Full Circle

AZ UBUNTU LINUX KÖZÖSSÉG FÜGGETLEN MAGAZINJA

Programozói sorozat – Különkiadás

Programozói sorozat Különkiadás

Programozzunk Pythonban

6. Kötet

A Full Circle magazin nem azonosítandó a Canonical Ltd.-vel!

A Full Circle magazin különkiadása

Full Circle

AZ UBUNTU LINUX KÖZÖSSÉG FÜGGETLEN MAGAZINJA

Üdvözöllek egy újabb, „egyetlen témáról szóló különkiadásban”

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

Most ez a **„Programozzunk Pythonban” 32–38. részének** az újabb kiadása (a magazin 60–67. 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
32. rész 3. oldal

Programozzunk Pythonban
33. rész 5. oldal

Programozzunk Pythonban
34. rész 8. oldal

Programozzunk Pythonban
35. rész 11. oldal

Programozzunk Pythonban
36. rész 15. oldal

Programozzunk Pythonban
37. rész 18. oldal

Programozzunk Pythonban
38. rész 22. oldal

SOME RIGHTS RESERVED

Minden szöveg- és képanyag, amelyet a magazin tartalmaz, a Creative Commons Nevezd meg! - Így add tovább! 3.0 Unported Licenc 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 Fircle 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 licensz alatt leszel köteles terjeszteni.

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

Programozzunk Pythonban – 32. rész

Írta: Greg D. Walters

Nagyon szeretem az Android tabletem. Nap mint nap használom ugyan, de az asztali számítógépet azért még mindig nem tudja teljes mértékben helyettesíteni. Egy tabletet általában én is csak arra használok, amire mindenki más: böngészésre, zenehallgatásra, videónézésre, játékokra stb. Az eszköz fontosságát igyekszem persze olyan alkalmazásokkal igazolni, amelyek a bevásárló- és tennivalólistákat kezelik, néha a tablet segítségével találom meg az olcsó benzinkutakat, vagy éppen vicces dolgokat az unokánknak, de azért mégis… Valójában ez csak egy játék számomra. Miért van szükség egy díszes érintőképernyős tabletre ahhoz, hogy bevásárlólistát készítsünk? Ismerjük be, azért használunk ilyen eszközöket, hogy lássuk az irigységet a többi vásárló arcán, miközben mi a listát görgetjük, és egy érintéssel kihúzzuk a már megvett tételeket bevásárlás közben. Geeknek lenni JÓ! Listakészítésre használhatnánk akár egy régi boríték hátlapját is, de az nem lenne elég menő, ugye?

A házas geek férfiak 99%-ának a felesége nem geek, és én is az ő táborukat erősítem. Elvettem egy csodálatos, szerető nőt, akit viszont nem érdekelnek különösebben az újfajta kütyük. Így amikor én egy ilyen láttán elkezdem csorgatni a nyálam, ő csak sóhajt egyet, és azt mondja: „Jó, ha TÉNYLEG úgy gondolod, hogy szükségünk van rá…” Persze amikor a cipőboltban az 50. pár cipőjét nézi ki magának, akkor ő is ugyanezzel a csodálattal bámulja a kiválasztott terméket.

Az igazat megvallva nem volt nehéz beszereznem az első tabletünket. A feleségemnek vettem, miközben kemoterápiás kezelésekre járt. Egy ideig próbálkozott egy laptoppal, de a súlya és a melegedés miatt a dolog nem igazán vált be. Laptopon nem akart e-bookokat olvasni, így ha olvasni akart, elkezdett zsonglőrködni egy könyvvel, egy laptoppal és egy mp3 lejátszóval. Mindeközben pedig mindenféle csövek álltak ki belőle, amelyeken keresztül mindenféle vegyületek áramlottak belé. A tablet nagyon jó választás volt, hiszen ettől a pillanattól kezdve egy eszközön tudott e-bookokat olvasni, zenét hallgatni, filmet nézni, a webet böngészni, az emailjeit megnézni, frissíteni a rákról írott blogját, követni a barátai facebook-eseményeit és persze játszani. Ha elfáradt, egyszerűen csak oldalra tette, sokkal egyszerűbb volt ez így, mintha egy méretes laptop, egy könyv, egy mp3-as lejátszó, egy távirányító és még ki tudja minek a társaságában kellett volna feküdnie.

Miközben az orvosok öntötték a szervezetébe a vegyszereket, én leültem a szoba sarkába egy konnektor közelében, és a hatéves laptopomon próbáltam dolgozni. A feladatok közötti időt Android programozással kapcsolatos keresésekkel töltöttem ki. Megtudtam, hogy ez a legtöbb esetben Javában történik. Már majdnem rávettem magam, hogy újra megtanulok Javában programozni, amikor találtam néhány olyan eszközt, amelyeknek köszönhetően az Android operációs rendszer Pythonban is programozható. Az egyik ilyen eszköz az „SL4A” (Scripting Layer for Android), erről fogok írni a következő néhány számban. A mai alkalommal az SL4A Androidon történő beállításáról lesz szó.

Kérdezheted, hogy vajon miért akarok én Android programozásról írni egy olyan magazinban, ami a Linuxról szól. A legegyszerűbb válasz az, hogy az Android alapja is Linux. Minden, ami Android, gyakorlatilag Linuxra épül!

Sok weboldal ír arról, hogyan indítsuk el az SL4A-t egy asztali Android Emulatoron. Erről inkább majd máskor írnék, most magával az Android eszközzel foglalkozunk. Az SL4A telepítéséhez szükséges fájlok itt találhatók: http://code.google.com/p/android-scripting/. Ne jöjjünk zavarba, töltsük le a megfelelő APK csomagot.

Az alkalmazásbeállításoknál engedélyezzük az „Unkown Sources” lehetőséget. A letöltés gyorsan megy, ezután telepítsük, majd keressük meg az ikonját, és érintsük meg. Egy meglehetősen kiábrándító fekete képernyőt kapunk „Scripts… No matches found” felirattal. Ez rendben van. Nyomjuk meg a Menu gombot és válasszuk ki a View-t. Ezután válasszuk ki az Interpreterst, majd újra a Menut és az Add-et. A következő menüben válasszuk a Python 2.6.2-t. Az eszköz ezután rákérdez, hogy le akarjuk-e tölteni a Pythont Androidra. Ha ez megvan, válasszuk az Opent. Egy újabb menüt kapunk Install, Import Modules, Browse Modules és Uninstall Modules lehetőségekkel. Válasszuk az Installt. A Python ezután letölti és telepíti a további szükséges modulokat. Ráadásként néhány példaszkriptet is kapunk. Végül érintsük meg a Back gombot, és az Interpreter képernyőn megjelenik a feltelepített Python 2.6.2. Érintsük meg újra a Back gombot , és láthatunk néhány példa Python-kódot is.

Ennyit mára. Egyelőre csak az étvágyat akartam meghozni mindenkinek. Fedezzétek fel a Python Androidos változatát! Hasznos lehet továbbá a http://developer.android.com/sdk/index.html link, innen az Android

> **Egy meglehetősen kiábrándító fekete képernyőt kapunk […] Ez rendben van.**

SDK (Software Development Kit) tölthető le az asztali számítógépünkre. Van benne egy Android Emulator, így egyből ki is lehet próbálni dolgokat. Az SDK beállítása Linux alatt tényleg nagyon egyszerű, így nem hiszem, hogy gond lesz vele.

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

Hogyan írjunk ékezeteket a billentyűzetről

Írta: Barry Smith

Ha a Linux rendszered francia, német vagy spanyol nyelvű és emiatt ékezeteket igényel, vagy ha időnként olyan ékezeteket kell használnod, amelyek angol szavakban nem szerepelnek, sokan nem tudják, hogy van egy nagyon egyszerű mód a billentyűzeten való megoldásra. A következő csak az angol nyelvű billentyűzetre vonatkozik.

Éles ékezet
Üsd le az Alt Gr + ;-t (pontosvesszőt). Emeld fel a kezed, majd üsd le a kívánt é magánhangzót

Kúpos ékezet
Üsd le az Alt Gr + '-t (idézőjelet). Emeld fel a kezed, majd üsd le a kívánt î magánhangzót.

Visszafelé dőlő ékezet
Üsd le az Alt Gr + #-t (kettőskeresztet). Emeld fel a kezed, majd üsd le a kívánt è magánhangzót.

Umlautos ékezet
Üsd le az Alt Gr + [-t. Emeld fel a kezed, majd üsd le az u magánhangzót: ü.

ñ – Üsd le az Alt Gr +]-t. Emeld fel a kezed, majd üsd le az n másalhangzót: ñ.

œ – Üsd le a Shift + Alt Gr-t. Emeld fel a kezed, majd üsd le az o-t, majd az e-t: œ. Az œ nem jelenik meg, amíg le nem ütjük az e-t is.

A ¿ és ¡ (fordított felkiáltójel) előhívásához, amelyet spanyolul mindig a kérdések és felszólítások előtt használok, nyomd le az Alt Gr + Shiftet, tartsd lenyomva mindkét billentyűt, majd nyomd le a _-t (aláhúzást) a ¿-hez vagy a !-t (felkiáltójelet) a ¡-hez.
Ha ezeket nagybetűvel akarod látni, csak nyomd le a Shift-et a betű leütése előtt.

Programozzunk Pythonban – 33. rész

Ezúttal az Android SDK-t fogjuk beállítani a saját asztali Linuxunkon. Létre fogunk hozni egy virtuális Android eszközt, feltelepítünk rá SL4A-t és pythont, valamint csinálunk egy gyors tesztet is.

Figyelem, ez nem egy olyan dolog, amit 1 GB-nál kevesebb memóriával ellátott gépeken lehet futtatni. Az emulátor memóriaigénye meglehetősen nagy, 512 MB ram egy Ubuntuval felszerelt laptopon kevés volt: működött ugyan, de rendkívül lassan.

Itt egy gyors lista arról, hogy mit fogunk csinálni. Lépésről lépésre haladunk:
• Feltelepítjük a Java JDK6-ot.
• Telepítjük az Android SDK indító csomagot.
• Létrehozzuk és beállítjuk az AVD-ket.
• Ellenőrizzük az AVD-t, telepítjük az SL4A-t és a Pythont.

Tulajdonképpen jó lenne az Eclipse és az Android ADT Eclipse plugint is feltelepíteni, de mivel ebben a sorozatban Eclipse-el nem foglalkoztunk, ezért erről itt nem lesz szó. Amennyiben mégis szükséged lenne rá, hasznos segítséget nyújthat a http://developer.android.com/sdk/installing.html honlap.

1. lépés - Java JDK 6

Azok alapján amit olvastam és próbáltam, a Sun kiadására van szükségünk. Az OpenJDK elvileg nem működik. A weben több információt is találsz erről, itt most csak azokat a lépéseket részletezném, amelyekre nekem szükségem volt. Írjuk be az alábbiakat egy terminálba...

```
sudo add-apt-repository ppa:ferramroberto/java

sudo apt-get update

sudo apt-get install sun-java6-jdk
```

Ha ez megvan, módosítsuk a .bashrc fájlunkat: állítsuk be megfelelően a „JAVA_HOME” környezeti változót, hogy a dolgok rendben működjenek. Nyissuk meg mondjuk gedittel és a fájl aljára írjuk be:

```
export JAVA_HOME="/usr/lib/jvm/java-6-sun-1.6.0.06"
```

Mentsük el a fájlt és lépjünk tovább.

2. Lépés - Android SDK Starter Pack

Most kezdődik az igazi „móka”. Érdemes megnézni a developer.android.com/sdk/index.html oldalt, itt található az SDK. Töltsük le a legújabb verziót Linuxra, a cikk írásának pillanatában ez az android-sdk_r18-linux.tgz. Csomagoljuk ki az archívumkezelő segítségével valahová, én a saját home mappámba tettem. Minden közvetlenül ebből a mappából indítható, így nincs szükség külön telepítésre. Az én esetemben tehát az elérési út: /home/greg/android-sdk-linux. Irány tehát ez a mappa, majd ezen belül a tools. Lesz itt egy „android” nevű fájl, ami az aktuális SDK-t futtatja. Az egyszerűség kedvéért létrehoztam egy indítóikont is az asztalomon.

Most jön az unalmas rész. Futtassuk az android fájlt, ezzel elindítjuk az Android SDK Managert. Ellenőrizni fogja az elérhető platformokat és frissíti azokat. Figyelmeztetlek rá, hogy ez eltarthat egy ideig, így csak akkor indítsd el, ha van rá elég időd. Azt javaslom, hogy csak egyetlen platformot indíts el. Kezdetnek jó lesz mondjuk az Android 2.1, hiszen ha egy régi platformra fejlesztünk, az az újabb változatokon is problémamentesen működni fog (általában). Az eszközök (Tools) beállítására is szükség van, egyszerűen csak jelöljük be a két elemhez tartozó négyzetet és kattintsunk a telepítésre. A platform kiválasztásával és az eszközök beállításával már majdnem elkészítettük az első virtuális gépünket.

3. Lépés - Az első AVD létrehozása és beállítása

Visszatérve az Android SDK Managerhez, válasszuk ki a Tools-t a főmenüből, majd a Manage AVDs-t. Egy új ablak nyílik meg. Mivel ez az első alkalom, nem lesznek előre beállított virtuális eszközeink. Kat-

tintsunk a „New" gombra. Ezzel egy újabb ablakot nyitunk, ahol a virtuális Android eszközünk tulajdonságait állíthatjuk be. Egy egyszerű Android emulátor eszköz beállításához az alábbi lépésekre van szükséged:

• Az eszköz nevének megadása. Ez akkor válik fontossá, ha egynél több eszközt kezdünk el használni.
• A cél platform szintjének beállítása
• Az SD kártya méretének beállítása (lásd alább).
• A Skin felbontás beállítása.
• Az eszköz létrehozása.

A név szövegdobozba tehát írjuk be: „Test1". A cél combo-boxban válasszuk az alábbit: Android 2.1 - API Level 7. Az „SD Card:" szövegdobozba írjunk 512-t és a legördülő menüben szerepeljen „MiB". A „Skin" alatt a felbontás legyen 800x600. (Persze játszadozhatunk a többi beépített mérettel is.) Végül kattintsunk a „Create AVD" gombra. Hamarosan megjelenik egy üzenetablak amely tájékoztat arról, hogy az AVD létrejött.

4. Lépés - Az AVD tesztelése, az SL4A és a Python telepítése

Most végre jöhet egy kis szórakozás. Válasszuk ki az előbb létrehozott AVD-t és kattintsunk a Start gombra. A felugró párbeszédablakban kattintsunk a „Launch" gombra. Néhány perc várakozás után a virtuális eszköz létrejön a memóriában, valamint betöltődik és elindul az Android platform. (A folyamat felgyorsításáról később még lesz szó.)

Amint az AVD elindul és megjelenik a „home" képernyő, telepíthetjük az SL4A-t. A böngészőt, vagy a home képernyő google web kereső dobozát használva keressük meg az „sl4a"-t. A letöltések oldalon meg fogjuk találni a megfelelő linket: http://code.google.com/p/android-scripting/downloads/list.

Lapozzunk az oldalon lefelé, amíg el nem érjük az sl4a_r5 linket. Nyissuk meg és érintsük meg az „sl4a_r5.apk"-t. Igen, érintsük meg és ne kattintsunk rá. Érdemes mostantól így gondolkodnunk, ezáltal könnyebb lesz átállnunk erre a programozási módra. A letöltés most mindenesetre elindult. A felül megjelenő értesítő sávot talán érdemes lesz odébb tenni, hogy a letöltött fájlhoz férjünk. Érintsük meg, majd utána az „install" gombot is.

A fájl letöltése után két lehetőségünk van: megnyithatjuk, vagy a „Done" érintésével kiléphetünk a telepítőből. Most érintsük meg az „Open"-t.

Az SL4A el fog indulni. Valószínüleg egy párbeszédablak meg fogja kérdezni, hogy beleegyezel-e a használati adatok nyomonkövetésébe. Rajtad áll, hogy engedélyezed vagy elutasítod a műveletet. Mielőtt továbblépnénk, érdemes megismerni néhány billentyűkombinációt, amelyek megkönnyítik majd a munkát. Mivel nincs „valódi" Android eszközünk, az olyan gombok mint a Back, Home vagy Menu nem állnak a rendelkezésünkre. Ezekre azonban szükségünk lesz a menüpontok közötti mozgások során. Itt van tehát néhány hasznos rövidítés:

Back - Escape
Home - Home
Menu - F2

Most a pythont fogjuk letölteni és telepíteni az SL4A-ra. Érintsük meg a Menü-t (nyomjuk meg az F2-t) és válasszuk a „View"-t, majd az „Interpreters"-t. Úgy tűnik, mintha semmi sem történt volna, de menjünk a Menün (F2) belül az „Add"-ba, majd görgessünk le és válasszuk a „Python 2.6.2"-t. Ez le fogja tölteni az alap Python csomagot Androidra. Telepítsük, majd nyissuk meg. Négy lehetőség közül választhatunk: Install, Import Modules, Browse Modules és Uninstall Module. Válasszuk az Install-t. El fog indulni a letöltés és az Androidra készült legújabb Python változat részei fel fognak települni. Eltarthat egy darabig.

Ha minden kész érintsük meg a Back-et (Escape billentyű) és lépjünk vissza az SL4A Interpreters képernyőre. Mindenünk megvan már ahhoz, hogy végre Androidon Pythonozhassunk. Érintsük meg a Python 2.6.2-t és egy „standard"

Python shellt kapunk, ami teljesen olyan, mint amit az asztali gépeken megszokhattunk. Írjuk be az alábbi három sort a shellbe, egyszerre csak egyet. Minden alkalommal várjuk meg a „>>>" promptot.

```
import android

droid = android.Android()

droid.makeToast("Hello from Python on Android")
```

Ha az utolsó sor után is Entert nyomunk, egy lekerekített dobozt láthatunk a shellen belül alul középen, amely azt mondja „Hello from Python on Android". Ezt csinálja a „droid.makeToast" parancs.

Megírtad az első Python scripted Androidra. Király, ugye?

Most készítsünk egy indítóikont az Android home képernyőjére. Érintsük meg a Home gombot (Home billentyű). Ha a 2.1-es platformot választod, megjelenik egy csúszka a képernyő jobb szélén. Más platformot választva lehet ez egy négyzet vagy egy apró négyzetekből álló téglalap is. Mindenesetre ezzel az Apps képernyőre juthatunk. Érintsük meg és keressük meg az SL4A ikont. Most egy „hosszú érintés" (hosszú kattintás) következik, amellyel a parancsikont létrehozzuk a Home képernyőn. Tegyük oda, ahová csak szeretnénk.

Ezután elkészítjük az első elmentett szkriptünket. Menjünk vissza az SL4A-hoz. Lennie kell ott egy minta szkriptnek, ami a Python 4 Androidhoz alapértelmezetten jár. Érintsük meg a Menu gombot és válasszuk az „Add"-ot, majd a listából a „Python 2.6.2"-t. Meg fog jelenni egy szkript szerkesztő. Felül van a fájlnév doboz, ahol „.py" végződés már alapból meg is jelenik. Alatta van a szerkesztő ablak, amelybe a programunk első két sora már be is van írva (dőlt betűkkel ebben az írásban is mellékeltem, ellenőrzés céljából. A mi első mintánkban is szükség lesz erre a két sorra.)

```
import android

droid = android.Android()
```

Írjuk be az alábbi két sort a python kódunkba:

```
uname = droid.dialogGetInput("What's your name?")

droid.makeToast("Hello %s from Python on Android") % uname.result
```

Az első sor egy párbeszédablakot fog létrehozni (droid.dialogGetInput()), amely a felhasználó nevére kérdez rá. A választ a programunk az uname.result-ban tárolja el. A droid.makeToast()-ot már korábban is használtuk.

Nevezzük el a fájlt andtest1.py-nak és érintsük meg a Done-t, majd pedig a „Save & Run"-t. Ha minden működik, egy párbeszédablaknak kell felugrania, amely a nevedet kérdezi meg. Miután beírtad, egy figyelmeztetés fog megjelenni alul az alábbi szöveggel: „Hello neved from Python on Android".

Ennyit mára. Kimeríthetetlenül sok dokumentáció érhető el ingyenesen az SL4A-ról. A következő alkalomig nyugodtan lehet ismerkedni a rendszerrel, kezdésnek ehhez javaslom a http://code.google.com/p/android-scripting/wiki/Tutorials címet.

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.

Android programozás és a Python - 3. rész

Ez alkalommal az SL4A használatáról lesz szó. Készítünk egy nagyobb programot, majd ezt az ADB-n keresztül átküldjük a virtuális gépünkre.

Kezdjük a kódolással. Egyszerűen csak ki fogunk próbálni néhány SL4A használata során rendelkezésre álló „widgetet”. Nyissuk meg a kedvenc szövegszerkesztőnket az asztali gépünkön.

Mentsük el „atest.py” néven a jobbra fent látható kódot, de ne próbáljuk meg elindítani.

```
import android

droid = android.Android()
uname = droid.dialogGetInput("Hello","What's your name?")
droid.makeToast("Hello %s from python on Android!" % uname.result)
```

```
droid.dialogCreateAlert(uname.result,"Would you like to play a game?")
droid.dialogSetPositiveButtonText('Yes')
droid.dialogSetNegativeButtonText('No')
droid.dialogShow()
while True: #wait for events for up to 10 seconds...
      response = droid.eventWait(10000).result
      if response == None:
      break
      if response["name"] == "dialog":
      break
droid.dialogDismiss()
```

Az első sor az android könyvtárat importálja. A második sorban ennek egy példányát hozzuk létre. A harmadik sor egy „Hello” névvel ellátott párbeszédablakot hoz létre és jelenít meg, a parancssorban „What’s your name?” szöveggel, valamint lesz egy szövegdobozunk, ahová a felhasználó beírhatja a nevét és még két gomb, egy „OK” és egy „Cancel”. Ha a felhasználó megnyomja az „OK”-t, a válasz az uname változóban tárolódik el. Az utolsó sorunk végül azt mondja „Hello {felhasználónév} from python on Android!” Ez persze nem új, korábban is csináltunk már ilyet. Bővítsük ki a kódunkat (fentebb).

Mentsük el atest1.py néven. Miután megbeszéltük, hogy mit csinál, el fogjuk küldeni a virtuális gépünkre.

Vessünk egy pillantást a most beírt első négy sorra. Egy értesítő típusú párbeszédablakot hozunk létre, amely azt kérdezi, hogy „Would you like to play a game?”. Az ilyen típusú párbeszédablakokban nincs olyan szövegdoboz, amibe beleírhatunk. A következő két sor két gombot hoz létre „Yes” és „No” felirattal, előbbi a „pozitív” gomb, míg utóbbi a „negatív”. Ezek a megnevezések a válasz jellegére utalnak. A következő sor megjeleníti a párbeszédablakot. Az ezt követő hét sor a felhasználó válaszát várja.

Létrehozunk egy egyszerű ciklust (while True:), majd 10 másodpercen keresztül várjuk a választ a droid.eventWait(érték) hívással. A válasz („pozitív” vagy „negatív”) a response változóban lesz eltárolva. Ha a response tartalmazza a „dialog” nevet, akkor kilépünk a ciklusból és visszaadjuk a response értékét. Ha az időtúllépés előtt nem történne semmi, akkor is kilépünk a ciklusból. A response változó aktu-

ális visszaadott értéke valahogy így néz majd ki (feltételezzük, hogy a „pozitív" vagyis a „Yes" gomb volt megnyomva)...

```
{u'data': {u'which':
u'positive'}, u'name':
u'dialog', u'time':
1339021661398000.0}
```

Látható, hogy az érték a 'data' szótárba került, a párbeszéd kulcsa a szótár 'name'-je és van még egy 'time' értékünk is, amivel nem foglalkozunk.

Végül eltüntetjük a párbeszédablakot.

A kód virtuális gépre küldése előtt el kell indítanunk a virtuális gépet. Indítsuk el az Android emulátort. Miután elindult, vegyük észre hogy a címsor elején négy számjegy szerepel. Ez a portra utal, amelyet az eszköz éppen figyel. Az én esetemben (és valószínüleg nálad is) ez az 5554-es.

Most küldjük el a kódot a virtuális gépre. Nyissunk egy terminál ablakot és váltsunk arra a mappára, ahová a fájlt elmentettük. Feltételezve, hogy az SDK elérési útját megfelelően beállítottuk, beírhatjuk az alábbi utasítást:

```
adb devices
```

Ezzel arra kérjük az adb-t, hogy mutassa meg az összes csatlakoztatott eszközt. Ez nem csak az Android emulátorra vonatkozhat, hanem az okostelefonokra, tabletekre és egyéb Android eszközökre is. Valami ilyet kellene látnod...

```
List of devices attached
emulator-5554    device
```

Tudva, hogy az eszközünk valóban csatlakoztatva van szeretnénk átküldeni rá az alkalmazást. Ennek szintaxisa:

```
adb push source_filename
destination_path_and_filename
```

Az én esetemben tehát:

```
adb push atest1.py
/sdcard/sl4a/scripts/atest1.py
```

Ha minden jól megy, az alábbihoz hasonló, meglehetősen csalódást keltő üzenetet kapunk:

```
11 KB/s (570 bytes in 0.046s)
```

Most az Android emulátoron indítsuk el az SL4A-t. Látnunk kellene az összes python szkriptet, közöttük az atest1.py-t is. Érintsük meg (kattintsunk rá) és egy felugró párbeszédablakot kapunk hat ikonnal. Balról jobbra ezek a „Run in a dialog window" (Futtatás párbeszédablakban), „Run outside of a window" (Ablakon kívüli futtatás), „Edit" (Szerkesztés), „Save" (Mentés), „Delete" (Törlés) és „Open in an external editor" (Megnyitás külső szerkesztőben). Érintsük meg a bal szélső ikont („Run in a dialog window"), így látni is fogjuk, hogy mi történik.

Az első párbeszéd a nevünket fogja kérdezni. Írjunk be valamit a dobozba és nyomjuk meg az „Ok" gombot. Egy üdvözlő üzenet fog megjelenni. Ezután egy értesítőablakot kapunk. Nyomjuk meg rajta az egyik gombot, hogy eltűnjön. A visszajelzések egyelőre nem érdekelnek minket, szóval mindegy melyik gombot választjuk. Bővítsük tovább a kódunkat (jobbra fent). Ezzel egyszerűen csak megnézzük a választ, és ha ennek értéke időtúllépés miatt „None", akkor kiíratjuk azt, hogy „Timed out". Ha a válasz olyan, mint amilyet várunk, akkor az adatot hozzárendeljük az rdialog változóhoz. Bővítsük a kódot még egy kicsit...

```
if response==None:
    print "Timed out."
else:
    rdialog=response["data"]
```

Ez a rész a gomblenyomás-esemény által átadott adatot nézi át. Ha a válasz tartalmazza a „which" kulcsot, akkor ez egy legitim gombnyomás volt számunkra. Ez után azt ellenőrizzük, hogy az eredmény „pozitív"-e („Ok" gomb). Ha igen, egy újabb értesítő párbeszédet készítünk, de ez alkalommal felajánl-

```
    if rdialog.has_key("which"):
    result=rdialog["which"]
    if result=="positive":
        droid.dialogCreateAlert("Play a Game","Select a game to play")
        droid.dialogSetItems(['Checkers','Chess','Hangman','Thermal
Nuclear War']) # 0,1,2,3
        droid.dialogShow()
        resp = droid.dialogGetResponse()
```

juk a felhasználónak, hogy egy listából válassza ki a neki megfelelő elemet. Esetünkben a lista a Checkers, Chess, Hangman és Thermal Nuclear War elemeket tartalmazza és ezekhez 0-tól 3-ig számokat rendelünk. (Ismerős? Igen, akárcsak a filmben.) Ezután a párbeszédablakot jelenítjük meg és várjuk a választ. Amire igazán kíváncsiak vagyunk, az egy szótár formájában jelenik meg. Feltételezve, hogy a felhasználó a Chess-t választotta, a válasz az alábbi formában érkezik:

```
Result(id=12,
result={u'item':1},
error=None)
```

```
if resp.result.has_key("item"):
    sel = resp.result['item']
    if sel == 0:
        droid.makeToast("Enjoy your checkers game")
    elif sel == 1:
        droid.makeToast("I like Chess")
    elif sel == 2:
        droid.makeToast("Want to 'hang around' for a while?")
    else:
        droid.makeToast("The only way to win is not to play...")
```

Ebben az esetben valóban érdekeltek vagyunk a visszaadott adatok eredményének részeiben. A kiválasztás az 1-es és az 'item' kulcsban van tartva. Jöjjön most a kód következő része... (fentebb)

Itt azt ellenőrizzük, hogy a válasz tartalmazza-e az „item" kulcsot és ha igen, azt a „sel" változóhoz rendeljük. Most egy if/elif/else ciklusban ellenőrizzük az értékeket és az lekezeljük az összes kiválasztottat. A droid.makeToast függvényt használjuk a válasz megjelenítésére. Természetesen hozzáadhatod ide a saját kódrészletedet is. Végül jöjjenek mai kódunk utolsó sorai... (jobbra középen)

Itt egyszerűen csak reagálunk a többi típusú gombnyomásra is.

Mentsük el, tegyük fel az eszközre és futtassuk a programunkat.

```
            elif result=="negative":
                droid.makeToast("Sorry.  See you later.")
        elif rdialog.has_key("canceled"):
            print "Sorry you can't make up your mind."
        else:
            print "unknown response=",response
print "Done"
```

Az SL4A tehát lehetőséget biztosít arra, hogy „GUI-zált" alkalmazásokat készítsünk, de nem teljes GUI appokat. Ez azért persze ne tartson vissza attól, hogy megírd a saját kódjaidat Androidra. Azzal azért ne számolj, hogy ezeket a programokat el tudod majd adni, a legtöbb ember ugyanis csak teljes GUI típusú appokat használ. Erről a következő alkalommal lesz szó. Az SL4A használatáról egyébként egyszerű kereséssel is rengeteg leírást és kiegészítő információt lehet találni a weben.

A kódot szokás szerint feltettem pastebinre:
http://pastebin.com/REkFYcSU

Találkozzunk legközelebb!

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

Programozzunk Pythonban – 35. rész

Írta: Greg D. Walters

Ez alkalommal teszünk egy rövid kitérőt az Android programozásból és egy GUI programozásra alkalmas új keretrendszert fedezünk fel: a ***Kivy***-t. A csomagot a http://kivy.org oldalról tölthetjük le. Arról, hogy Ubuntu alatt hogyan kell feltelepíteni a Kivyt találunk egy leírást az alábbi címen: http://kivy.org/docs/installation/installation-ubuntu.html.

A Kivy egy nyílt forráskódú könyvtár, amely a multi-touch kijelzők használatát segíti elő. Ha ez még önmagában nem elég jó, a program egyébként platformfüggetlen, azaz működik Linux, Windows, Mac OSX, IOS és (tádám) Android rendszeren is. Így már talán érthető, hogy miért erről szól ez a cikk. Jegyezzük hát meg, gyakorlatilag bármilyen kódot írunk Kivy-ben, az működni fog az összes, előbb felsorolt platformon is a kódok módosítása nélkül.

Mielőtt még túl messzire mennénk, hadd mondjak előbb néhány dolgot. A Kivy RENDKÍVÜL erőteljes. A Kivy új eszközöket ad a GUI programozás segítéséhez. Mindazonáltal a Kivy-t elég nehéz kezelni. A lehetőségeink néhány előre megadott widget (elem) használatára korlátozódnak. A Kivy-nek ráadásul nincs GUI tervezője, így jól el kell terveznünk a dolgokat, mielőtt bármi bonyolultba kezdenénk. Nem szabad elfelejteni, hogy a Kivy folyamatos fejlesztés alatt áll, bizonyos dolgok gyorsan megváltozhatnak. A teszt kódjaim eddig problémamentesen futottak az új Kivy változatokon is, de nincs arra garancia, hogy ez mindig így lesz.

Ahelyett, hogy rögtön belecsapnánk a lecsóba és megírnánk a kódunkat, most inkább bepillantunk néhány Kivy példakódba, a saját kód írását a következő hónapban kezdjük.

Miután kicsomagoljuk a Kivy-t a saját mappájába, használjuk a terminált és lépjünk be ebbe a mappába. Nálam ez a /home/greg/Kivy-1.3.0. Most váltsunk az examples mappára és ezen belül a widgets-re. Nézzük az accordion_1.py példát.

A kód nagyon egyszerű, de készít nekünk egy csinos kis widgetet (lásd alul).

Az első három sor import utasítás. Minden olyan widgetet, amit használni szeretnénk importálnunk kell, valamint szükség van az App-ra is a kivy.app-ból.

A következő nyolc sor a fő alkalmazás osztály. Előbb definiáljuk az osztályt, majd egy build nevű rutint készítünk. A Kivy programokban szinte minden esetben szükség van egy build rutinra. Ezután egy root objektumot hozunk létre az Accordion widget-ből, majd öt Accordionitem-et és beállítjuk ezek címét. A folytatásban hozzáadunk tíz címkét „Very big content” szöveggel, majd az egyes címkéket hozzáadjuk a root widgethez (ez az Accordion) és végül visszatérünk a root objek-

```
from kivy.uix.accordion import Accordion, AccordionItem
from kivy.uix.label import Label
from kivy.app import App

class AccordionApp(App):
     def build(self):
          root = Accordion()
          for x in xrange(5):
               item = AccordionItem(title='Title %d' % x)
               item.add_widget(Label(text='Very big content\n' * 10))
               root.add_widget(item)
          return root

if __name__ == '__main__':
     AccordionApp().run()
```

tummal. Ez tulajdonképpen megjeleníti a root objektumot a Kivy által létrehozott ablakban. A végén van még egy „if __name__” utasítás, az alkalmazás ezután indul el.

Lépjünk tovább és indítsuk el, nézzük meg mi történik.

Rövid időn belül megjelenik egy ablak, benne öt függőleges sávval. A megnyitáshoz kattintsunk a sávra, a tíz címke meg fog jelenni. Természetesen minden sávban ugyanaz a szöveg szerepel, de ezen már bizonyára könnyen képes lennél változtatni.

Az Accordion widget nagyon sok mindenre használható, az első dolog ami nekem az eszembe szokott jutni egy konfigurációs képernyő készítése… minden sáv egy külön konfigurációs gyűjtőnek tekinthető.

Most a textalign.py példát tekintjük át. Nem olyan „menő”, mint az előző, de ez a példa remekül szemléltet olyan dolgokat, amire később még szükség lesz.

Mielőtt belenéznénk a kódba, futtassuk le a programot.

Az ablak tetején egy címkét kell látnunk, kilenc darab felirattal ellátott piros dobozt 3x3-as rácsban és az ablak alján négy gombot. Ezeket megnyomva (megérintve) a piros dobozokban lévő szöveg elrendezése megváltozik. Ez a példa két dolog miatt érdekes: egyrészt bemutatja a fontos widgetek kezelését, másrészt pedig azok elrendezésének megváltoztatását.

Jöjjön most a kód. Részekre bontva fogjuk átnézni, kezdjük az import-tal…

```
from kivy.app import App
from kivy.uix.label import Label
from kivy.uix.gridlayout import GridLayout
from kivy.uix.floatlayout import FloatLayout
from kivy.properties import ObjectProperty
```

Alatta valami egyedi következik. Egy olyan osztályt hozunk létre, amelyben nincs egy sor programkód sem. Erre kicsit később még visszatérünk.

```
class BoundedLabel(Label):

  pass
```

Most egy „Selector" névre hallgató osztályt hozunk létre.

```
class Selector(FloatLayout):

app = ObjectProperty(None)
```

Majd pedig az Application osztályt.

Létrehozzuk a select rutint. Készítünk egy GridLayout widgetet (amit gridnek nevezünk), ez három sorból és három oszlopból áll. Ez a rács tartalmazza majd a kilenc piros dobozt.

```
for valign in ('bottom',
'middle', 'top'):

for halign in ('left',
'center', 'right'):
```

Két ciklusunk van, egy külső és egy belső.

```
label = BoundedLabel(text='V:
%s\nH: %s' % (valign, halign),

size_hint=(None, None),
halign=halign, valign=valign)
```

A fenti kódban a BoundedLabel widget egy példányát hozzuk létre minden egyes piros doboz esetén. Mondhatnánk, hogy „Várjunk csak, itt nincs is BoundedLabel widget! Egyszerűen csak egy értékadás történik a kódban.” Igen is, meg nem is. Egy widgetet is példányosítunk mellette, de erről később még lesz szó.

Az alábbi kódrészben a select rutinban beállított „case” változó értékét vizsgáljuk meg.

Most pedig eltávolítjuk a rácsot, hogy egy tiszta képernyőt kapjunk.

```
if self.grid:

self.root.remove_widget(self.
grid)
```

```
class TextAlignApp(App):

      def select(self, case):

      grid = GridLayout(rows=3, cols=3, spacing=10, size_hint=(None, None),
                        pos_hint={'center_x': .5, 'center_y': .5})
```

A kötés (bind) módja állítja be a méretet és a rács hozzáadódik a gyökér objektumhoz.

```
grid.bind(minimum_size=grid.s
etter('size'))

self.grid = grid

self.root.add_widget(grid)
```

Emlékezzünk, az előző példánál azt mondtam, hogy szinte mindig egy build rutint kell használnunk. Nézzünk erre egy példát. A root objektumot egy FloatLayout widget-tel hozzuk létre. Ezután a Selector osztály meghívásával egy Selector objektumot készítünk, majd ezt hozzáadjuk a root objektumhoz és inicializáljuk a kijelzőt a self.select(0) meghívásával.

Az alkalmazás így végül már futtatható.

```
TextAlignApp().run()
```

Még mielőtt tovább lépnénk, tisztázzunk egy-két dolgot. Ha belenézel a .py fájlt tartalmazó mappába, találsz ott egy textalign.kv-t is. Ez egy speciális fájl amelyet a Kivy arra használ, hogy a saját widget-eidet és szabályaidat létrehozd. Amikor a Kivy alkalmazásod elindul, ugyanebben a mappában keresni fogja a .kv segítő fájlt. Ha megtalálja, egyből be is tölti azt. Nézzük most a .kv fájlban lévő kódot.

Az első sor azt mondja meg, hogy melyik a legrégebbi Kivy verzió, amire szükségünk van az alkalmazás futtatásához.

```
#:kivy 1.0
```

Ezután a BoundedLabel widgetet hozzuk létre. Az alkalmazás összes piros doboza egy BoundedLabel.

A Color a doboz háttérszínét állítja be pirosra (rgb: 1,0,0). A Rectangle widget egy téglalapot hoz létre. Az aktuális alkalmazás kódjában a BoundLabel widget meghívásával a címkét mint szülő adjuk át. A címke mérete és pozíciója (itt a .kv fájlban) tetszőlegesen beállítható.

Létrehozzuk a Selector widgetet. Ez az ablak alján megjelenő négy gombot, valamint a tetején található címkét tartalmazza.

Vegyük észre, hogy az ablak tetején címként működő címke pozíciója (pos_hint) „top”, magassága 50 pixel a betűmérete pedig 16-os. Minden gomb szövege középre van igazítva. Az on_release egy kötésszerű utasítás, így ha a gombot elengedjük, (ebben az esetben) meghívódik a root.app.select függvény a „case” értékkel.

```
if case == 0:
                    label.text_size = (None, None)
          elif case == 1:
                    label.text_size = (label.width, None)
          elif case == 2:
          label.text_size = (None, label.height)
          else:
          label.text_size = label.size
                    grid.add_widget(label)
```

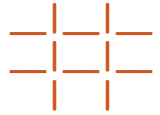
```
     def build(self):
     self.root = FloatLayout()
          self.selector = Selector(app=self)
     self.root.add_widget(self.selector)
     self.grid = None
          self.select(0)
     return self.root
```

```
<BoundedLabel>:
     canvas.before:
          Color:
               rgb: 1, 0, 0
     Rectangle:
          pos: self.pos
               size: self.size
```

Remélem, hogy most már látható az egésznek az értelme: a Kivynek bizony van ereje.

Beszéljünk most két másik widgetről, amelyek már feltűntek a kódban, de eddig elegánsan átugrottam őket: ez a GridLayout és a FloatLayout.

A GridLayout egy szülő widget, amely sor és oszlop leírásokat alkalmaz a widgetek cellákban történő elhelyezésére. A mi esetünkben egy 3x3-as rács áll a rendelkezésünkre (olyan, mint a Tic-Tac-Toe játékban).



Ha egy widgetet a GridLayout-ba szeretnénk helyezni, az add_widget eljárást alkalmazhatjuk. Itt van egy probléma: csak hozzáadási sorrendben lehet elhelyezni az elemeket a rácsban. Ráadásul az elemeket balról jobbra, fentről lefelé sorrendben rakhatjuk csak le. Nem hozhatunk létre üres cellát. Csalni persze azért lehet. Rád hagyom, hogy kitaláld ennek módját.

A FloatLayout elem egyszerűen csak egy olyan szülőnek tűnik, ami más gyermek elemeket tartalmaz.

Lassan a végére érek. Ez alkalommal tulajdonképpen csak az érdeklődéseteket szerettem volna felkelteni a Kivy iránt, remélem sikerült. A következő néhány cikkben részletesebben áttekintjük a Kivy által nyújtott lehetőségeket, szó lesz például APK készítéséről is, amellyel alkalmazásunkat kiadhatjuk Androidra.

Addig is nyugodtan nézelődjetek a Kivy példakódok között, de ajánlom a Kivy dokumentációját is: http://kivy.org/docs/.

```
<Selector>:
      Label:
      pos_hint: {'top': 1}
            size_hint_y: None
      height: 50
            font_size: 16
      text: 'Demonstration of text valign and halign'
      BoxLayout:
            size_hint_y: None
      height: 50
            ToggleButton:
                  halign: 'center'
            group: 'case'
                  text: 'label.text_size =\n(None, None)'
                  on_release: root.app.select(0)
                  state: 'down'
      ToggleButton:
                  halign: 'center'
            group: 'case'
                  text: 'label.text_size =\n(label.width, None)'
                  on_release: root.app.select(1)
      ToggleButton:
                  halign: 'center'
                  group: 'case'
            text: 'label.text_size =\n(None, label.height)'
                  on_release: root.app.select(2)
      ToggleButton:
                  halign: 'center'
                  group: 'case'
            text: 'label.text_size =\n(label.width, label.height)'
            on_release: root.app.select(3)
```

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

Hogyanok

Programozzunk Pythonban – 36. rész

Írta: Greg D. Walters

Mielőtt belekezdenék, megemlíteném, hogy a Programozzunk Pythonban sorozat az FCM hasábjain éppen három évvel ezelőtt indult. Köszönetet szeretnék mondani Ronnie-nak és a teljes Full Circle Magazine stábnak a támogatásért. Köszönettel tartozom továbbá Nektek, olvasóknak, nem gondoltam volna, hogy a Programozzunk Pythonban sorozat ilyen sokáig fog tartani.

Továbbá, olvastam olyan megjegyzéseket, hogy három év után a címben szereplő „Beginning” kifejezés már egyáltalán nem aktuális és érdemes lenne lecserélni. Három év után is kezdők lennénk? Nos, tulajdonképpen egyetértek, de ugyanakkor mostanában is kapok olyan leveleket, amelyben az olvasó jelzi, hogy rátalált az FCM Python sorozatára és elkezdte azt olvasni az elejétől. Ezek az emberek pedig kezdők. A 37. résztől mindenesetre a „Beginning” szó eltűnik a sorozat címéből.

És most térjünk a lényegre... Kivy következik.

Képzeld azt, hogy egy gitáron játszol. Nem léggitáron, hanem igazin. Tegyük fel, hogy nem te vagy a legjobb gitáros a világon és néhány akkord lefogása bizony problémákat okoz még. Például a „standard” C, E, G, F típusú akkordok már mennek, de néhány más akkord - mint a fisz moll vagy cisz moll akkord - megy ugyan, de a váltások csak lassan. Mit tennél, ha a fellépésig már csak néhány heted lenne hátra és gyorsan fel kellene venned a ritmust? A megoldás erre a kapodaszter, az a vicces kis izé, amit néha a gitár nyakán lehet látni. Ennek segítségével megváltoztatható mind a hat üres húr hangmagassága. Ezt nevezzük transzponálásnak. Transzponálni néha akár fejben is tudunk, máskor pedig az a könnyebb, ha leülünk és egy darab papírra fölírjuk az új hangokat. Például ha az eredeti akkordunk fisz mol volt és a kapodasztert a 2-es bundra helyezzük, egyszerűen játszhatunk e molt. Mindehhez persze idő kell. Készítsünk hát egy alkalmazást, amely végigfut a bund pozíciókon és megmondja, hogy milyen állásban lesz könnyű a kiválasztott akkordokat lejátszani.

Ez az app elég egyszerű lesz. Lesz egy felirat címkéje, egy gomb az „alap” skálánkkal szöveg formátumban, egy görgetőnézet (scrollview, egy csodálatos szülő widget amely további vezérlőket is tartalmaz és lehetővé teszi a vezérlés belsejének „eldobását” a görgetés vezérléséhez) sok gombbal, melyek elmozdítják a skálát, mint szöveget és egy kilépés gomb. Valahogy így fog kinézni -> lásd a lenti képet.

Kezdjük egy új python fájllal, a main.py-val. Ez akkor válik fontossá, amikor a Kivy-ből egy Android alkalmazást akarunk készíteni. Adjuk hozzá az import utasításainkat.

Figyeljük meg a második sort, „kivy.require('1.0.8')". Ezzel biztosítjuk azt, hogy a Kivy által nyújtott legújabb és legjobb eszközöket is használatba vehessük. A 3. sorban egy system exit-et is behozunk, lesz ugyanis egy kilépés gombunk.

A „Transpose" osztályunk eleje így néz ki:

```
class Transpose(App):

    def exit(instance):

        sys.exit()
```

Most a build rutinunkon dolgo-

```
------------------------------------------------------------------
                      Transposer Ver 0.1

     C  C#/Db  D  D#/Eb   E    F   F#/Gb   G  G#/Ab  A   A#/Bb  B    C

 1| C#/Db  D  D#/Eb  E     F  F#/Gb   G   G#/Ab  A  A#/Bb  B       C   C#/Db
 2|   D  D#/Eb  E     F   F#/Gb  G   G#/Ab  A   A#/Bb  B    C  C#/Db D
------------------------------------------------------------------
```

zunk. Erre minden Kivy app-ban szükség van.

```
import kivy
kivy.require('1.0.8')
from sys import exit
from kivy.app import App
from kivy.core.window import Window
from kivy.uix.button import Button
from kivy.uix.label import Label
from kivy.uix.anchorlayout import AnchorLayout
from kivy.uix.scrollview import ScrollView
from kivy.uix.gridlayout import GridLayout
```

Hát ez elég zavaros. Sajnos a szövegszerkesztő nem minden esetben őrzi meg a tagolást rendesen, még rögzített szélességű betűtípusoknál sem. Az ötlet az, hogy a text1 karakterlánc egy egyszerű skála, amely a „C" hanggal kezdődik. Mindegyiknek középre igazítottnak kellene lennie, 5 szóközzel. Jobbra lent látható.

A text2 ugyanez megismételve. Használni fogunk egy eltolást a text2 karakterláncban, hogy beleférjünk a gomb szövegmezőjébe a scrollview widget-ben.

Most létrehozzuk a gyökér (root) objektumot (a fő ablakunkat), amely a GridLayout widget-et tartalmazza. Régen, amikor még Glade-re fejlesztettünk GUI-t, előkerült ott egy grid view widget. Nos, a GridLayout itt gyakorlatilag ugyanaz. A rácsban létrehozott minden cella újabb widget-eket tartalmazhat. Ne feledjük, ezeket a widget-eket csak létrehozási sorrendben tudjuk elhelyezni a rácson.

```
root =
GridLayout(orientation='ver-
tical', spacing=10,
cols=1,rows=3)
```

Ebben az esetben a dolog így néz ki...

```
---------------------------
(0)             title label
---------------------------
(1)             main button
---------------------------
(2)             scrollview
---------------------------
```

A scrollview több elemet tartalmaz - a mi esetünkben gombokat. Most létrehozzuk az alkalmazásunk tetején megjelenő címkét.

```
lbl = Label(text='Transposer
Ver 0.1',
      font_size=20,
      size_hint=(None,None),
      size=(480,20),
      padding=(10,10))
```

A beállított tulajdonságok önmagukat magyarázzák, talán csak a padding és a size_hint említése indokolt. A padding az elem körüli pixelek száma x,y referencia rendszerben. A size_hint a Kivy dokumentációja alapján (ahol X ugyanaz, mint az Y):

X size_hint. Megadja, hogy mekkora helyet használhat a widget az X irányban, a szülő widgethez mérten relatívan. Csak a Layout és Window használja a hint-et. Az érték százalékban értendő és float-ként tárolódik 0. és 1. között, ahol 1. a szülő teljes méretét, 0.5 pedig annak 50%-át jelenti.

Ebben az esetben a size_hint „none"-ra van állítva, így az érték az alapértelmezett 100%, avagy 1 lesz. Ez később válik majd fontossá.

Most a „main" gombunkat határozzuk meg. Ez egy statikus referencia a skálára.

Ez a kódrészlet is könnyen értelmezhető.

Adjuk hozzá a widgeteket a root objektumhoz, azaz a GridLayout widget-hez. A címke (lbl) az első cellába kerül, a gomb (btn1) a má-

```
    def build(self):
        #-------------------------------------------------------------
        text1 = "  C  C#/Db  D  D#/Eb  E   F  F#/Gb  G  G#/Ab  A  A#/Bb  B  C"
        text2 = "  C  C#/Db  D  D#/Eb  E   F  F#/Gb  G  G#/Ab  A  A#/Bb  B  C  C#/Db  D
 D#/Eb  E   F  F#/Gb  G  G#/Ab  A  A#/Bb  B    C  C#/Db"
        #-------------------------------------------------------------
```

```
|    |    |    |    |    |    |    |    |    |    |    |    |
12345678901234567890123456789012345678901234567890123456
  C  C#/Db  E   F  F#/Gb  G  G#/Ab  A  A#/Bb  B   C
```

sodikba.

```
#----------------------------
root.add_widget(lbl)
root.add_widget(btn1)
#----------------------------
```

Most egy nehezebben értelmezhető rész következik. Egy újabb GridLayout objektumot hozunk létre és elnevezzük „s"-nek. Ezután a következő widget magasságához kötjük, ami ebben az esetben a ScrollView lesz (NEM a gombok).

```
s = GridLayout(cols=1,
spacing = 10, size_hint_y =
None)
s.bind(minimum_height=s.set-
ter('height'))
```

Létrehozunk 20 gombot, feltöltjük a text tulajdonságát és hozzáadjuk a GridLayout-hoz.

Most létrehozzuk a ScrollView-t, beállítjuk a méretét és hozzáadjuk a gyökér GridLayout-hoz.

```
sv =
ScrollView(size_hint=(None,
None), size=(600,400))

sv.center = Window.center
root.add_widget(sv)
```

Végül hozzáadjuk a GridLayout-ot, amely tartalmazza az összes gombot a ScrollView-ban és visszaadja az alkalmazásnak a root objektumot.

```
sv.add_widget(s)
return root
```

Itt van még az „if __name__" rutinunk. Vegyük észre: biztosítottuk magunknak a lehetőséget arra, hogy az alkalmazás android app-ként is használható legyen.

```
if __name__ in
 ('__main__','__andro-
id__'):

    Transpose().run()
```

Biztosan kíváncsi vagy, hogy miért használtam gombokat címkék helyett az összes szöveges objektumunkra. Azért, mert a címkék a Kivy-ben alapból nem rendelkeznek látható határokkal. Ezzel a következő részben foglalkozunk majd. Létre fogunk hozni egy exit gombot is és egy kicsit még tovább megyünk.

A forráskód megtalálható PasteBin-en:
http://pastebin.com/hsicnyt1

Találkozzunk hát legközelebb és köszönöm a velem töltött elmúlt három évet!

```
    btn1 = Button(text = "  " + text1,size=(680,40),
            size_hint=(None, None),
            halign='left',
            font_name='data/fonts/DroidSansMono.ttf',
            padding=(20,20))
```

```
for i in range(0,19):
    if i <= 12:
        if i < 10:
            t1 = " " + str(i) + "| "
        else:
            t1 = str(i) + "| "
    else:
        t1 = ''
        text2 = ''
    btn = Button(text = t1 + text2[(i*5):(i*5)+65],
           size=(680, 40),
           size_hint=(None,None),
           halign='left',
           font_name='data/fonts/DroidSansMono.ttf')
    s.add_widget(btn)
#------------------------------------------------------------
```

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

Programozzunk Pythonban – 37. rész

Ebben a hónapban befejezzük a Kivy-ben készített transzponáló programunkat. Remélhetőleg megőrizted az előző számban használt kódod, most ugyanis arra fogunk építkezni. Ha nincs meg, nézz utána az FCM 64. számában.

Kezdjük hát az előző hónapban végzett munka áttekintésével. Egy olyan alkalmazást készítettünk, amely segít a gitárosoknak a hangok gyors transzponálásában. A célunk az, hogy ez az alkalmazás Linux és Windows mellett androidon is futtatható legyen. Amikor gyakorolni megyünk a bandával, én a tabletemen szoktam magammal vinni a programot. Meg szerettem volna mutatni, hogyan készíthetünk a projektünkből egy csomagot Androidra, de mivel menet közben megváltozott néhány dolog, ezt a részt inkább a következő hónapra hagynám.

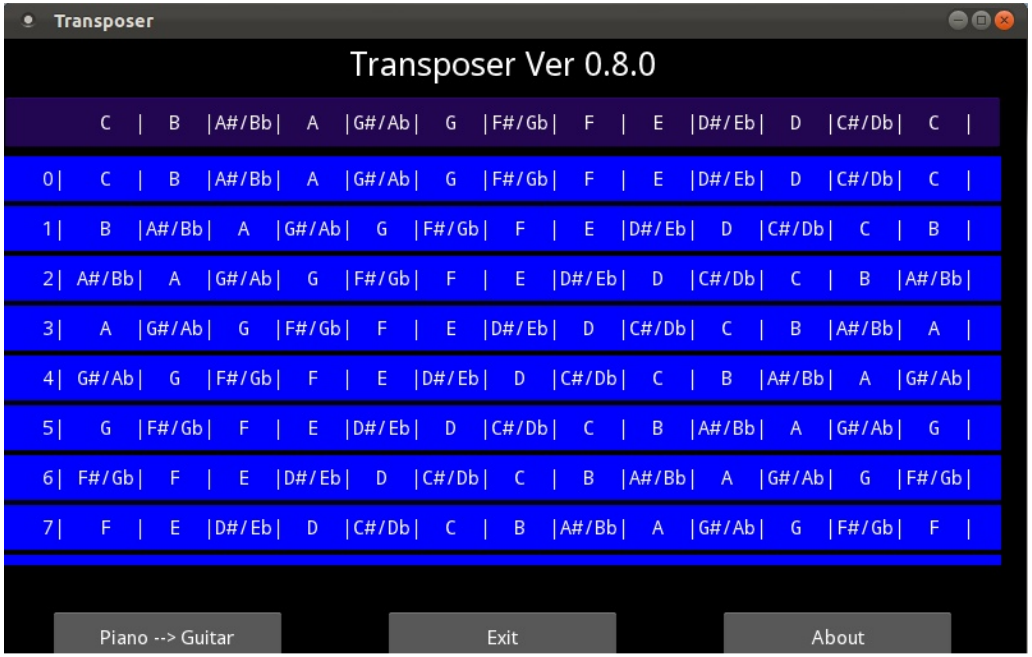
Az app-ot az alábbi állapotban hagytuk... (balra lent)

Ha majd végzünk, ilyen lesz... (jobbra lent)

A legszembetűnőbb változás talán az eredetileg unalmas, szürke címkék új, kék színe. Megjelent továbbá három gomb, a görgethető címkék pedig jobban kitöltik a rendelkezésre álló képernyő szélességét. Ránézésre ezen kívül viszont nincs változás. Az egyik gomb az „about” nevet viseli, rákattintva némi információt kapunk a programról egy új ablakban, de mi ezen keresztül megtanulhatjuk a felugró ablakok készítését. Egy másik gomb az exit (kilépés) és ezen kívül van még egy, amely a címkékben szereplő szöveg felcserélésére szolgál - ezzel a zongoráról gitárra, vagy a gitárról zongorára történő transzponálást tehetjük könnyebbé.

```
#:kivy 1.0
#:import kivy kivy

<BoundedLabel>:
      canvas.before:
      Color:
            rgb: 0, 0, 1
      Rectangle:
            pos: self.pos
            size: self.size
```



Kezdjük egy olyan .kv fájl létrehozásával, amely a színes címkékért fog felelni. Nagyon egyszerű az egész.

Az első két sor is szükséges, ezek mondják meg, hogy a Kivy melyik verzióját használjuk. Létrehozunk egy új típusú címkét, egy „BoundedLabel”-t. A színt RGB értékként (0 és 1 között, az 1 a 100 %-ot jelöli) adjuk meg és ahogy az látható, a címke kék lesz. Létrehozunk egy téglalapot is, ez lesz az aktuális címke. Mentsük el „transpose.kv" néven. A fájl nevének meg kell egyezaen a használni kívánt osztály nevével.

Ha ez megvan, a múltkor készített kódba a transpose osztály elé írjuk be az alábbi sorokat:

```
class BoundedLabel(Label):

    pass
```

Ahhoz, hogy működjön, csak egy definícióra van szükségünk. Adjuk hozzá az import részhez az alábbiakat:

```
from kivy.uix.popup import
Popup
```

Ennek segítségével később létre tudunk hozni felugró (popup) ablakokat. Most a Transpose osztály def build részébe írjuk be a következő sorokat:

```
def LoadLabels(w):
    if w == 0:
            tex0 = self.text1
            tex1 = self.text2
        else:
            tex0 = self.text3
            tex1 = self.text4
        for i in range(0,22):
            if i <= 12:
               if i < 10:
                  t1 = " " + str(i) + "| "
               else:
                  t1 = str(i) + "| "
                  t = tex1
            else:
                t1 = ''
                t = ''
            l = BoundedLabel(text=t1+t[(i*6):(i*6)+78], size=(780, 35),
                size_hint=(None,None),halign='left',
                font_name='data/fonts/DroidSansMono.ttf')
            s.add_widget(l)
```

A LoadLabels rutin adja majd a színes címkéket (BoundedLabel) és a felcserélhetőséget. Ezek a dolgok tulajdonképpen előkerültek már a múltkor is. Az értéket átadjuk a „w” paraméternek, amely meghatározza, hogy melyik szöveg jelenjen meg. Az l=BoundedLabel sorhoz hasonlóval volt már szerencsénk, a különbség csak annyi, hogy Button widget helyett BoundedLabel widgeteket használunk. A Swap rutin következik. Helyezzük a jobbra látható kódot a LoadLabels alá.

```
def Swap(instance):
    if self.whichway == 0:
        self.whichway = 1
        btnWhich.text = "Guitar --> Piano"
        btn1.text = "    " + self.text3
        s.clear_widgets()
        LoadLabels(1)
    else:
        self.whichway = 0
        btnWhich.text = "Piano --> Guitar"
        btn1.text = "    " + self.text1
        s.clear_widgets()
        LoadLabels(0)
```

```
self.whichway=0
self.text1 = "  C  |  B  |A#/Bb|  A  |G#/Ab|  G  |F#/Gb|  F  |  E  |D#/Eb|  D  |C#/Db|  C  |"
self.text2 = "  C  |  B  |A#/Bb|  A  |G#/Ab|  G  |F#/Gb|  F  |  E  |D#/Eb|  D  |C#/Db|  C  |  B  |A#/Bb|  A  |G#/Ab|  G  |F#/Gb|  F  |  E  |D#/Ab|  D  |C#/Db|  C  |"
self.text3 = "  C  |C#/Db|  D  |D#/Eb|  E  |  F  |F#/Gb|  G  |G#/Ab|  A  |A#/Bb|  B  |  C  |"
self.text4 = "  C  |C#/Db|  D  |D#/Eb|  E  |  F  |F#/Gb|  G  |G#/Ab|  A  |A#/Bb|  B  |  C  |C#/Db|  D  |D#/Eb|  E  |  F  |F#/Gb|  G  |G#/Ab|  A  |A#/Bb|  B  |  C  |C#/Db|"
```

A kód önmagát magyarázza. Egy változó (self.whichway) meghatározza, hogy „milyen módon” szeretnénk megjeleníteni a címkéket... gitárról zongorára, vagy zongoráról gitárra.

Mentsük el azt, ami már megvan, mert innen még elég sok változtatást fogunk eszközölni.

Cseréljük ki a text1-et és a text2-t meghatározó sorokat a fentebb láthatóra.

Figyeljünk arra, hogy a sorok itt rosszul tördelődnek, a programkódodban viszont helyesen kell szerepelniük. A self.whichway értékét 0-ra állítjuk, ez lesz a swap folyamatunk során az alapértelmezett. Ezután definiáljunk négy húrt (a múltkor még csak kettő volt). Feltűnhet, hogy a text3 és text4 szövege a text1 és text2 fordítottja.

Most a root-ot definiáló sort fogjuk megváltoztatni. Ezt:

```
root = GridLayout(orientation='vertical', spacing=10, cols=1,rows=3)
```

erre:

```
root = GridLayout(orientation='vertical', spacing=6, cols=1, rows=4, row_default_height=40)
```

A tagolást 10-ről 6-ra állítottuk, a sor alapértelmezett magasságát pedig 40 pixelre. Változtassuk meg a címke (következő sor) szövegét „text='Transposer Ver 0.8.0'”-ra. Minden más marad úgy, ahogy van.

Most a gombot definiáló sort változtatjuk meg, erről:

```
btn1 = Button(text = "   " + text1,size=(680,40),

 size_hint=(None,None),

 halign='left',

font_name='data/fonts/DroidSansMono.ttf',

 padding=(20,20))
```

erre:

```
btn1 = Button(text = "      " + self.text1,size=(780,20),

 size_hint=(None, None),

 halign='left',

font_name='data/fonts/DroidSansMono.ttf',

 padding=(20,2),

background_color=[0.39,0.07,.92,1])
```

Vegyük észre, hogy az első definíció formázását is megváltoztattam. A lényeges változások a méretben (680,40-ről 720,20-ra) és a gomb háttérszínénél vannak. Ne feledjük, a gombok háttérszínét meg tudjuk változtatni, a „standard” címkékét nem.

Ezután három AnchorLayout widgetet definiálunk, amelyeket később fogunk felhasználni. al0 (AnchorLayout0), al1, és al2-nek neveztem el őket. Megírjuk még a kód About Popup részét is, és definiáljuk a gombjainkat a hozzájuk tartozó kötésekkel együtt (következő oldalon balra fent).

Keressük meg az „s = GridLayout” sort és a spacing értékét változtassuk meg 10-ről 4-re, majd adjuk hozzá az alábbi sort az s.bind sor után (épp a ciklus elé):

```
LoadLabels(0)
```

Ez meg fogja hívni a LoadLabels-t az alapértelmezett 0 „which” értékkel.

Most kommenteljük ki a teljes for ciklust, azaz a „for i in range (0,19);”-től az „s.add_widget(btn)”-ig. Nincs rá szükségünk, hiszen a LoadLabels rutin mindezt megcsinálja majd nekünk.

Most mentsük el a kódot újra és próbáljuk meg lefuttatni. Egy sötét lila gombot kellene látni felül és a

```
al0 = AnchorLayout()
al1 = AnchorLayout()
al2 = AnchorLayout()
popup = Popup(title='About Transposer',
        content=Label(text='Written by G.D. Walters'),
        size_hint=(None,None),size=(400,400))
btnWhich = Button(text = "Piano --> Guitar",
           size=(180,40),size_hint=(None,None))
btnWhich.bind(on_release=Swap)
btnAbout = Button(text="About",size=(180,40),
           size_hint=(None,None))
btnAbout.bind(on_release=ShowAbout)
btnExit = Button(text="Exit", size=(180,40),
          size_hint=(None,None))
btnExit.bind(on_release=exit)
```

csinos kék BoundLabel-öket. Ezen felül észre fogjuk venni, hogy a BoundLabel-ök a görgő ablakban közelebb kerültek egymáshoz, a szöveg így sokkal olvashatóbb.

Már majdnem a végére értünk a kódnak, de néhány dologra még szükség van. Az „sv = ScrollView” sor után írjuk be az alábbiakat:

```
sv.size = (720, 320)
```

Ezzel a ScrollView widget méretét állítjuk be 720 x 320-asra - így a root ablakon belül szélesebbé válik. A „return root” sor elé még írjuk be a következő (jobbra fent látható) néhány sort.

Itt adjunk hozzá három gombot az AnchorLayout widget-hez, hozzuk létre az AnchorLayout-okat tartó GridLayout-ot és végül az AnchorLayout-okat hozzáadjuk a GridLayout-hoz.

Menjünk vissza a „def Swap” rutin alá és írjuk be az alábbiakat.

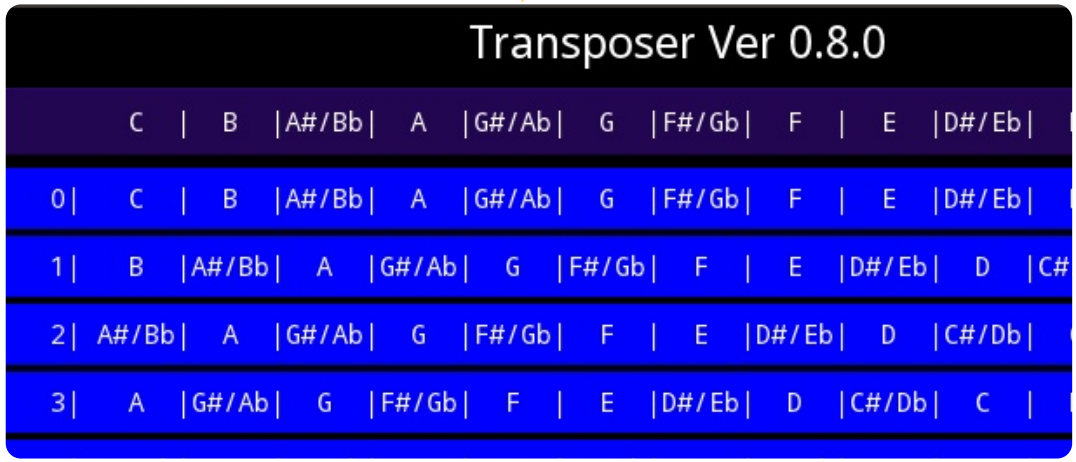
```
def ShowAbout(instance):

    popup.open()
```

Ezzel készen is vagyunk. Mentsük és futtassuk le a kódot. Ha megnyomjuk az About gombot, egy egyszerű popup-ot kapunk. Ha ezen kívülre kattintunk valahol, nyomban el is tűnik.

A kódunk tehát kész. A teljes forráskód megtalálható a

```
        al0.add_widget(btnWhich)
        al1.add_widget(btnExit)
        al2.add_widget(btnAbout)
        bgl = GridLayout(orientation='vertical',
              spacing=6, cols=3,rows=1,
              row_default_height=40)
        bgl.add_widget(al0)
        bgl.add_widget(al1)
        bgl.add_widget(al2)
```



http://pastebin.com/GftmjENs címen.

A folytatásban egy android csomagot szeretnénk majd készíteni - de erre már csak a következő alkalommal kerül sor.

Ha elébe szeretnél szaladni a dolgoknak, javaslom a http://kivy.org/docs/guide/packaging-android.html dokumentációt.

Találkozzunk legközelebb.

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

Hogyanok

Programozzunk Pythonban – 38. rész

Írta: Greg Walters

Ahogy a 37. részben ígértem, a már létrehozott transzponáló alkalmazás segítségével egy APK-t készítünk, amit majd az Andorid készülékedre telepíthetsz.

Mielőtt belevágnánk, győződjünk meg róla, hogy minden rendelkezésünkre áll. Az első amire szükségünk van, az a két file, amiket létrehoztunk egy olyan mappába melyhez könnyedén hozzáférhetünk. Nevezzük el „áthelyezőnek” (transposer). Hozd létre a saját mappádban, majd másold át a két fájlt (transpose.kv és transpose.py) ebbe a mappába. Ezután nevezd át a transpose.py-t main.py-ra. Ez a rész fontos.

Ezek után hivatkozással kell ellátni a kivy csomagolási instrukciókat egy böngészőben. A link http://kivy.org/docs/guide/packaging-android.html. Ezt használjuk a következő lépések elvégzése során nem egészen ugyanúgy, ahogy ezt a Kivy alkalmazottak szándékozták. Az android SK már rendelkezésedre áll a korábbi leírásból. Ideális esetben át fogod nézni és összeszedni az összes szoftvert, ami ott fel van sorolva, de a most nekünk kellőeket itt is végigkövetheted. Töltsd le a Phyton-for-android szoftvert. Nyiss egy terminálablakot és írd be a következőt:

```
git clone git://github.com/kivy/python-for-android
```

Ez letölti és beállítja a szoftvert, amire szükségünk van a folytatáshoz. Most a terminálablakban változtasd meg a mappát a python-for-android/dist/default mappára.

Egy build.py nevű file-t fogsz találni, ez lesz az, ami az összes munkát el fogja nekünk végezni, és most következik a csoda.

A build.py program különféle parancssori argumentumokat fog figyelembe venni és létrehozza neked az APK-t. A built.py szerkezete fentebb látható, közvetlenül a Kivy dokumentációból.

```
./build.py --dir <path to your app>
           --name "<title>"
           --package <org.of.your.app>
           --version <human version>
           --icon <path to an icon to use>
           --orientation <landscape|portrait>
           --permission <android permission like VIBRATE> (multiple allowed)
           <debug|release> <installd|installr|...>
```

A mi felhasználásunkra a következő parancsot használjuk: (a “\” sorfolytató karakter).

```
./build.py --dir ~/transposer
--package org.RainyDay.transposer \
--name "RainyDay Transposer"
--version 1.0.0 debug
```

Nézzük meg a parancs részleteit:

./build.py – ez az alkalmazás
--dir ~/transposer – ez a mappa, ahol az alkalmazás kódja található
--package org.RainyDay.transposer – ez a csomag neve
--name “RainyDay Transposer” – ez az alkalmazás neve mely meg fog jelenni az alkalmazás fiókban
--version 1.0.0 – ez az alkalmazás verziója
debug – kibocsátás mértéke (ellenőrző vagy kibocsátó)(debug vagy release).

Miután ezeket elvégezted, feltéve, ha minden a várakozásnak megfelelően működött, akkor a /bin mappában számos fájlt találsz. Az, amit éppen keresel a következő néven szerepel „RainyDayTransposer-1.0.0-debug.apk". Kedvenc fájlkezelő alkalmazásoddal átmásolhatod az android készülékedre és telepítheted, mint bármely más alkalmazást a különféle app boltokból.

Ez minden, amit időm engedett ebben a hónapban.

Közreműködnél?

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

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

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

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

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

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

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

A FULL CIRCLE-NEK SZÜKSÉGE VAN RÁD!

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

A Full Circle Csapata

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

Full Circle Magazin
Magyar Fordítócsapat

Koordinátor:
Pércsy Kornél
Fordítók:
Palotás Anna
Tömösközi Máté Ferenc

Lektor:
Balogh Péter

Szerkesztő/Korrektor:
Heim Tibor

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