Full Circle

AZ UBUNTU LINUX KÖZÖSSÉG FÜGGETLEN MAGAZINJA

Programozói sorozat – Különkiadás

Programozói sorozat Különkiadás

PROGRAMOZZUNK PYTHONBAN

3. Kötet

A Full Circle magazin különkiadása

Full Circle

AZ UBUNTU LINUX KÖZÖSSÉG FÜGGETLEN MAGAZINJA

Programozzunk Pythonban
17. rész **3. oldal**

Programozzunk Pythonban
18. rész **6. oldal**

Programozzunk Pythonban
19. rész **9. oldal**

Üdvözöllek egy újabb, „egyetlen témáról szóló különkiadásban”

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

Most ez a **„Programozzunk Pythonban” 17.-21. részének** az újabb kiadása (a magazin 43.-47. számaiból), semmi extra, csak a tények.

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

Jó szórakozást!

Programozzunk Pythonban
20. rész **11. oldal**

Programozzunk Pythonban
21. rész **16. oldal**

Hogyanok

Programozzunk Pythonban – 17. rész

Írta: Greg Walters

Épp az utolsó simításokat végeztem sorozatunk utolsó darabján, amikor egy programozóversenyről szóló e-mailt kaptam. Sajnos nincs már időnk ezzel foglalkozni, de különböző oldalak egész évben lehetőséget nyújtanak a versenyzésre. Ha érdekel, a versenykiírást a http://www.freiesmagazin.de/third_programming_contest oldalon találhatod meg. Ez azonban felhívta a figyelmemet arra a hiányosságra, hogy nem nagyon foglalkoztunk szerver-kliens programozással. Ezért most elmélyedünk egy kicsit ebben a témában is, és meglátjuk, hogy hova lyukadunk ki.

Nos, milyen is egy szerver-kliens alkalmazás? A legegyszerűbben megfogalmazni talán így lehetne: ha egy olyan programot (vagy egy webes interfészt) használunk, ami adatokat kér egy másik alkalmazástól vagy számítógéptől, akkor az szerver-kliens felépítésű. Nézzünk meg egy már ismerős példát. Emlékszünk még a szakácskönyves programra? Az egy NAGYON egyszerű (és nem is túl jó) példája a szerver-kliens felépítésnek. Az SQLite adatbázis gyakorlatilag egy szerver, az alkalmazás, amit mi írtunk pedig egy kliens. A következő viszont ennél egy jobb példa. Az irodánk valamely távoli részében lévő számítógépen van egy adatbázisunk. Ez adatokat tárol az üzletünk készletéről. Ha egy POS terminált használunk (10-ből 1-et), akkor minden kassza egy kliens, és az előbb említett adatbázis a szerver.

Mi nem fogunk most ilyen szintű rendszert létrehozni, de az alapokat elsajátíthatjuk.

Az első dolog, amit meg kell terveznünk, az az adatbázis helye. Sokaknak csak egy számítógépük van otthon. Némelyeknek pedig akár hét vagy nyolc.

Ahhoz, hogy egy szerver-kliens rendszert használni tudjunk, a kliens gépről rá kell csatlakozni a kiszolgáló gépre. Ezt socketekkel valósítjuk meg. Ha gyerekkorunkban csináltunk konzervdobozokból telefont, akkor az egészről már lehet valamilyen elképzelésünk. Ha mégsem, akkor hadd írjam le ezt a régmúlt dolgot. Először is, anyukánkat meg kellet kérni, hogy rakjon el két babos, vagy valami hasonló konzervdobozt, melyeket óvatosan megtisztítottunk, majd kivittük őket a garázsba. Apró szögekkel és egy kalapáccsal lyukat ütöttünk az aljukra és szereztünk egy 15 láb hosszú spárgát (ismét csak anyutól), ezt követően áthúztuk mindkét a dobozon és - hogy ki ne szaladjanak - kötöttünk egy nagy csomót mindkét végére. Ha elkészültünk, akkor elhívtuk legjobb barátunkat és valamelyikünk az egyik dobozba üvöltött amíg a másik a füléhez tartotta a párját. A vibráció a kifeszített spárgán keresztül átment az egyik konzervből a másikba. Természetesen hallottuk egymást a dobozok nélkül is, de az akkor teljesen lényegtelen volt és nagyon is élveztük. A socket kábé ennek a megfelelője. A kliens közvetlen kapcsolatban áll a szerverrel (gondoljunk a spárgára). Ha sok ügyfél kapcsolódik a szerverre, akkor mindegyik kliensnek saját konzervdoboza lenne, melyhez ugyanennyi kapcsolódik a kiszolgálónál is. A lényeg az az, hogy mindegyik kliensnek közvetlen kapcsolata van a szerverrel.

Készítsünk egy egyszerű szervert és klienst. Először a kiszolgálóval kezdünk. Pszeudokódban leírva a következőt kell tennünk:

Socket létrehozása
Szerver nevének lekérése
Port kiválasztása
Socket csatolása a címhez és porthoz
Várakozás kapcsolatra
Ha csatlakoztunk...
    Csatlakozás elfogadása
    Kapcsolat kiírása
    Kapcsolat lezárása

A szerver konkrét kódját a következő oldal bal alsó részén láthatjuk.

Szóval, létre hozzuk a socketet, lekérjük a kiszolgáló hostname-jét, hozzácsatoljuk a socketet a porthoz és bejövő kapcsolatra várakozunk. Amikor kapunk egy kapcsolat kérést, kiíratjuk, hogy csatlakoztunk és elküldjük a “Hello and Goodbye” üzenetet, illetve lezárjuk a kapcsolatot.

Most már csak egy kliensre van szükségünk, hogy működjön az egész (jobbra lenn).

A kód majdnem olyan mint a szerveré, csak ebben az esetben kiíratjuk a kapott üzenetet, majd lezárjuk a kapcsolatot.

A programok kimenete könnyen kitalálható. A szerver oldalán a következőt kapjuk:

```
My hostname is  earth

I'm now connected to  ('127.0.1.1', 45879)
```

és a kliensné:

```
Hello and Goodbye
```

Tehát elég egyértelmű. Most csináljunk valami valósabbat. Hozzunk létre egy olyan szervert, ami csinál is valamit. A szerver 2-es verziójának kódja itt található: http://fullcirclemagazine.pastebin.com/Az8vNUv7

Nézzük meg egy kicsit közelebbről. Az import utasítások után létrehozunk néhány változót. A BUFSIZ tartalmazza az ügyféltől kapott adatokat tároló puffer méretét. Ezután beállítjuk a figyelni kívánt port számát és egy listát, ami a kiszolgálót és annak portját tartalmazza.

Létrehozzuk a ServCmd osztályt. Az __init__ rutinban elkészítjük a socketet és hozzácsatoljuk az interfészt ehhez a sockethez. A run metódusban elkezdjük figyelni a bejövő kapcsolatokat és várunk a kliens utasításaira.

Az os.popen() rutint akkor használjuk, amikor kapunk valami utasítást az ügyféltől. Ez létrehoz egy parancsablakot és lefuttatja az utasításokat.

```
#!/usr/bin/env python
#server1.py
import socket
soc = socket.socket()
hostname = socket.gethostname()
print "My hostname is ", hostname
port = 21000
soc.bind((hostname,port))
soc.listen(5)
while True:
     con,address = soc.accept()
     print "I'm now connected to ",address
     con.send("Hello and Goodbye")
     con.close()
```

```
#!/usr/bin/env python
# client2.py

from socket import *
from time import time
from time import sleep
import sys
BUFSIZE = 4096

class CmdLine:
    def __init__(self,host):
        self.HOST = host
        self.PORT = 29876
        self.ADDR = (self.HOST,self.PORT)
        self.sock = None

    def makeConnection(self):
        self.sock = socket( AF_INET,SOCK_STREAM)
        self.sock.connect(self.ADDR)

    def sendCmd(self, cmd):
        self.sock.send(cmd)

    def getResults(self):
        data = self.sock.recv(BUFSIZE)
        print data

if __name__ == '__main__':
    conn = CmdLine('localhost')
    conn.makeConnection()
    conn.sendCmd('ls -al')
    conn.getResults()
    conn.sendCmd('BYE')
```

```
#!/usr/bin/python
# client1.py
#====================
import socket

soc = socket.socket()
hostname = socket.gethostname()
port = 21000

soc.connect((hostname, port))
print soc.recv(1024)
soc.close
```

A kliens (jobbra fenn) sokkal egyszerűbb. Most csak a küldést nézzük meg (a többit már mi magunk is megértjük). A **conn.sendCmd()** sor (31.) elküld egy egyszerű **ls -al** kérést. Itt van az erre adott válasz kimenete (a tietek ettől eltérhet):

Szerver:

```
python server2.py
...listening
...connected:  ('127.0.0.1', 42198)
Command received -  ls -al
Command received -  BYE
...listening
```

Kliens:

```
python client2a.py
total 72
drwxr-xr-x 2 greg greg 4096 2010-
11-08 05:49 .
drwxr-xr-x 5 greg greg 4096 2010-
11-04 06:29 ..
-rw-r--r-- 1 greg greg  751 2010-
11-08 05:31 client2a.py
-rw-r--r-- 1 greg greg  760 2010-
11-08 05:28 client2a.py~
-rw-r--r-- 1 greg greg  737 2010-
11-08 05:25 client2.py
-rw-r--r-- 1 greg greg  733 2010-
11-08 04:37 client2.py~
-rw-r--r-- 1 greg greg 1595 2010-
11-08 05:30 client2.pyc
-rw-r--r-- 1 greg greg  449 2010-
11-07 07:38 ping2.py
-rw-r--r-- 1 greg greg  466 2010-
11-07 10:01 python_client1.py
-rw-r--r-- 1 greg greg  466 2010-
11-07 10:01 python_client1.py~
-rw-r--r-- 1 greg greg  691 2010-
11-07 09:51 python_server1.py
-rw-r--r-- 1 greg greg  666 2010-
11-06 06:57 python_server1.py~
-rw-r--r-- 1 greg greg  445 2010-
11-04 06:29 re-test1.py
-rw-r--r-- 1 greg greg 1318 2010-
11-08 05:49 server2a.py
-rw-r--r-- 1 greg greg 1302 2010-
11-08 05:30 server2a.py~
-rw-r--r-- 1 greg greg 1268 2010-
11-06 08:02 server2.py
-rw-r--r-- 1 greg greg 1445 2010-
11-06 07:50 server2.py~
-rw-r--r-- 1 greg greg 2279 2010-
11-08 05:30 server2.pyc
```

Egy másik gépről is rácsatlakozhatunk anélkül, hogy bármit is módosítanánk - kivéve a kliens kódjában a conn = CmdLine("localhost") (29) sort. Ebben az esetben a "localhost"-ot kell valamilyen IP címre átírni. Az otthoni hálózatomban ez a következőképpen alakult:

```
conn = CmdLine('192.168.2.12')
```

Nos, most már képesek vagyunk üzenetek küldözgetésére egyik gépről (vagy terminálról) a másikra.

Következő alkalommal tovább bővítjük szerver-kliens alkalmazásunkat.

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

Hogyanok

Írta: Greg Walters

Programozzunk Pythonban – 18. rész

Legutóbb egy nagyon egyszerű szerver-kliens alkalmazást készítettünk. Ezt fogjuk most egy kicsit kibővíteni. A kiszolgáló egy tic-tac-toe (avagy egy 3x3-as amőba) tábla kezelője lesz, az ügyfél pedig csak I/O-val fog foglalkozni.

Kezdésnek vegyük elő a múltkori szerver kódját. Ezt fogjuk a cikk folyamán módosítgatni. Ha nem lenne meg a kód, akkor a http://fullcirclemagazine.pastebin.com/UhquVK4N oldalról töltsétek le az e havi kódot és kövessétek a módosításokat ezen. Az első változás az __init__ rutinban lesz, ahol két új változót fogunk inicializálni: a self.playert és a self.gameboardot. A gameboard (tábla) egy egyszerű listák listája, avagy egy hagyományos tömb. Az elemeket a következőképpen tudjuk elérni (így látványosabb mint egy egyszeri lista). Ez a lista fogja az adatainkat tartalmazni. Minden cellában három fajta bejegyzés lehetséges: a „-” azt jelenti, hogy a cella üres, az „X” azt, hogy az első játékos foglalta el, a „O” pedig azt, hogy a második játékosé. A táblázat a következő képpen néz ki két dimenzióban:

```
[0][0] | [0][1] | [0][2]
[1][0] | [1][1] | [1][2]
[2][0] | [2][1] | [2][2]
```

Szóval a szerver __init__ rutinját a következő sorokkal kell kiegészíteni:

```
# The next three lines are
new...

    self.player = 1

    self.gameboard = [['-','-
','-'],['-','-','-'],['-','-
','-']]

    self.run()
```

A run, a listen és a servCmd rutinban nem lesz változás, ehelyett a procCmd metódusra fogunk koncentrálni.

Az előző alkalommal a szerver egy parancsra várakozott a klienstől, majd azt átadta az os.popen rutinnak. Most azonban elemezni fogjuk a kapott parancsot. Ebben az esetben három utasítás lehetséges. Ezek a ‘Start’, a ‘Move' és a ‘GOODBYE’. Amikor a ‘Start’-ot kapjuk, az azt jelenti, hogy a szervernek inicializálnia kell a játékteret „-”-ra, majd el kell küldenie egy „kiíratást” a kliensnek.

A ‘Move’ egy összetett utasítás, ami tartalmazza magát a parancsot és a játékos által kiszemelt pozíciót. Például: ‘Move A3’. Az utasítást úgy kell feldarabolnunk, hogy három részt kapjunk: a ‘move’ szót, illetve a sor és az oszlop azonosítóit. Végül a ‘GOODBYE’ egyszerűen előkészíti a táblát egy újabb menetre.

Nos, megkaptuk az ügyfél utasítását a procCmd-ben. Ezután meg kell néznünk, hogy mit is kellene csinálni. A procCmd metódusban keressük meg az ötödik sort és az „if self.processingloop” után töröljük ki az összes kódot. Itt van a Start utasításhoz tartozó rész:

```
if self.processingloop:
  if cmd == 'Start':
    self.InitGameBoard()
    self.PrintGameBoard(1)
```

Következőnek nézzük meg a Move-hoz tartozó kódot (jobbra látható). Először az első négy karakterét ellenőrizzük le az átadott utasításnak. Ha ez a ‘Move’-al egyezik meg, akkor kivesszük a maradék részt az 5-ös pozíciótól kezdődően (mivel 0-tól számoljuk az indexeket), és egy position nevű változóhoz rendeljük hozzá őket. Ezután leellenőrizzük, hogy az első karakter ‘A’, ‘B’, vagy ‘C’-e. Ezek a klienstől kapott sorokat reprezentálják. A következő karakter amit kiszedünk, az az oszlopokat azonosító egész lesz:

```
if cmd[:4] == 'Move':
print "MOVE COMMAND"
position = cmd[5:]
if position[0] == 'A':
    row = 0
elif position[0] == 'B':
    row = 1

elif position[0] == 'C':
    row = 2
else:
    self.cli.send('Invalid
position')
    return
col = int(position[1])-1
```

Ezt követően gyorsan leellenőrizzük, hogy a sor benne van-e a megengedett tartományban:

```
if row < 0 or row > 2:
    self.cli.send('Invalid position')
    return
```

Végül megnézzük, hogy a pozíció üres-e („-”), majd ha az első játékossal van dolgunk, akkor beírjuk ide az „X”-et, különben pedig az „O”-t. Ezután meghívjuk a PrintGameBoard rutint a ”0” paraméterrel:

```
if self.gameboard[row][col] == '-':
    if self.player == 1:

        self.gameboard[row][col] = "X"

else:
        self.gameboard[row][col] = "O"

self.PrintGameBoard(0)
```

Ezzel be is fejeztük a procCmd-t. Következőnek a „játékteret inicializáló” rutint kell elkészítenünk. Mindössze annyi dolga lesz, hogy mindenhova elhelyezi a „-” jelet, amit a move logikája fog a cella üresség-énak vizsgálatára felhasználni:

```
def InitGameBoard(self):

    self.gameboard = [['-','-','-'],['-','-','-'],['-','-','-']]
```

A PrintGameBoard rutin (lenn) kiíratja a táblát, majd meghívja a checkwin rutint és beállítja a játékost. Egy nagy sztringet hozunk létre, melyet elküldünk a kliensnek, így annak minden lépés alkalmával csak egyszer kell hallgatóznia. A firsttime paraméter a tábla kiíratása miatt lesz elküldve, amikor a kliens először kapcsolódik vagy reseteli a játékot:

Ezután leellenőrizzük, hogy a firsttime paraméter 0-ra vagy 1-re van-e állítva (lenn). Akkor fogjuk csak megnézni, hogy az adott játékos nyert-e, ha a firsttime 0 értékű. Ebben az esetben a *Player X WINS!* üzenetet íratjuk ki. Ha az aktuális játékos nem nyert, akkor az “Enter move...” fog a képernyőre kerülni. Végül elküldjük a kliensnek a karakterláncot a cli.send rutinnal.

A következő oldalon látható, hogy a szerver lefuttatja a win rutint. Itt már beállítottuk a játékost “X”-re, vagy “O”-ra, szóval egy egyszerű ciklust kell már csak használnunk. A ‘C’ ciklusváltozó a listánk egyes sorait reprezentálja. Először le kell ellenőrizzük az összes sort egy vízszintes nyerésre.

```
if firsttime == 0:
            if self.player == 1:
                ret = self.checkwin("X")
            else:
                ret = self.checkwin("O")
            if ret == True:
                if self.player == 1:
                    outp += "Player 1 WINS!"
                else:
                    outp += "Player 2 WINS!"
            else:
                if self.player == 1:
                    self.player = 2
                else:
                    self.player = 1
                outp += ('Enter move for player %s' % self.player)
        self.cli.send(outp)
```

```
def PrintGameBoard(self,firsttime):
    #Print the header row
    outp = ('   1   2   3') + chr(13) + chr(10)
    outp += (" A {0} | {1} | {2}".format(self.gameboard[0][0],self.gameboard[0][1],self.gameboard[0][2])) + chr(13)+chr(10)
    outp += ('  ------------')+ chr(13)+chr(10)
    outp += (" B {0} | {1} | {2}".format(self.gameboard[1][0],self.gameboard[1][1],self.gameboard[1][2]))+ chr(13)+chr(10)
    outp += ('  ------------')+ chr(13)+chr(10)
    outp += (" C {0} | {1} | {2}".format(self.gameboard[2][0],self.gameboard[2][1],self.gameboard[2][2]))+ chr(13)+chr(10)
    outp += ('  ------------')+ chr(13)+chr(10)
```

Először ellenőrzünk minden SOR-t a vízszintes nyerésért:

```
    def checkwin(self,player):
        #loop through rows and columns
        for c in range(0,3):
        #check for horizontal line
            if self.gameboard[c][0] == player and
self.gameboard[c][1] == player and self.gameboard[c][2] ==
player:
                print "*********\n\n%s wins\n\n*********" %
player
                playerwin = True
                return playerwin
```

Utána minden OSZLOP-ot:

```
            #check for vertical line
            elif self.gameboard[0][c] == player and
self.gameboard[1][c] == player and self.gameboard[2][c] ==
player:
                print "** %s wins **" % player
                playerwin = True
                return playerwin
```

Most az ÁTLÓS nyeréseket ellenőrizzuk balról jobbra ...

```
            #check for diagonal win (left to right)
            elif self.gameboard[0][0] == player and
self.gameboard[1][1] == player and self.gameboard[2][2] ==
player:
                print "** %s wins **" % player
                playerwin = True
                return playerwin
```

Aztán jobbról balra...

```
            #check for diagonal win (right to left)
            elif self.gameboard[0][2] == player and
self.gameboard[1][1] == player and self.gameboard[2][0] ==
player:
                print "** %s wins **" % player
                playerwin = True
                return playerwin
```

Végül, ha nem volt nyerés akkor "HAMIS" értékkel térunk vissza:

```
        else:
            playerwin = False
            return playerwin
```

A kliens

Ismét a múltkori rutinnal kezdünk. A változások közvetlenül a conn.makeConnection után kezdődnek. Elküldjük a Startot, majd a különböző Move-okat és végül a Goodbye utasítást. Innen a legfontosabb dolog, hogy csak egy utasítást kell elküldenünk. Gondoljunk úgy erre, mint egy udvarias társalgásra. Mond ki az állításod, majd várj egy válaszra, és így tovább. Ebben az egyszerű példában ahhoz, hogy jobban megértsük a lényeget, a raw_inputot csak elvétve fogjuk használni:

```
if __name__ == '__main__':
    conn = CmdLine('local-
host')
    conn.makeConnection()
    conn.sendCmd('Start')
    conn.getResults()
    conn.sendCmd('Move A3')
    conn.getResults()
    r = raw_input("Press
Enter")
    conn.sendCmd('Move B2')
    conn.getResults()
    r = raw_input("Press
Enter")
```

Folytassuk az egészet a sendCmd, getResults, raw_input sorozat alábbi paramétereivel (az A3-hoz és B2-höz már megvan a kód): C1, A1, C3, B3, C2 és végül küldjük el a GOODBYE utasítást.

Házi feladat

A kliens alkalmazásból vegyük ki a move utasításokat és helyette használjuk a raw_input()-ot a játékos lépéseinek “A3” vagy “B2” formátumban bekéréséhez, majd – mielőtt elküldenénk a szervernek – kapcsoljuk őket a “Move” utasításhoz.

Következő alkalommal úgy módosítjuk a szervert, hogy az a másik játékos helyett játsszon.

A szerver-kliens teljes kódja a http://fullcirclemagazine.pastebin.com/UhquVK4N vagy a http://thedesignatedgeek.com oldalon található meg.

Greg Walters a RainyDay Solutions Kft. tulajdonosa, amely egy tanács-adó cég a coloradói Aurórában. Greg 1972 óta foglalkozik programozás-sal. Szeret főzni, túrázni, zenét hallgatni, valamint a szabadidejét család-jával tölteni.

Programozzunk Pythonban – 19. rész

Írta: Greg Walters

Ez alkalommal a Tic-Tac-Toe programunk befejezésén fogunk dolgozni. A legtöbb cikkemmel ellentétben most nem fogom odaadni a kódot. Nektek kell azt elkészíteni, ahogy én azt előre megszabom. 18 hónap után már kezetekben vannak az eszközeitek és a tudásotok egy ilyen projekt befejezéséhez.

Először nézzük meg a Tic-Tac-Toe játékmenetét. Ezt pszeudó kódban tesszük meg. Előtte viszont vessünk még egy pillantást a játéktérre. Valahogy így néz ki:

```
Sarok |Oldal      |Sarok
------+--------+------
Oldal |Közép     |Oldal
------+--------+------
Sarok |Oldal      |Sarok
```

Mindig az „X” játékos kezd. A legjobb, amit tehet az, hogy egy sarkot foglal el, hogy melyiket, nem számít, bármelyik sarok megteszi. Az „X” játékos lépéseinek permutációival fogunk először foglalkozni. (jobbra).Az „O” játékos állás-pontja jobbra lenn látható.

```
IF “O” takes a CORNER square THEN
     # Scenario 1
     “X” should take one of the remaining corner squares. Doesn't matter which.
     IF “O” blocks the win THEN
          “X” takes remaining corner square.
          Finish for win.
     ELSE
          Finish for win.
ELIF “O” takes a SIDE square THEN
     # Scenario 2
     “X” takes CENTER square
     IF “O” blocks win THEN
          “X” takes corner square that is not bordered by any “O”
          Finish for win.
     ELSE
          Finish for win.
ELSE
     # “O” has played in the CENTER square – Scenario 3
     “X” takes corner square diagonally to original move
     IF “O” plays on corner square
          “X” plays remaining open corner square
          Finish for win.
     ELSE
          # Game will be a draw – Scenario 4
          Block “O” win.
          Block any other possible wins
          Draw Game.
```

Néhány lehetséges lejátszás található a következő oldalon.

Feltűnhet, hogy a logika egy kissé bonyolult, de könnyen lebontható IF utasításokra (figyeljétek meg, hogy „Then”-t használtam, de Pythonba ehelyett „:” van). Most már tudnotok kellene, hogy miként módosíthatjátok a múlt havi kódot, vagy hogyan írhattok egy teljesen új asztali tic-tac-toe-t.

```
IF “X” plays to non-center square
THEN
     “O” takes Center Square
     IF “X” has corner square AND
side square THEN
          #Scenario 5
          “O” takes corner diagonally
from corner “X”
          Block possible wins to a
draw.
     ELSE
          # “X” has two Edge squares
– Scenario 6
          “O” moves to corner
bordered by both “X”s
          IF “X” blocks win THEN
               “O” takes any square.
               Block and force draw
          ELSE
               Finish for win.
```

```
Scenario 1
 X | - | -    X | - | -    X | - | -    X | - | -    X | - | X    X | - | X    X | X | X
 - | - | -    - | - | -    - | - | -    O | - | -    O | - | -    O | O | -    O | O | -
 - | - | -    - | - | O    X | - | O    X | - | O    X | - | O    X | - | O    X | - | O

Scenario 2
 X | - | -    X | - | -    X | - | -    X | - | -    X | - | X    X | - | X    X | X | X
 - | - | -    O | - | -    O | X | -    O | X | -    O | X | -    O | X | -    O | X | -
 - | - | -    - | - | -    - | - | -    - | - | O    - | - | O    O | - | O    X | - | O

Scenario 3
 X | - | -    X | - | -    X | - | -    X | - | X    X | O | X    X | O | X    X | O | X
 - | - | -    - | O | -    - | O | -    - | O | -    - | O | -    - | O | -    - | O | X
 - | - | -    - | - | -    - | - | X    O | - | X    O | - | X    O | - | X    O | - | X

Scenario 4
 X | - | -    X | - | -    X | - | -    X | - | -    X | - | -    X | - | X    X | O | X
 - | - | -    - | O | -    - | O | O    X | O | O    X | O | O    X | O | O    X | O | O
 - | - | -    - | - | -    - | - | X    - | - | X    O | - | X    O | - | X    O | - | X

Scenario 5
 X | - | -    X | - | -    X | - | -    X | - | -    X | - | -    X | - | -    X | - | X
 - | - | -    - | O | -    - | O | X    - | O | X    X | O | X    X | O | X    X | O | X
 - | - | -    - | - | -    - | - | -    - | - | O    - | - | O    O | - | O    O | - | O

Scenario 6
 - | - | -    - | - | -    - | - | -    - | - | -    - | - | X    O | - | X    O | - | X
 X | - | -    X | O | -    X | O | -    X | O | -    X | O | -    X | O | -    X | O | -
 - | - | -    - | - | -    - | X | -    O | X | -    O | X | -    O | X | -    O | X | O
```

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.

Hogyanok

Írta: Greg Walters

Programozzunk Pythonban – 20. rész

Üdvözöllek! Ez alkalommal újra a GUI-programozással fogunk foglalkozni, de most a pyGTK könyvtárat fogjuk használni. Egyelőre nem használunk GUI-tervezőt, egyszerűen csak a könyvtárral dolgozunk.

A Synaptic segítségével telepítsük fel a python-gtk2, python-gtk2-tutorial és python-gtk2-doc csomagokat.

Vágjunk is a közepébe, és hozzuk létre első pyGTK-t használó programunkat, ami jobbra fenn látható.

Egy darabig erre az egyszerű kódkészletre fogunk építkezni. A harmadik sorban van is egy új utasítás. A „pygtk.require('2.0')” azt jelenti, hogy a programunk nem fog a pygtk modul legalább 2.0-s verziója nélkül futni. Az __init__ rutinban egy ablakot rendelünk a self.window változóhoz (8. sor), majd kirajzoljuk azt (9. sor). Emlékeztetőül: az __init__ rutin lefut, amint példányosítjuk az osztályt (13. sor). Mentsük el a kódot „simple1.py” néven.

```
# simple.py
import pygtk
pygtk.require('2.0')
import gtk

class Simple:
     def __init__(self):
          self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
          self.window.show()
     def main(self):
          gtk.main()
if __name__ == "__main__":
     simple = Simple()
     simple.main()
```

Futtassuk terminálban. Egy egyszerű ablak fog feltűnni valahol az asztalunkon. Az enyémen a bal felső sarokban jelenik meg. A program befejezéséhez a terminálban le kell ütnünk a Ctrl-C-t. Hogy miért? Mert még nem írtunk hozzá olyan kódot, ami eltüntetné és leállítaná az alkalmazást. Ez lesz a következő dolog, amit megcsinálunk. Írjuk be az alábbi sort a self.window.show() elé:

```
self.window.connect("delete_event", self.delete_event)
```

Ezt követően a gtk.main() hívás után rakjuk be a következő rutint:

```
def delete_event(self, widget, event, data=None):
    gtk.main_quit()
    return False
```

Újra mentsük el a programunkat „simple2.py” néven, és ismét futtassuk terminálban.

Most, ha a címsorban az X-re kattintunk, az alkalmazás ki fog lépni. De mi is történik itt valójában? Az első sor, amit elhelyeztünk (self.window.connect...) hozzáköti a delete eseményt egy callback rutinhoz, ebben az esetben self.delete_event-hez. Azzal, hogy „False”-t adunk vissza a rendszernek, a tényleges ablak rendszermemóriából való törlése is megtörténik.

Nos, nem tudom, ti hogy vagytok vele, de én szeretem, ha az alkalmazások a képernyő közepén nyílnak meg, nem pedig egy véletlen helyen, vagy a sarokban – ahol akár valami el is takarhatja. Módosítsuk a kódot úgy, hogy középre helyezze az ablakot. Csak annyit kell tennünk, hogy beírjuk az __init__ metódusban a self.window.connect elé a következő sort:

```
self.window.set_position(gtk.WIN_POS_CENTER)
```

Ahogy sejthetjük is, ez az ablak pozícióját a képernyő közepére állítja. Mentsük el „simple3.py” néven, és futtassuk.

Így már sokkal szebb, de nem sokra megyünk vele. Próbáljunk meg egy widgetet hozzáadni. Ha még emlékeztek a réges-régi Boa Constructor-beli munkánkra, akkor felidézhetitek, hogy a widget egyszerűen csak egy előre elkészített ablakba helyezhető vezérlőelem, amivel ott valamit csinálhatunk. A legegyszerűbb ezek közül a gomb.

Írjuk be az alábbi kódrészletet az előző kódunkba, közvetenül az __init__ rutinban található self.window.connect sor után:

```
self.button = gtk.Button("Close Me")
self.button.connect("clicked",self.btn1Clicked,None)
self.window.add(self.button)
self.button.show()
```

Az első sor létrehozza a gombot és a felületén lévő szöveget. A következő sor hozzákapcsolja azt a kattintás eseményhez. A harmadik elhelyezi a gombot az ablakban, az utolsó pedig megjeleníti. A self.button.connectet megvizsgálva megfigyelhetjük, hogy annak három paramétere van. Az első az esemény, amelyhez kapcsolódni akarunk, a második az a rutin, amelyik az esemény hatására elindul – ebben az esetben ez a „self.btn1Clicked” – és a harmadik egy, az imént meghatározott rutinnak átadott, argumentum (ha van).

Következőnek létre kell hoznunk a self.btn1Clicked rutint. Helyezzük el az alábbiakat a self.delete_event rutin után:

```
def btn1Clicked(self,widget,data=None):
    print "Button 1 clicked"
    gtk.main_quit()
```

Ahogy láthatjuk, a rutin nem sok mindent csinál. Egyszerűen kiíratja a terminálra, hogy „Button 1 clicked”, majd meghívja a gtk.main_quit()-et. Ezzel bezárja az ablakot, és leállítja a program működését – mintha csak az X-re kattintottunk volna a címsorban. Mentsük el „simple4.py” néven, és futtassuk terminálban. Láthatjuk, ahogy az ablakunk középen megjelenik a „Close me” gombbal. Kattintsunk rá, és az alkalmazás a terv szerint bezáródik. Figyeljük meg, hogy az ablak sokkal kisebb lett, mint a simple3.py esetében. Ugyan át tudjuk méretezni, de a gomb vele együtt átalakul. Hogy miért van ez? Nos, mivel csak egyszerűen rádobtuk a gombot az ablakra, ezért az ablak átméreteződik, hogy pontosan elférjen benne a vezérlőelem.

Kissé megszegtük a GUI-programozás szabályait azzal, hogy közvetlenül – anélkül, hogy bármilyen containert használtunk volna – az űrlapra helyeztük a gombot. Emlékeztek még arra, hogy az első Boa Constructoros GUI-s cikkekben egy sizer dobozt (containert) használtunk a vezérlőelemek tárolására? Most is ezt kellene tennünk (még abban az esetben is, ha csak egy elemünk van). A következő példában egy HBoxot (horizontal box, vízszintes doboz) helyezünk el gombunk tárolására, és egy újabb gombot is elhelyezünk abban. Ha függőleges containerre lenne szükségünk, VBoxot használnánk.

Kezdetnek vegyük a „simple4.py”-t alapul. Töröljünk mindent a self.wndow.connect(...) és a self.window.show() sorok között! Ide fogjuk elhelyezni az új sorokat. A HBox és az első gomb kódja:

```
self.box1 = gtk.HBox(False,0)
self.window.add(self.box1)
self.button = gtk.Button("Button 1")
self.button.connect("clicked",self.btn1Clicked,None)
self.box1.pack_start(self.button,True,True,0)
self.button.show()
```

Lépésekre bontva ez a kód a következőt csinálja: először elhelyezünk egy self.box1 nevű HBoxot. A HBoxnak átadott paraméterek: homogeneous (True vagy False) és spacing:

```
HBox = gtk.HBox(homogeneous=False, spacing=0)
```

A homogeneous nevű paraméter azt szabályozza, hogy a dobozban lévő widgetek ugyanakkorák legyenek-e (a HBox esetében ez a szélességre vonatkozik, a VBox-nál pedig a magasságra). A mi esetünkben ennek false értéket adunk, illetve a spacingnek 0-t. Ezután hozzáadjuk a dobozt az ablakhoz. Létrehozzuk a gombot ugyanúgy, ahogyan az előbb, és hozzákapcsoljuk annak

kattintás eseményét az általunk írt rutinhoz.

Ezen a ponton egy új paranccsal találkozunk. A self.box1.pack_start utasítással adjuk a containerhez (HBox) a gombot. Ezt a parancsot a self.window.add helyett használjuk a containerben való elhelyezéshez. A parancs (mint fentebb is):

```
box.pack_start(widget,expand=True, fill=True, padding=0)
```

A pack_start utasítás paraméterei a következők: az első maga a widget, ezután jön az expand (True vagy False), majd a fill (True vagy False), és végül a padding érték. A containerek esetén a spacing a widgetek között kimaradó helyet jelenti, a padding pedig a widget jobb/bal szélére vonatkozik. Az expand argumentummal kiválaszthatjuk, hogy a widgetek feltöltsék-e a maradék helyet (True), vagy a doboz összemegy pont akkorára, hogy elférjenek benne (False). A fillnek csak akkor van hatása, ha az expand True. Legvégül látható-vá tesszük a gombot. A következő kódrészlet a második gombé:

```
self.button2 = gtk.Button("Button 2")
self.button2.connect("clicked",self.btn2Clicked,None)
self.box1.pack_start(self.button2,True,True,0)
self.button2.show()
self.box1.show()
```

Vegyük észre, hogy ez a kód lényegében ugyanaz, mint az első widget esetében. Az új kódrészlet utolsó sora jeleníti meg a dobozt.

Ezután meg kell írnunk a self.btn2Clicked rutint. A self.btn1Clicked rutin után helyezzük el az alábbiakat:

```
def btn2Clicked(self,widget,data=None):
    print "Button 2 clicked"
```

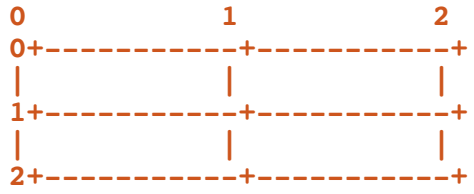
és kommenteljük ki a btn1Clicked rutinban a következőt:

```
gtk.main_quit()
```

Azt szeretnénk elérni, hogy mindkét gomb kiírja a „Button X Clicked” szöveget az ablak bezárása nélkül.

Mentsük el „simple4a.py” néven. Futtassuk terminálban. Azt tapasztaljuk, hogy a középre helyezett ablakban két gomb van (pont az ablak széléig érnek): az egyik „Button 1” címkével, a másik „Button 2”-vel rendelkezik. Kattintsunk rájuk, és figyeljük meg, hogy megfelelően válaszolnak a kattintás eseményre. Mielőtt bezárnánk az ablakot, méretezzük azt át (vonszoljuk az ablak jobb alsó sarkát), és vegyük észre, hogy a gombok az átméretezéssel együtt növekednek és mennek össze. Ahhoz, hogy megértsük az expand paraméter működését, a self.box1.pack_start expand értékét állítsuk át True-ról False-ra mindkét sorban. Futtassuk újra a programot, és figyeljük meg, hogy mi történik. Ez alkalommal kezdetben az ablak ugyanúgy néz ki, mint eddig, de amikor átméretezzük, a gombok megtartják szélességüket, és az ablakméret növekedésével üres terület marad a jobb oldalon. Következőnek változtassuk vissza az expandrametert True-ra és állítsuk a fillt False-ra. Indítsuk el a programot: ugyan a gombok mérete nem változott, de átméretezéskor üres hely található a gombok két oldalán. Ne feledd, a fill nem csinál semmit, ha az expand False-ra van állítva.

A widgetek csoportosításának egy másik módja a table használata. Sokszor, ha mindent könnyen el tudunk helyezni egy rácsszerű alakzatban, a table a legmegfelelőbb (és legegyszerűbb) választás. Egy table-re gondolhatunk úgy, mint egy táblázat soraira és oszlopaira, melyekben widgetek vannak. Mindegyik widget egy vagy több cellát foglal el (ahogy a program megkívánja). Talán a következő ábra segít ennek elképzelésében. Itt van egy 2×2-es rács:



Az első sorba két gombot fogunk elhelyezni. Egyet az 1-es oszlopba, egyet pedig a 2-esbe. A második sorba egy mindkét

oszlopot elfoglaló gombot rakunk. Így:

```
0           1           2
0+----------+----------+
| Button 1  | Button 2  |
1+----------+----------+
|       Button 3        |
2+----------+----------+
```

A táblázat elkészítéséhez létre kell hoznunk a table objektumot, és hozzá kell adnunk az ablakhoz. A table létrehozásához írjuk be ezt:

```
Table = gtk.Table(rows=1,columns=1,homogeneous=True)
```

Ha a homogeneous paraméter True, akkor a table dobozai átméreteződnek a táblázatban lévő legnagyobb widget méretére. Ha viszont False, akkor az egyes dobozok méretét az adott sor legmagasabb widgetje, illetve az oszlop legszélesebb widgetje adja. Ezután létrehozzuk a widgetet (ahogyan a gombokat előzőleg), majd hozzákapcsoljuk azt a táblázat megfelelő sorához és oszlopához. Ez a következőképpen történik:

```
table.attach(widget,left point,right point,top point,bottom
point,xoptions=EXPAND|FILL,yoptions=EXPAND|FILL,
xpadding=0,ypadding=0)
```

Itt csak az első öt paraméter kötelező. Tehát ahhoz, hogy egy gombot a nullás számú sor nullás oszlopába helyezzünk el, a következő parancsot kell használnunk:

```
table.attach(buttonx,0,1,0,1)
```

Ha viszont a nullás sor egyes számú oszlopába raknánk (ne feledd, hogy a számozás itt 0-tól kezdődik), mint fentebb a 2-es gombot, a hívás így nézne ki:

```
table.attach(buttonx,1,2,0,1)
```

Remélem, ez eddig világos, mint a vakablak! Mindenesetre kezdjük el a kódot, és menet közben talán jobban megértjük. Először a szokásos rész jön:

```
# table1.py
import pygtk
pygtk.require('2.0')
import gtk
class Table:
   def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.set_position(gtk.WIN_POS_CENTER)
        self.window.set_title("Table Test 1")
        self.window.set_border_width(20)
        self.window.set_size_request(250, 100)
        self.window.connect("delete_event", self.delete_event)
```

Van egy-két új dolog, amit meg kell tárgyalnunk, mielőtt továbblépnénk. A 9. sor az ablak címét „Table Test 1”-re állítja. A „set_border_width” hívás arra való, hogy az egész ablak köré 20 pixeles szegélyt adjunk, mielőtt widgeteket helyeznénk el azon. Végül 250×100 pixeles méretet kényszerítünk az ablakra a „set_size_request” függvénnyel. Eddig tiszta? Most hozzuk létre a táblázatot, és helyezzük el az ablakban:

```
table = gtk.Table(2, 2, True)
# 2x2-es rács létrehozása
self.window.add(table)
```

Most pedig elkészítjük az első gombunkat, beállítjuk az eseménykezelőt, majd elhelyezzük egy cellában, és láthatóvá tesszük:

```
button1 = gtk.Button("Button 1")
button1.connect("clicked",self.callback,"button 1")
table.attach(button1,0,1,0,1)
button1.show()
És most a kettes számú gomb:
button2 = gtk.Button("Button 2")
button2.connect("clicked",self.callback,"button 2")
table.attach(button2,1,2,0,1)
button2.show()
```

Lényegében ugyanaz, mint az első gomb esetében, a table.attach hívásban lévő különbséget leszámítva. Továbbá figyeljük meg, hogy az eseménykezelő rutint „self.callback”-nek hívják, és mindegyik gombra megegyezik. Ez idáig rendben is van. Mindjárt megértitek, hogy mit is csinálunk.

Vegyük a 3. gombot! Ez lesz a „Quit” gombunk:

```
button3 = gtk.Button("Quit")
button3.connect("clic-
ked",self.ExitApp,"button 3")
table.attach(button3,0,2,1,2)
button3.show()
```

Végül megjelenítjük a táblázatot, illetve az ablakot. Továbbá itt van a már használt main és delete rutin:

```
  table.show()
  self.window.show()
def main(self):
   gtk.main()
def delete_event(self,
widget, event, data=None):

   gtk.main_quit()
   return False
```

Most jöhet az érdekesebb rész! Az 1-es és 2-es gombra a „self.callback” eseménykezelőt állítottuk be, aminek itt van a kódja:

```
def callback(self,widget,da-
ta=None):
    print "%s was pressed" %
data
```

Amikor a felhasználó a gombra kattint, a kattintás esemény bekövetkezik, és az eseménykezelő beállításakor megadott adatok elküldésre kerülnek. Az 1-es gomb esetében a küldött adat a „button 1”, a 2-esnél pedig „button 2”. Mindössze annyit csinálunk, hogy kiírjuk a terminálra a „button x was pressed” szöveget. Biztosra veszem, hogy már megfogalmazódott bennetek az, hogy egy olyan eszközre bukkantunk, mely nagyon hasznos lehet egy jól felépített IF | ELIF | ELSE vezérlési szerkezettel egybekapcsolva.

Befejezésül definiálnunk kell az „ExitApp” eljárást a „Quit” gombhoz:

```
def ExitApp(self, widget,
event, data=None):
    print "Quit button was
pressed"
    gtk.main_quit()
```

És a végső main kódja:

```
if __name__ == "__main__":
  table.main()
```

Egyesítsük az egész kódot egy „table1.py” nevű alkalmazásba, és futtassuk terminálban.

Emlékeztetőül, ha pyGTK-val szeretnénk GUI programot készíteni, ezek a lépései:

- Hozzuk létre az ablakot.
- A widgetek tárolására készítsünk HBox, VBox vagy Table objektumokat.
- Helyezzük el a widgeteket pack vagy attach hívásokkal (attól függően, hogy dobozról vagy táblázatról van szó).
- Tegyük láthatóvá a widgeteket (show hívás).
- Tegyük láthatóvá a dobozt vagy a táblázatot.
- Tegyük láthatóvá az ablakot.

Most már a továbbhaladáshoz szükséges eszközök és tudás nagy részére szert tettünk. A kódok fenn vannak a Pastebinen: http://fullcirclemagazine.pastebin.com/wnzRsXn9. Remélem, találkozunk a következő hónapban is!

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

Hogyanok

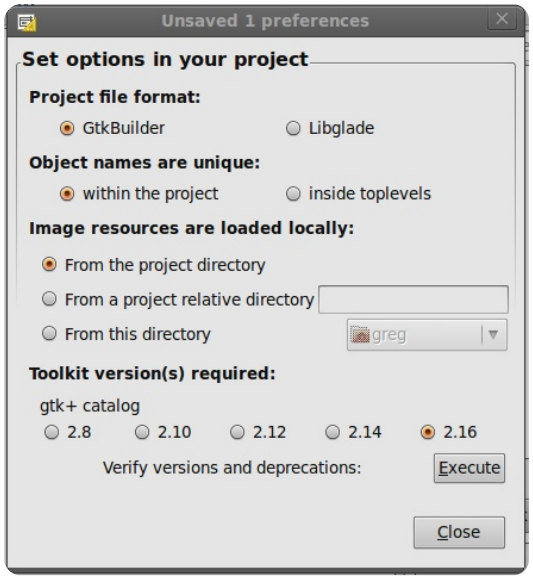
Írta: Greg Walters

Programozzunk Pythonban – 21. rész

Ha már egy ideje követed a cikkeimet, akkor emlékezhetsz az 5. és 6. részben a Boa Constructoros GUI-s alkalmazásra. Nos, ez alkalommal a Glade Designerrel fogunk foglalkozni, ami egy teljesen más program, de hasonlóan épül fel. Az Ubuntu Szoftverközpontból tudod telepíteni: keress rá a glade-re és telepítsd fel a GTK+ 2 User Interface Buildert.

Csak hogy tudj róla, ennek az alkalmazásnak a ismertetéséhez több részt is fel fogunk használni. A végső célunk egy MP3 és hasonló médiafájlokat kezelő playlist készítő program létrehozása. A tutorial ezen része a felület tervezésére fog koncentrálni. Következő alkalommal pedig az egészet összetartó kódról fogunk beszélni.

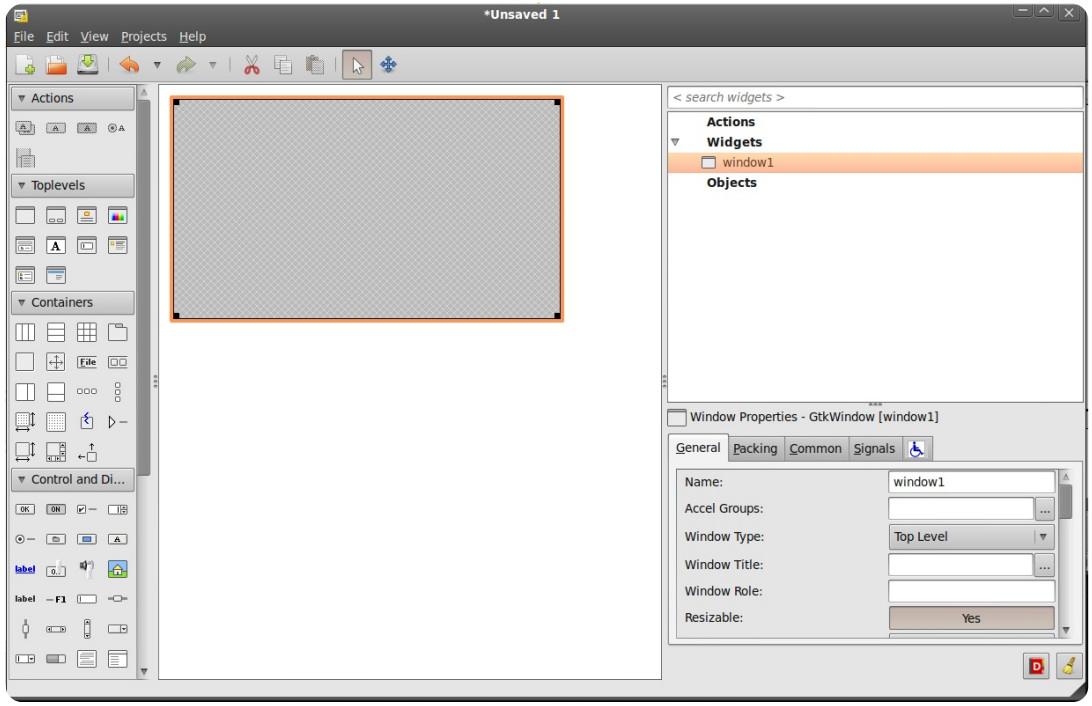
Kezdjük is el a tervezést. Amikor először elindítod a Glade designert, egy beállítások ablakkal fogsz találkozni (fenn). Itt válasszuk ki a Libglade-et, és az „inside toplevels”-t, majd



kattintsunk a close-ra. Ezzel eljutottunk a designer ablakba.

Nézzük meg a fő ablakot (jobbra). A bal oldalon vannak az eszközeink, középen a tervező terület és a jobb oldalon a tulajdonságok és a hierarchia ablak.

Az eszközöknél keressük meg a „Toplevels” nevezetű részt, és kattintsunk az első elemére (ha az egeret felette hagyjuk, akkor „Window”-t ír ki). Ezzel létre tudjuk hozni az üres ablak felületét, melyen



majd dolgozni fogunk.

Figyeljük meg, hogy a hierarchia Widgets részén található egy window1. Menjünk le a tulajdonságok részhez és állítsuk át a name-et window1-ről MainWindowra, illetve a Window Title-t „Playlist Maker v1.0”-ra. Mentsük el munkánkat „PlaylistMaker.glade” néven. Mielőtt továbblépnénk, a tulajdonságok General fülén keressük meg a Window Positiont, és állítsuk Centerre. Kattintsunk a jelölőnégyzetre a Default Width-nél és írjunk be 650-et. Ugyanezt ismételjük meg a Default Heighttel is, csak most 350-nel. Következőnek kattintsunk a Common fülre és görgessünk le a „Visible” bejegyzésig. BIZONYOSODJUNK MEG AFELŐL, HOGY EZ „YES”-RE VAN ÁLLÍTVA – különben az ablakunk nem jelenne meg. Végül válasszuk ki a Signals fület és a GtkObject-nél kattintsunk a jobbra mutató

nyílra. A destroynál a handler oszlop lenyíló listáján válasszuk ki az „on_MainWindow_destroy”-t. Ezzel létrehoztunk egy olyan eseményt, ami az „X”-re való kattintással bezárja az ablakot. Figyelmeztetlek: miután beállítottuk a destroy eseményt, kattintsunk valahova kívülre, hogy elmentődjenek a beállítások. Erre egy bug miatt van szükség. Mentsünk ismét.

Mint előzőleg, most is vboxokba és hboxokba kell raknunk widgetjeinket. Ez talán a GUI programozás legkönnyebben elfelejthető szabálya. A fő ablakban egy függőleges boxot fogunk elhelyezni. Ehhez válasszuk ki a Container-beli eszközök közül a Vertical Box-ot (második ikon balról az első sorban), majd kattintsunk a tervezőbe lévő ablakunkba. Az így felugró ablakban megadhatjuk, hogy az hány elemű legyen. Az alapértelmezett a három, de nekünk most ötre lesz szükségünk. Az elrendezés fentről-lefelé a következő: eszköztár, egy treelist vezérlő, két vízszintes terület a címeknek, gomboknak és text boxoknak, illetve egy állapotsor.

Most már elhelyezhetjük a widgetjeinket. Először csináljunk egy toolbart. Ez a negyedik ikon lesz a containerek második sorában. Kattintsunk a vbox legfelső sorára. Ez a rész szinte teljesen el fog tűnni. Ne aggódjunk, néhány perc múlva visszahozzuk.

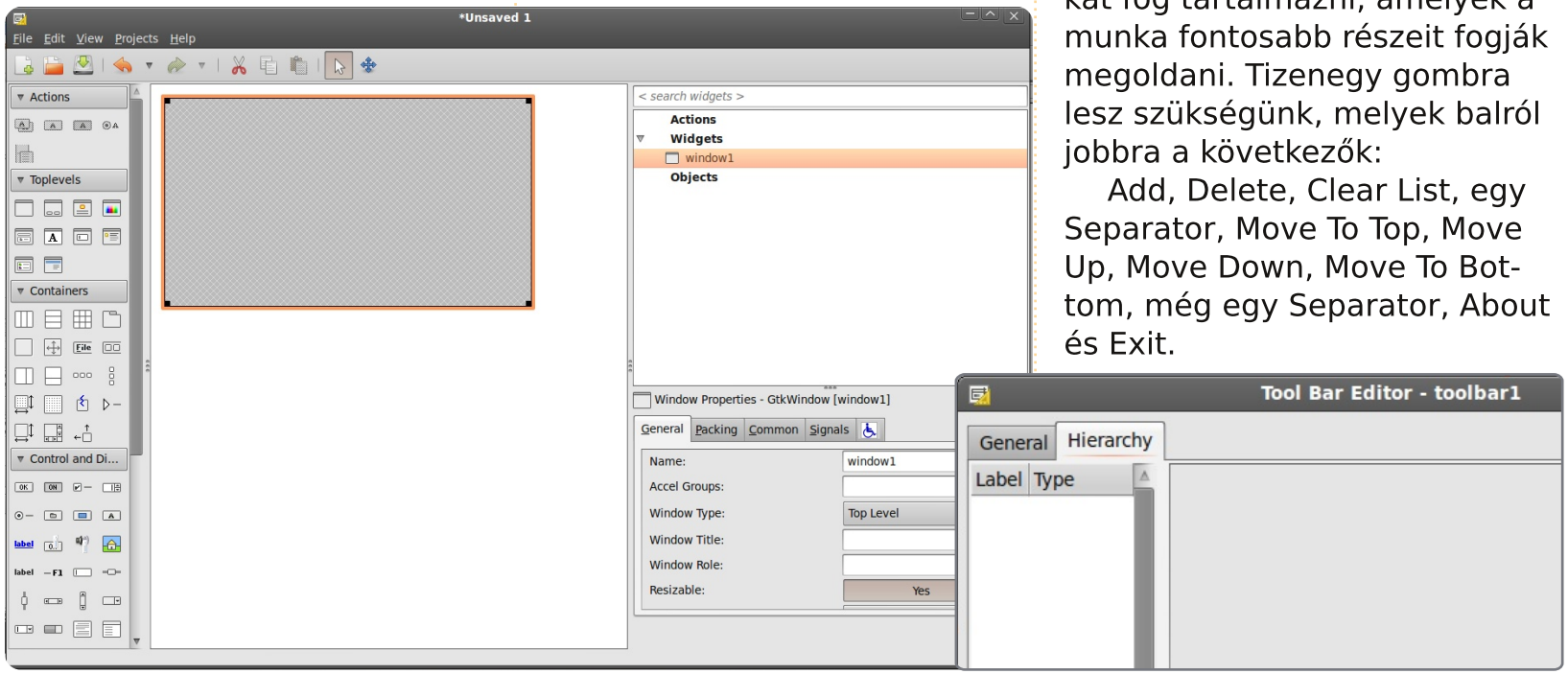
A következő helyre a treelist tárolásához egy Scrolled Window-t kell elhelyeznünk. Segítségével görgetni tudunk a treelistben. Tehát, keressük meg a Scrolled Window ikont a Containers részben (balról a második ikon az ötödik sorban), és kattintsunk a vbox második helyére. Ezután a következő két rublikába egy-egy Horizontal boxot fogunk elhelyezni. Mindkettőnek három-három része legyen. Végül adjuk az alsó sorhoz a Status Bart. Ezt a Control and Display rész végén találjuk. Ezen a ponton a tervező nézet a lent láthatóhoz hasonlóan kell, hogy kinézzen.

Végül, de nem utolsó sorban, helyezzük el a Control and Displayből a Tree View widgetet a scrolled windowba. Egy felugró ablakban meg kell adnunk, hogy melyik TreeView modellt akarjuk használni. Egyelőre kattintsunk az „OK” gombra. Ezt később fogjuk beállítani.

Most koncentráljunk egy kicsit a Scroll Windowra, válasszuk is ki a hierarchia ablakban. Görgessünk le egészen a General fül „Horizontal Scrollbar Policy” részéhez és állítsuk át „Always”-re, majd csináljuk meg ugyanezt a Vertical Scrollbar Policy-re is. Mentsünk.

Eddig minden rendben, most nézzük meg az eszköztárunkat. Ez a terület az alkalmazásunk felső részén, közvetlenül a címsor alatt lesz, és olyan gombokat fog tartalmazni, amelyek a munka fontosabb részeit fogják megoldani. Tizenegy gombra lesz szükségünk, melyek balról jobbra a következők:

Add, Delete, Clear List, egy Separator, Move To Top, Move Up, Move Down, Move To Bottom, még egy Separator, About és Exit.



A hierarchiában kattintsunk a „toolbar1”-re, ezzel kijelölve azt. A Glade Designer felső részén van egy ceruza-szerű dolog. Kattintsunk rá. Így tudjuk előhozni az eszköztár-szerkesztőt. Kattintsunk a Hierarchy fülre. Valami ilyesmivel találjuk szembe magunkat (előző oldal jobb alsó sarka).

Most az összes eszköztári gomb elhelyezése következik. Ennek lépései:

- Kattintsunk az Add gombra.
- Változtassuk meg a gomb nevét.
- Változtassuk meg a gomb címkéjét.
- Válasszuk ki a képet.

Ezt mind a tizenegy widgetre meg fogjuk ismételni. Tehát, kattintsunk a name-re és írjuk be a „tbtnAdd” szöveget. Görgessünk le az Edit Labelig és írjuk be, hogy „Add”, majd még egy kicsit lentebb, az Edit Image alatti Stock ID szövegdoboz lenyíló listájában válasszuk ki az „Add”-ot (hogy később a kódban hivatkozhassunk rá). A „tbtn” a „Toolbar Button” rövidítése. Ezzel az elnevezési konvencióval könnyű lesz a kódban megtalálni, nem beszélve arról, hogy öndokumentáló is lesz.

Most már csak az eszköztár többi widgetjét kell elhelyeznünk. Hozzunk létre még egy gombot a Delete-nek. Ez (mint ahogy már sejthetitek) „tbtnDelete” nevű lesz. Ismét állítsuk be a címkét és az ikont. Következőnek készítsük el a „tbtnClearAll” nevű gombot és használjuk hozzá a Clear ikont. Ezen a ponton egy Separatorra van szükségünk. Kattintsunk az Addra és a name-hez írjuk be a „Sep1” szöveget, majd a lenyíló listában válasszuk ki a Separatort.

Helyezzük el a maradék widgetet „tbtnMoveToTop”, „tbtnMoveUp”, „tbtnMoveDown”, „tbtnMoveToBottom”, „Sep2”, „tbtnAbout” és „tbtnQuit” néven. Biztos vagyok benne, hogy meg tudjátok találni a hozzájuk tartozó ikonokat. Amint elkészültünk, lépjünk ki a hierarchia ablakból és mentsük el munkánkat. A jobbra látható képhez hasonló dolgot kellene kapnunk.

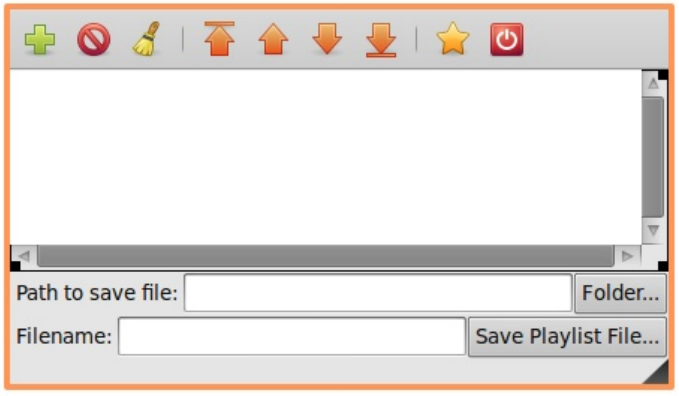
Most a feladatunk az, hogy beállítsuk a létrehozott gombok eseménykezelőit. A hierarchiában válasszuk ki a tbtnAdd widgetet. Így ki kellet jelölnünk a hierarchia megfelelő elemét és magát a gombot is. A kattintás eseményt a tulajdonságok rész Signals fülén találjuk a GtkToolButtont alatt. A kattintás kezelőjénél – mint korábban is – válasszuk ki az „on_tmbtnAdd_clicked”-et, majd ahhoz, hogy elmentődjenek a változtatások, kattintsunk fölé vagy alá. Tegyük meg ugyanezt az összes többi gombra. Se a mellékattintásról, se a mentésről ne feledkezzünk el! Az elválasztóknál nincs szükség eseményekre, egyszerűen hagyjuk ki őket.

Ezután fel kell töltenünk a hboxainkat. A felső hbox a következőket tartalmazza: egy címkét, egy text widgetet és egy gombot. Az eszközök közül válasszuk ki a Labelt (ne a képet), és helyezzük el a bal oldali rubrikában. Ezután rakjunk egy Text Entry widgetet középre, majd egy gombot a jobb szélre. A másik hboxra is csináljuk meg ugyanezt.

Eljött az idő, hogy beállítsuk a widgetek tulajdonságait. A hierarchiában válasszuk ki a label1-et a hbox1-en belül. A tulajdonságok General fülén görgessünk le az „Edit label appearance” részig, és állítsuk a címkét „Path to save file:”-ra. Következőnek a Package fül Expandját állítsuk „No”-ra. A packing ismerős lehet a múlt alkalomról. A padding legyen négy, így egy kis helyet hagyunk ki a címke két oldalán. Most a button1-et választva állítsuk az Expandot „No”-ra. A General fülön állítsuk a nevét „btnGetFolder”-re. Figyeljük meg, hogy mivel ez nem eszköztár gomb,

ezért nem raktunk ki elé a „t”-t. Görgessünk le a Label bejegyzésig és gépeljük be, hogy „Folder...”, majd kattintsunk a Signals fülre és állítsuk be az eseményt „GtkButton/clicked” ről „on_btnGetFolder_clicked”-re. Mielőtt a következő hbox widgetjeinek tulajdonságaival babrálnánk, még egy dolgot meg kell tennünk. A hierarchián belül válasszuk ki a hbox1-et és a Packing alatti expandot állítsuk ismét „No”-ra. Ezzel a hboxnak kevesebb helyre van szüksége. Végül állítsuk be a Text Entry widget nevét „txrPath”-ra.

Ismételjük meg ugyanezt a hbox2-re is: állítsuk az Expandot „No”-ra, a címkét „Filename:”-re, és a paddinget 4-re. A gomb neve legyen „btnSavePlaylist”, a szövege „Save Playlist File...” és az Expandja „No”. Állítsuk be a kattintás eseményét, majd a Text Entry widget nevét „txtFilename”-re. Mentsünk!




```
<widget class="GtkWindow" id="MainWindow">
    <property name="visible">True</property>
    <property name="title" translatable="yes">Playlist Maker v1.0</property>
    <property name="window_position">center</property>
    <property name="default_width">650</property>
    <property name="default_height">350</property>
    <signal name="destroy" handler="on_MainWindow_destroy"/>
```

Végül az ablakunknak a bal sarokban látható képhez hasonlónak kellene lennie.

Ez mind szép és jó, de mit is csináltunk? Ezt a programot még futtatni sem tudjuk, mivel nincs hozzá forráskód. Amit létrehoztunk, az valójában egy „playlistmake.glade” nevű XML fájl. Nehogy összezavarjon a kiterjesztés. Ez tényleg egy XML fájl. Ha nagyon alaposak vagyunk, akkor megnyithatjuk kedvenc szövegszerkesztőnkkel (esetemben ez a gedit), és megnézhetjük annak tartalmát.

Láthatjuk, hogy egy egyszerű szöveg le tudja írni az ablakot, annak minden widgetjével együtt. Nézzük meg például a main widget kódját (fenn).

Láthatjuk, hogy a widget neve „MainWindow”, és a címkéje „Playlist maker v1.0”, illetve megtalálhatjuk itt többek között annak eseménykezelőjét is.

Nézzük meg az eszköztár gombjainak kódját (lent).

Remélhetőleg kezd érthető lenni a dolog. Most már csak annyi dolgunk van, hogy egy olyan kódot írunk, ami megjeleníti hosszú munkánk gyümölcsét működés közben. Hozzuk elő a kódszerkesztőt és kezdjük ezzel...

```
<child>
   <widget class="GtkToolButton" id="tbtnAdd">
      <property name="visible">True</property>
      <property name="label" translatable="yes">Add</property>
      <property name="use_underline">True</property>
      <property name="stock_id">gtk-add</property>
      <signal name="clicked" handler="on_tbtnAdd_clicked"/>
   </widget>
   <packing>
       <property name="expand">False</property>
       <property name="homogeneous">True</property>
   </packing>
</child>
```

Nos, körülbelül ugyanúgy létrehoztuk az importjainkat, mint múlt hónapban. Vegyük észre, hogy a mutagen.mp3-ból a „sys”-t és az „MP3”-at is importáljuk. A mutagent még a 9-es részben telepítettük, szóval ha nincs már meg nálad, akkor ebben a részben tudsz utánajárni. A mutagen importjára a következő alkalommal lesz szükségünk, és a sys import is csak azért kell, hogy a program az utolsó kivételnél rendesen ki tudjon lépni.

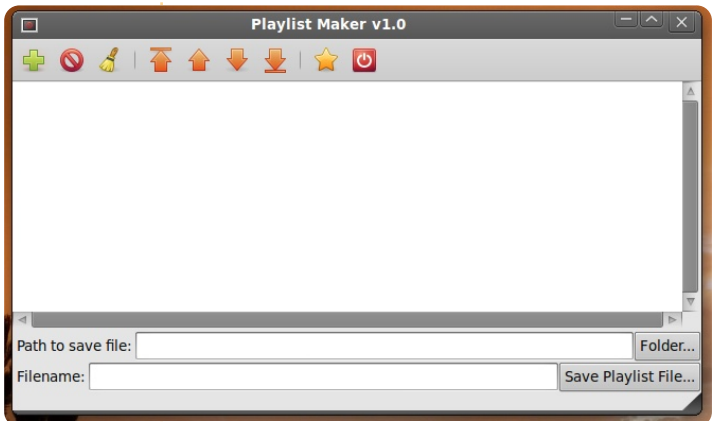
```
#!/usr/bin/env python
import sys
from mutagen.mp3 import MP3
try:
    import pygtk
    pygtk.require("2.0")
except:
    pass
try:
    import gtk
    import gtk.glade
except:
    sys.exit(1)
```

Ezután hozzuk létre az ablakot definiáló osztályunkat. Ezt láthatjuk a jobbra fenn lévő ábrán.

```
class PlayListMaker:
    def __init__(self):
        #================================================
        #                    Window Creation
        #================================================
        self.gladefile = "playlistmaker.glade"
        self.wTree =
gtk.glade.XML(self.gladefile,"MainWindow")
```

Többnyire megegyezik a múltkorival. Nézzük az utolsó két sort. A glade fájlt (self.gladfile) úgy definiáljuk, hogy a Glade designerben létrehozott fájl nevét tartalmazza. Figyeljük továbbá meg, hogy nem vesszük bele az elérési utat, csak a fájl nevét. Ha a glade fájlunk nem a kód mellett lenne, akkor egy path-ra is szükségünk lesz. Mindanozonáltal, mindig jó dolog egy helyen tárolni őket. Következőnek az ablakunkat definiáljuk self.wTree néven. Ehhez fogunk minden olyan alkalommal fordulni, amikor az ablakra van szükségünk. Továbbá azt is megadjuk, hogy a fájl egy XML, és a megjelenítendő ablak a „MainWindow”. Több ablakunk is lehet egy glade fájlon belül. Majd máskor erre is kitérünk.

Nézzük most az eseményeket. Előző hónapban az eseménykezelő rutinokhoz a button.connect vagy window.connect hívásokat használtuk. Most egy kicsit máshogy csináljuk: egy dictionary-t fogunk használni. Ez egy olyan tömbszerűség, aminél indexek helyett egy kulccsal érjük el az adattagként tárolt elemeket. Ezek a Key és a Data. Itt van a kód, ami egy kicsit tisztába teszi a dolgot. Egyelőre csak két eseménnyel dolgozunk. (lent)



A két esemény – az „on_MainWindow_destroy” és az „on_tbtnQuit_ clicked” – melyek a dictionary kulcsai. A dictionary adata mindkettőnél a „gtk.main_quit”. Bármikor, amikor a GUI egy eseményt generál, a rendszer megkeresi vele a dictionary kulcsát, így innentől tudni fogja, hogy mely rutint kell meghívnia. Következőnek hozzá kell kapcsolnunk az ablak üzenet-kezelőjéhez. Ezt a következő sorral tesszük meg:

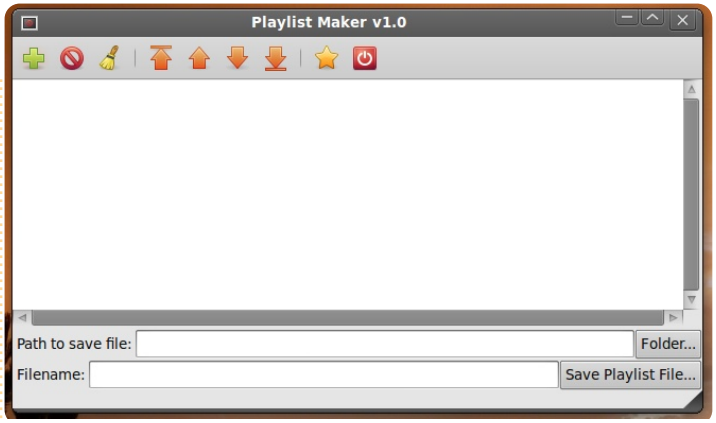
```
self.wTree.signal_autoconnect(dict)
```

Már majdnem készen vagyunk, csak a main rutinunkat kell elhelyezni:

```
if __name__ == "__main__":
    plm = PlayListMaker()
    gtk.main()
```

```
#================================================
#                  Create Event Handlers
#================================================
dict = {"on_MainWindow_destroy": gtk.main_quit,
        "on_tbtnQuit_clicked": gtk.main_quit}
```

Mentsük el ezt a fájlt „playlistmaker.py" néven! Futtassuk (jobbra)!



Induláskor még a bezáródáson kívül nem sokat csinál. A maradék a következő alkalomra marad. Azért hogy felkeltsem az érdeklődéseteket: TreeViewt, dialógusokat és hasonlókat fogunk használni. Találkozzunk a következő számban is!

Glade fájl:
http://fullcirclemagazine.pastebin.com/YM6U0Ee3

Python forráskód:
http://fullcirclemagazine.pastebin.com/wbfDmmBh

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.

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 Linuxos **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 vélemé-nyeiteket, desktopjaitok kinézetét és történeteiteket. Szükségünk van a Fókuszban rovathoz játékok, programok és hardverek áttekintő leírására, a Hogyanok rovatban szereplő cikkekre (K/X/Ubuntu témával), ezenkívül, ha bármilyen kérdés, javaslat merül fel bennetek, nyugodtan küldjétek a következő címre: articles@fullcirclemagazine.org

A Full Circle Csapata

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

Full Circle Magazin
Magyar Fordítócsapat

Koordinátor:
Pércsy Kornél
Fordítók:
Palotás Anna
Tömösközi Máté Ferenc
Szerkesztő, korrektor:
Heim Tibor

Köszönet a Canonical-nek és a fordítócsapatoknak világszerte, továbbá **Thorsten Wilms**-nek a jelenlegi Full Circle logóért.