgets) a widgetek főablakba való elhelyezése történik. Mint múltkor, most is a grid geometria kezelőt fogjuk használni. Vegyük észre, hogy a BuildWidgets az "f" objektumot (a gyöker ablakot) adja vissza, amit a PlaceWidgetsnek továbbítunk.

Ez lenne a BuildWidgets rutinunk. A "self"-el kezdődő sorok két ok miatt lettek szétválasztva. Az első az, hogy jó szokás, ha a sorok hosszát 80 karakter vagy kevesebb alatt tartjuk. A második pedig, hogy a csodálatos szerkesztőnk segít vele. Két dolgot tehetünk. Vagy minden sort eredeti hosszán hagyunk, vagy megtartjuk ahogy itt van-

```
def BuildWidgets(self, master):
    # Define our widgets
    frame = Frame(master)
    # Labels
    self.lblframe = Frame(frame, relief = SUNKEN, padx = 3, pady = 3,
                           borderwidth = 2, width = 500)
    self.lbl1 = Label(self.lblframe, text="Flat Label", relief = FLAT,
                      width = 13, borderwidth = 2)
    self.lbl2 = Label(self.lblframe, text="Sunken Label", relief = SUNKEN,
                      width = 13, borderwidth = 2)
    self.lbl3 = Label(self.lblframe, text="Ridge Label", relief = RIDGE, width = 13,
                      borderwidth = 2)
    self.lbl4 = Label(self.lblframe, text="Raised Label", relief = RAISED,
                      width = 13, borderwidth = 2)
    self.lbl5 = Label(self.lblframe, text="Groove Label", relief = GROOVE,
                      width = 13, borderwidth = 2)

    return frame
```

nak. A Python lehetőséget ad a sorok megtörésére mindaddig, amíg a kérdéses részek (kapcsos) zárójelek között vannak. Mint ahogy már korábban is mondtam, mielőtt elhelyeznénk a widgeteket a griden, definiálni fogjuk őket. A következő rutinban láthatjuk majd, hogyan lehet az elhelyezéskor is létrehozni őket. Ennek ellenére, ha a példányosítást korábban tesszük meg, akkor jobban követhető lesz a kód, mivel a (legtöbb) definiálást amúgy is ebben a rutinban csináljuk meg.

Először létrehozzuk a fő keretet. Ide fogjuk elhelyezni az összes többi widgetet. Következőnek létrehozzuk az lblframe nevű gyermek keretét, ami öt darab címkét fog

tartalmazni. Itt be is állítjuk a keret különböző tulajdonságait. A relief 'SUNKEN' lesz, a padding (padx) balra és jobbra 3 pixelnyi, illetve szintén 3 a felső és alsó (pady) is. A keretszélességet (borderwidth) 2 pixelre állítjuk, hogy a bemélyedés láthatóbb legyen. Mivel alapértelmezésként a keretszélesség 0-ra van állítva, a relief módosítása nem látszana. Végül az össz szélességet 500 pixelnyire adjuk meg.

Ezután definiáljuk mindegyik címke widgetünket. A szülő a self.lblframe lesz (nem a frame), így mindegyik címke az lblframe-é, az lblframe pedig a frame fia. Figyeljük meg, hogy mind az öt definíció az elnevezésektől (lbl1, lbl2, stb.), a szövegektől és a kinézettől (relief)

eltekintve teljesen ugyanaz. Végül visszaadjuk a hívó rutinnak (`__init__`) a kész keretet.

A következő oldal jobb felső sarkában található a PlaceWidgets metódus.

A frame objektumot egy master nevű paraméterként kapjuk meg. Hogy következetesek legyünk a BuildWidgets rutinban lévőekkel, ezt a 'frame'-hez rendeljük hozzá. Ezután beállítjuk a fő gridünket: (frame.grid(column = 0, row = 0)). Ha ezt elmulasztjuk, akkor semmi sem fog helyesen működni. Ezt követően kezdjük a widgetek elhelyezését. Először a címkéinket tartalmazó kerettel (lblframe) foglalkozunk. A tulajdonságoknál a column

0, a row 1, a padding 5 pixelnyi minden oldalon, a span 5 oszlopnyi (bal és jobb), és végül a “sticky” tulajdonság segítségével széthúzzuk az egészet (“WE”, azaz nyugat és kelet). Most jön az a rész, ami egy kicsit ellentmond a szabályunknak. Egy olyan címkét fogunk elhelyezni, ami a keret első widgetje és korábban még nem hoztuk létre. Amikor elkészítjük, a szülő - mint a többinél is - a lblframe lesz. A text “Labels |”, a width 15, és az anchor east (kelet, ‘e’) lesz. Ha még emlékeztek az előző alkalomra, akkor tudhatjátok, hogy az anchorral megadhatjuk, hogy a widgeten belül hol jelenjen meg a szöveg. Ebben az esetben a jobb keret mentén fog látszódni. Most egy érdekes dolog jön. A grid pozíciót (és bármely más grid tulajdonságot is, amire szükségünk van) egyszerűen csak a “.grid” címkéhez való hozzáfűzéssel adjuk meg.

Következőnek elhelyezzük a griden az 1. oszloptól és 0. sortól kezdve az összes többi címkét.

Itt van a DefineVars rutin. Vegyük észre, hogy most csak a pass utasítást használjuk. Mivel most nincs szükségünk rá, ezért a későbbiekben fogjuk csak kitölteni:

```
def DefineVars(self):  
    # Define our  
    resources  
    pass
```

És végül elhelyezzük a fő rutinunk kódját is:

```
root = Tk()  
root.geometry('750x40+150+150')  
root.title("Widget  
Demo 1")  
demo = Demo(root)  
root.mainloop()
```

Először példányosítjuk a Tk-t. Ezután beállítjuk a fő ablak méretét 750 pixel szélesre és 40 pixel magasra, illetve a pozícióját a képernyő bal felső sarkától 150 pixelre. Ezután megadjuk az ablak nevét és létrehozuk a Demo objektumot. Végül meghívjuk a Tk mainloopját.

Próbáljuk meg futtatni. Öt címkét és az “utólag elhelyezett” feliratot kellene látnunk különböző látvány effektekkel.

Gombok

Mentsük el az eddigieket widgetdemo1a.py néven és csapjunk még hozzá néhány gombot. Mivel már eleve könnyen bővíthe-

```
def PlaceWidgets(self, master):  
    frame = master  
    # Place the widgets  
    frame.grid(column = 0, row = 0)  
    # Place the labels  
    self.lblframe.grid(column = 0, row = 1, padx = 5, pady = 5,  
                        columnspan = 5, sticky='WE')  
    l = Label(self.lblframe, text='Labels |', width=15,  
              anchor='e').grid(column=0, row=0)  
    self.lbl1.grid(column = 1, row = 0, padx = 3, pady = 5)  
    self.lbl2.grid(column = 2, row = 0, padx = 3, pady = 5)  
    self.lbl3.grid(column = 3, row = 0, padx = 3, pady = 5)  
    self.lbl4.grid(column = 4, row = 0, padx = 3, pady = 5)  
    self.lbl5.grid(column = 5, row = 0, padx = 3, pady = 5)
```

tőként építettük fel programunkat, ezért egyszerűen csak hozzá kell írunk a kérdéses részeket. Kezdjük a BuildWidgets rutinnal. A címke definíciók után, de még a “return frame” előtt helyezzük el amit a következő oldal jobb felső sarkában látunk.

Itt semmi újat nem találunk. Létrehoztuk a gombokat a tulajdonságaikkal együtt, és a .bindon keresztül beállítottuk a callbackjei-

ket. Vegyük észre, hogy a lamdát használtuk a megnyomott gombot azonosító értékek (1-5) átadásakor. Most a PlaceWidgets rutinnal fogunk foglalkozni. Helyezzük el az alábbi kódot közvetlenül a címkék hozzáadása után:

Itt sincs semmi érdekes. Egyszerűen csak egy sor elágazást használunk a kattintott gomb kiíratásához. A legfontosabb dolog (amikor a programot futtatjuk), hogy a be

```
# Place the buttons  
self.btnframe.grid(column=0, row = 2, padx = 5,  
                    pady = 5, columnspan = 5, sticky = 'WE')  
l = Label(self.btnframe, text='Buttons |', width=15,  
          anchor='e').grid(column=0, row=0)  
self.btn1.grid(column = 1, row = 0, padx = 3, pady = 3)  
self.btn2.grid(column = 2, row = 0, padx = 3, pady = 3)  
self.btn3.grid(column = 3, row = 0, padx = 3, pady = 3)  
self.btn4.grid(column = 4, row = 0, padx = 3, pady = 3)  
self.btn5.grid(column = 5, row = 0, padx = 3, pady = 3)
```

mélyedékes (sunken) gomb kattintáskor nem fog „mozogni”. Ezt a kinézetet általában csak akkor használjuk, ha azt akarjuk, hogy a gomb klikkelés után “lenn maradjon”. Végül egy kicsit módosítanunk kell a geometry utasítást, hogy az új widgeteket is támogassa:

```
root.geometry(
    '750x110+150+150')
```

Ezzel készen is vagyunk. Mentjük és futtassuk.

Miután elmentettük widgetdemo1b.py néven, folytathatjuk a jelölő dobozokkal.

Jelölődobozok

Ahogy már mondtam, a demónak ebben a részében egy hagyományos gomb és két jelölődoboz lesz. Az első doboz a szokásos kinézettel rendelkezik, a második pedig egy “ragadós” gomb szerűség lesz - utóbbi amikor nincs kiválasztva (avagy bejelölve), egy sima gomb kinézetét veszi fel, kiválasztáskor, pedig úgy néz ki, mintha beragadt volna. Mindezt az indicatoron attribútum False-ra való állításával érhetjük el. A “hagyományos” gombra való kattintás a jelölődo-

bozokat kapcsolja be és ki. Programozás ügyileg ehhez a dobozok .toggle metódusát használjuk. A bal egérgombbal való kattintás eseményét (egérgomb felengedése) egy olyan függvényhez rendeljük, amivel üzeneteket küldhetünk - ebben az esetben - a terminálra. Ezen felül még használunk két változót (egyet-egyét jelölődobozonként), amiket bármikor le tudunk kérdezni. Így minden alkalommal, amikor rájuk kattintunk, ezeket az értékeket kérjük le és írjuk ki. Mivel sok widget épül a kód változókkal foglalkozó részére, ezért fordítsunk ezekre kiemelt figyelmet.

A BuildWidgets rutinba - közvetlenül az előzőleg hozzáírt gombokat érintő kód és a return utasítás közé - helyezzük el a következő oldal jobb felső sarkában látható részt.

Ezzel is találkoztunk már korábban. Létrehozzuk a frame-et a widgetjeinknek. Beállítjuk a gombot és a két jelölő dobozt. Helyezzük is el ezeket (az előbbi kódrészlet allatti rész).

```
# Buttons
self.btnframe = Frame(frame, relief = SUNKEN, padx = 3, pady = 3,
    borderwidth = 2, width = 500)
self.btn1 = Button(self.btnframe, text="Flat Button",
    relief = FLAT, borderwidth = 2)
self.btn2 = Button(self.btnframe, text="Sunken Button",
    relief = SUNKEN, borderwidth = 2)
self.btn3 = Button(self.btnframe, text="Ridge Button",
    relief = RIDGE, borderwidth = 2)
self.btn4 = Button(self.btnframe, text="Raised Button",
    relief = RAISED, borderwidth = 2)
self.btn5 = Button(self.btnframe, text="Groove Button",
    relief = GROOVE, borderwidth = 2)
self.btn1.bind('<ButtonRelease-1>', lambda e: self.BtnCallback(1))
self.btn2.bind('<ButtonRelease-1>', lambda e: self.BtnCallback(2))
self.btn3.bind('<ButtonRelease-1>', lambda e: self.BtnCallback(3))
self.btn4.bind('<ButtonRelease-1>', lambda e: self.BtnCallback(4))
self.btn5.bind('<ButtonRelease-1>', lambda e: self.BtnCallback(5))
```

```
def BtnCallback(self, val):
    if val == 1:
        print("Flat Button Clicked...")
    elif val == 2:
        print("Sunken Button Clicked...")
    elif val == 3:
        print("Ridge Button Clicked...")
    elif val == 4:
        print("Raised Button Clicked...")
    elif val == 5:
        print("Groove Button Clicked...")
```

Most definiáljuk a már említett két változót is. A DefineVars alatt kommenteljük ki a pass utasítást és helyezzük el a következőt:

```
self.Chk1Val = IntVar()
self.Chk2Val = IntVar()
```

A gomb callback returnje után írjuk be a következő oldal jobb alsó lévő kódot.

És végül cseréljük le a geometry utasítást ezzel:

```
root.geometry('750x170+150+150')
```

Mentsünk és futtassunk. Mentsük el widgetdemo1c.py néven és térjünk át a rádió gombokra.

Rádió gombok

Ha elég idősek vagytok, akkor még láthattátok a csatorna választáshoz nyomógombokat használó autórádiókat. Ebben az esetben meg tudjátok érteni, hogy miért hívjuk ezeket rádió gomboknak. Rádió gombok használatakor kiemelten fontos a variable tulajdonság. Ezzel tudjuk a gombokat csoportokba szervezni. Ebben a demóban az első csoportot a self.RBVal tartja össze. A másodikat a self.RBValue2. Még arra is szükség van, hogy a value attribútumot tervezéskor állítsuk be, így biztosak lehetünk afelől, hogy a gombok a rájuk való kattintás után mindig a megfelelő értékkel térnek vissza.

Visszatérve a BuildWidgets rutinra, közvetlenül a return utasítás előtt helyezzük el a következő oldalon található kódot.

A PlaceWidgetsben pedig egy fontos dolgot kiemelnék. A címke definícióknál láthatjuk, hogy ho-

gyan használhatunk zárójeleket a hosszú sorok megformázására a kód elrontása nélkül.

A DefineVarsba a következő kerüljön:

```
self.RBVal = IntVar()
```

Helyezzük el a kattintás rutinokat:

```
def RBClick(self):
    print("Radio
    Button clicked -
    Value is
    {0}".format(self.RBVal.get()))
def RBClick2(self):
    print("Radio
    Button clicked -
    Value is
    {0}".format(self.RBValue2.get()))
```

```
def btnToggle(self,p1):
    self.chk1.toggle()
    self.chk2.toggle()
    print("Check box 1 value is {0}".format(self.Chk1Val.get()))
    print("Check box 2 value is {0}".format(self.Chk2Val.get()))

def ChkBoxClick(self,val):
    if val == 1:
        print("Check box 1 value is {0}".format(self.Chk1Val.get()))
    elif val == 2:
        print("Check box 2 value is {0}".format(self.Chk2Val.get()))
```

```
# Check Boxes
self.cbframe = Frame(frame, relief = SUNKEN, padx = 3, pady = 3,
                      borderwidth = 2, width = 500)
self.chk1 = Checkbutton(self.cbframe, text = "Normal Checkbox",
                        variable=self.Chk1Val)
self.chk2 = Checkbutton(self.cbframe, text = "Checkbox",
                        variable=self.Chk2Val,indicatoron = False)
self.chk1.bind('<ButtonRelease-1>',lambda e: self.ChkBoxClick(1))
self.chk2.bind('<ButtonRelease-1>',lambda e: self.ChkBoxClick(2))
self.btnToggleCB = Button(self.cbframe,text="Toggle Cbs")
self.btnToggleCB.bind('<ButtonRelease-1>',self.btnToggle)
```

```
# Place the Checkboxes and toggle button
self.cbframe.grid(column = 0, row = 3, padx = 5, pady = 5,
                  columnspan = 5,sticky = 'WE')
l = Label(self.cbframe,text='Check Boxes | ',width=15,
          anchor='e').grid(column=0,row=0)
self.btnToggleCB.grid(column = 1, row = 0, padx = 3, pady = 3)
self.chk1.grid(column = 2, row = 0, padx = 3, pady = 3)
self.chk2.grid(column = 3, row = 0, padx = 3, pady = 3)
```

Hogyanok – Programozzunk Pythonban – 26. rész

És végül a következő geometry utasítást.

```
root.geometry(
    '750x220+150+150')
```

Mentsük el widgetdemo1d.py-

ként és futtassuk. Mindezek után a hagyományos szövegdobozokat fogjuk megnézni.

Szövegdobozok

Már használtunk szövegdobozokat a különböző GUI-s megoldásokban. Most viszont azt is megmutatom, hogy hogyan lehet a szövegdoboz felhasználó általi módosítását megakadályozni annak

letiltásával. Ez akkor hasznos, ha meg akarunk mutatni valamit a felhasználónak, de csak "szerkesztés" módban engedélyezzük a módosítást. Mostan

```
# Radio Buttons
self.rbframe = Frame(frame, relief = SUNKEN, padx = 3, pady = 3, borderwidth = 2, width = 500)
self.rb1 = Radiobutton(self.rbframe, text = "Radio 1", variable = self.RBVal, value = 1)
self.rb2 = Radiobutton(self.rbframe, text = "Radio 2", variable = self.RBVal, value = 2)
self.rb3 = Radiobutton(self.rbframe, text = "Radio 3", variable = self.RBVal, value = 3)
self.rb1.bind('<ButtonRelease-1>', lambda e: self.RBClick())
self.rb2.bind('<ButtonRelease-1>', lambda e: self.RBClick())
self.rb3.bind('<ButtonRelease-1>', lambda e: self.RBClick())
self.rb4 = Radiobutton(self.rbframe, text = "Radio 4", variable = self.RBVal2, value = "1-1")
self.rb5 = Radiobutton(self.rbframe, text = "Radio 5", variable = self.RBVal2, value = "1-2")
self.rb6 = Radiobutton(self.rbframe, text = "Radio 6", variable = self.RBVal2, value = "1-3")
self.rb4.bind('<ButtonRelease-1>', lambda e: self.RBClick2())
self.rb5.bind('<ButtonRelease-1>', lambda e: self.RBClick2())
self.rb6.bind('<ButtonRelease-1>', lambda e: self.RBClick2())
```

A PlaceWidgetsben pedig ezt:

```
# Place the Radio Buttons and select the first one
self.rbframe.grid(column = 0, row = 4, padx = 5, pady = 5, columnspan = 5, sticky = 'WE')
l = Label(self.rbframe,
    text='Radio Buttons |',
    width=15, anchor='e').grid(column=0, row=0)
self.rb1.grid(column = 2, row = 0, padx = 3, pady = 3, sticky = 'EW')
self.rb2.grid(column = 3, row = 0, padx = 3, pady = 3, sticky = 'WE')
self.rb3.grid(column = 4, row = 0, padx = 3, pady = 3, sticky = 'WE')
self.RBVal.set("1")
l = Label(self.rbframe, text='| Another Set |',
    width = 15,
    anchor = 'e').grid(column = 5, row = 0)
self.rb4.grid(column = 6, row = 0)
self.rb5.grid(column = 7, row = 0)
self.rb6.grid(column = 8, row = 0)
self.RBVal2.set("1-1")
```

ra már biztosan tudjátok, hogy legelőször a BuildWidgets rutint kell kiegészítenünk.

Listbox

Ezután jön a listánk. Kezdjük a BuildWidgets-szel és a következő kóddal:

Mint mindig, most is létrehozuk a frame-ünket. Ezután jön a függőleges görgetősáv elkészítése. Erre azért van szükség a lista elkészítése előtt, mert hivatkoznunk kell a sáv '.set' metódusára. Vegyük észre a 'height=5' attribútumot. Ezzel azt mondjuk meg a listának, hogy legfeljebb 5 elemet mutathat egyszerre. A .bind utasításban a '<<ListboxSelect>>' -et használjuk eseményként. Ezt egy virtuális eseménynek nevezzük, mivel valójában nem egy tényleges eseményről van szó.

Most pedig a PlaceWidgets további kódja jön (következő oldalon balra).

Üzenet ablakok

Ez a rész egyszerűen csak egy sor "hagyományos" gombot fog tartalmazni, melyek a különböző

üzenet ablakokat hívják meg. Ezeket már használtuk egy másik GUI toolkiten keresztül. Öt különböző típusal fogunk foglalkozni (de van még több is). Ebben a részben az Info-ra, a Warningra, az Errorra, a Questionre és a Yes/No-ra fogunk kitérni. Ezek akkor nagyon hasznosak, amikor valamilyen információt akarunk látványosan a felhasználónak eljuttatni. Helyezzük a BuildWidgetsbe a következőket az alsó ábra alapján.

Itt van a segéd rutin. Az első három (Info, Warning és Error) esetében egyszerűen meghívjuk a 'tkMessageBox.showinfo'-t - vagy az épp odaillőt - a két paramétere segítségével. Az első az üzenet ablakának címsora, a második a ténylegesen megjele

```
# Textboxes
self.tbframe = Frame(frame, relief = SUNKEN, padx = 3, pady = 3, borderwidth = 2, width = 500)
self.txt1 = Entry(self.tbframe, width = 10)
self.txt2 = Entry(self.tbframe, disabledbackground="#cccccc", width = 10)
self.btnDisable = Button(self.tbframe, text = "Enable/Disable")
self.btnDisable.bind('<ButtonRelease-1>', self.btnDisableClick)
```

Majd jön a PlaceWidgets rutin:

```
# Place the Textboxes
self.tbframe.grid(column = 0, row = 5, padx = 5, pady = 5, columnspan = 5, sticky = 'WE')
l = Label(self.tbframe, text='Textboxes | ', width=15, anchor='e').grid(column=0, row=0)
self.txt1.grid(column = 2, row = 0, padx = 3, pady = 3)
self.txt2.grid(column = 3, row = 0, padx = 3, pady = 3)
self.btnDisable.grid(column = 1, row = 0, padx = 3, pady = 3)
```

A DefineVars végére ez a sor kerül:

```
self.Disabled = False
```

Most helyezzük el a gombra való kattintásra válaszoló eseményt:

```
def btnDisableClick(self, p1):
    if self.Disabled == False:
        self.Disabled = True
        self.txt2.configure(state='disabled')
    else:
        self.Disabled = False
        self.txt2.configure(state='normal')
```

És végül írjuk át a geometry utasítást:

```
root.geometry('750x270+150+150')
```

Mentsük widgetdemo1d.py néven és futtassuk.

```
# Place the Listbox and support buttons
self.lstframe.grid(column = 0, row = 6, padx = 5,
pady = 5, columnspan = 5, sticky = 'WE')
l = Label(self.lstframe, text='List Box |', width=15,
anchor='e').grid(column=0, row=0, rowspan=2)
self.lbox.grid(column = 2, row = 0, rowspan=2)
self.VScroll.grid(column = 3, row = 0, rowspan = 2,
sticky = 'NSW')
self.btnClearLBox.grid(column = 1, row = 0, padx =
5)
self.btnFillLBox.grid(column = 1, row = 1, padx =
5)
```

A DefineVarsba a következő kell...

```
# List for List box items
self.examples = ['Item One', 'Item Two', 'Item
Three', 'Item Four']
```

És az alábbi segéd rutinok:

```
def ClearList(self):
    self.lbox.delete(0, END)

def FillList(self):
    # Note, clear the listbox first...no check is done
    for ex in self.examples:
        self.lbox.insert(END, ex)
    # insert([0, ACTIVE, END], item)

def LBoxSelect(self, p1):
    print("Listbox Item clicked")
    items = self.lbox.curselection()
    selitem = items[0]
    print("Index of selected item =
{0}".format(selitem))
    print("Text of selected item =
{0}".format(self.lbox.get(selitem)))
```

Végül frissítjük a geometry sort.

```
root.geometry('750x370+150+150')
```

Mentsük el widgetdemo1e.py néven és futtassuk. Most az utolsó módosításokra fog sor kerülni.

```
# List Box Stuff
self.lstframe = Frame(frame,
    relief = SUNKEN,
    padx = 3,
    pady = 3,
    borderwidth = 2,
    width = 500
)
# Scrollbar for list box
self.VScroll = Scrollbar(self.lstframe)
self.lbox = Listbox(self.lstframe,
    height = 5,
    yscrollcommand = self.VScroll.set)
# default height is 10

self.lbox.bind('<<ListboxSelect>>', self.LBoxSelect)
self.VScroll.config(command =
self.lbox.yview)
self.btnClearLBox = Button(
    self.lstframe,
    text = "Clear List",
    command = self.ClearList,
    width = 11
)
self.btnFillLBox = Button(
    self.lstframe,
    text = "Fill List",
    command = self.FillList,
    width = 11
)
# <<ListboxSelect>> is virtual event
# Fill the list box
self.FillList()
```

Hogyanok – Programozzunk Pythonban – 26. rész

nítendő üzenet. Az ikont a tkinter elintézi helyettünk. Azoknál az ablakoknál, amelyek választ is szolgáltatnak (question, yes/no), egy változót használunk a megnyomott gomb értékének tárolására. A question ablak esetében a válasz vagy "yes" ("igen") vagy "no" ("nem"), illetve a yes/no-nál, a kapott érték vagy "True" vagy "False".

Végül módosítjuk a geometry sort is:

```
root.geometry('750x490+550+150')
```

Mentsük widgetdemo1f.py-ként és játszaddozzuk el egy kicsit vele.

A widgetdemo1f.py kódja a pastebin oldalán található:

<http://pastebin.com/ZqrgHcdG>

Erre az alkalomra ennyi. Remélem elég ösztönzőleg hatott a tkinter többi nyaláncságainak

felfedezéséhez. Találkozzunk legközelebb is!

```
def ShowMessageBox(self, which):
    if which == 1:
        tkMessageBox.showinfo('Demo', 'This is an INFO messagebox')
    elif which == 2:
        tkMessageBox.showwarning('Demo', 'This is a WARNING messagebox')
    elif which == 3:
        tkMessageBox.showerror('Demo', 'This is an ERROR messagebox')
    elif which == 4:
        resp = tkMessageBox.askquestion('Demo', 'This is a QUESTION messagebox?')
        print('{0} was pressed...'.format(resp))
    elif which == 5:
        resp = tkMessageBox.askyesno('Demo', 'This is a YES/NO messagebox')
        print('{0} was pressed...'.format(resp))
```

```
# Buttons to show message boxes and dialogs
self.mbframe = Frame(frame, relief = SUNKEN, padx = 3, pady = 3, borderwidth = 2)
self.btnMBInfo = Button(self.mbframe, text = "Info")
self.btnMBWarning = Button(self.mbframe, text = "Warning")
self.btnMBError = Button(self.mbframe, text = "Error")
self.btnMBQuestion = Button(self.mbframe, text = "Question")
self.btnMBYesNo = Button(self.mbframe, text = "Yes/No")
self.btnMBInfo.bind('<ButtonRelease-1>', lambda e: self.ShowMessageBox(1))
self.btnMBWarning.bind('<ButtonRelease-1>', lambda e: self.ShowMessageBox(2))
self.btnMBError.bind('<ButtonRelease-1>', lambda e: self.ShowMessageBox(3))
self.btnMBQuestion.bind('<ButtonRelease-1>', lambda e: self.ShowMessageBox(4))
self.btnMBYesNo.bind('<ButtonRelease-1>', lambda e: self.ShowMessageBox(5))
```

Most helyezzük el a PlaceWidgets kódját:

```
# MessageBox buttons and frame
self.mbframe.grid(column = 0, row = 7, columnspan = 5, padx = 5, sticky = 'WE')
l = Label(self.mbframe, text='Message Boxes |', width=15, anchor='e').grid(column=0, row=0)
self.btnMBInfo.grid(column = 1, row = 0, padx = 3)
self.btnMBWarning.grid(column = 2, row = 0, padx = 3)
self.btnMBError.grid(column = 3, row = 0, padx = 3)
self.btnMBQuestion.grid(column = 4, row = 0, padx = 3)
self.btnMBYesNo.grid(column = 5, row = 0, padx = 3)
```



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.



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 rovatához 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.

