full circle

AZ UBUNTU LINUX KÖZÖSSÉG FÜGGETLEN MAGAZINJA

Programozói sorozat - Különkiadás

Programozói sorozat
Különkiadás

PROGRAMOZZUNK PYTHONBAN

1. Kötet

A Full Circle magazin különkiadása

Full Circle

AZ UBUNTU LINUX KÖZÖSSÉG FÜGGETLEN MAGAZINJA

Üdvözöllek egy újabb, „egyetlen témáról szóló különkiadásban”

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

Most ez a „Programozzunk Pythonban” első nyolc részének az újabb kiadása (a magazin 27.-34. 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 – 1. rész

Írta: Greg Walters

ELŐZŐ SZÁMOK:
N/A

ITT HASZNÁLHATÓ:
ubuntu kubuntu xubuntu

KATEGÓRIÁK:
Fejlesztés Grafika Internet M/média Rendszer

ESZKÖZÖK:
CD/DVD Merevlemez USB eszköz Laptop Vezeték nélküli

A legtöbb használatban lévő programozási nyelv közül talán a Python az, amelyik a legkönnyebben elsajátítható. A nyelvet az 1980-as évek végén készítették, és azóta igen sokat fejlődött. A legtöbb Linux rendszeren már előtelepítve megtalálható, ennek ellenére a legtöbbször figyelmen kívül hagyjuk, amikor egy új nyelv megtanulása mellett döntünk. A cikkben most a parancssoros programozással foglalkozunk, a jövőben pedig eljátszadozgatunk a GUI (Graphical User Interface, azaz grafikus kezelőfelület) programozásával. Csobbanjunk is rögtön a dolgok közepébe egy egyszerű alkalmazás készítésével!

Első programunk

Néhány sornyi kódot fogunk írni egy gedit-féle szövegszerkesztővel, majd pedig megtárgyaljuk, hogy melyik sor mit végez.

Gépeljük be a következő négy sort:

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

print 'Hello.  I am a python program.'

name = raw_input("What is your name? ")

print "Hello there, " + name + "!"
```

Ennyi az egész. Mentsük el hello.py néven valahova. Én talán egy home mappabeli python_peldak nevű mappába tenném. Ez az egyszerű példa már mutatja, hogy mennyire könnyű a kódolás Pythonban. Mielőtt használnánk a programot, előbb futtathatóvá kell tennünk. Ezt a

```
chmod +x hello.py
```

parancs begépelésével érhetjük el abban a mappában, ahol a python fájlunk van. Most már elindíthatjuk a programot.

```
greg@earth:~/python_examples$
./hello.py

Hello.  I am a python program.

What is your name? Ferd Burphel

Hello there, Ferd Burphel!

greg@earth:~/python_examples$
```

Eddig mindez nem is volt túl bonyolult. Eljött az idő, hogy megnézzük a sorok jelentését.

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

Ez a rész mondja meg a rendszernek, hogy egy python programmal van dolga, és annak alapértelmezett python fordítóját kell használnia a kód futtatásához.

```
print 'Hello. I am a python
program.'
```

Egyszerűen megfogalmazva, ez a sor íratja ki a „Hello, I am a python program.” szöveget a terminálra.

```
name = raw_input("What is your
name? ")
```

Itt már egy kissé bonyolódnak a dolgok. Ennek a sornak két része is van. Az első a name =, a második a raw_input("What is your name? "). Először a utóbbit nézzünk meg. A raw_input parancs kiíratja a promptot a terminálra ("What is your name? ", ami magyarul: „Hogy hívnak? ”), majd vár a felhasználóra (rád), hogy begépeljen valamit (egy <Enter>-rel lezárva). Most nézzük az elsőt: name =. A parancs ezen részében a „name” nevű változóhoz rendelünk valamit. Mi egy változó? Képzeljük úgy el, mint egy cipősdobozt. Egy cipősdobozt tárgyak tárolására lehet használni -- cipők, számítógép alkatrészek, papírok, stb. A cipősdoboz számára nem lényeges, hogy mi is van benne – csak dolgok tárolására használjuk. Ebben az esetben azt tartalmazza, amit begépelünk. Esetemben, a Fred Burphel nevet adtam neki. A Python itt

csak fogja ezt, majd eltárolja a „name” nevű cipősdobozba, későbbi felhasználásra.

```
print "Hello there, " + name +
"!"
```

Még egyszer használjuk a print parancsot valami megjelenítésére a képernyőn – ebben az esetben ez a „Hello there, ” („Hali, ”), plusz még a „name” változó tartalma (akármi is legyen benne), illetve még egy felkiáltójel van a végén. Ebben a sorban három információdarab összerakására koncentráltunk: a „Hello there”, a „name” tartalma és a felkiáltójel.

Most pedig szakítsunk egy kis időt néhány dolog alaposabb megértésére, mielőtt továbblépnénk a következő példára. Nyissunk meg egy terminált és gépeljük be:

```
python
```

Valami ilyesmit kellene kapnunk:

```
greg@earth:~/python_examples$
python

Python 2.5.2 (r252:60911, Oct  5
2008, 19:24:49)

[GCC 4.3.2] on linux2
Type "help", "copyright", "cre-
dits" or "license" for more in-
formation.

>>>
```

Most a python felületén vagyunk. Ettől kezdve sok dolgot tudunk majd véghezvinni, de előbb nézzük meg, hogy valójában mik is vannak itt. Az első dolog, amit észre fogunk venni az a python verziószáma -- az enyém 2.5.2. A következő az az, hogy a súgóért a „help”-et kell begépelni. Ezt majd kipróbálhatjuk később. Most azonban gépeljük be:

```
print 2+2
```

és üssünk Enter-t. A következőt kapjuk vissza:

```
>>> print 2+2
4
>>>
```

Remélem feltűnt, hogy a „print” szót kisbetűsen írtuk. Mi történne, ha „Print 2+2”-őt írnánk? A fordító az alábbi módon válaszolna:

```
>>> Print 2+2
  File "<stdin>", line 1
    Print 2+2
          ^
SyntaxError: invalid syntax
>>>
```

Ez azért van, mert amíg a „print” szó egy létező parancs, addig a „Print” nem ismert. A kis- és nagybetűs különbségek igen fontosak a Pythonban.

Most pedig játszadozzunk el egy kicsit a változókkal. Írjuk be:

```
var = 2+2
```

Nem fogunk semmilyen változást észlelni azon kívül, hogy a Python prompt (a '>>>') visszatér. Nincs is semmi gond, mert azt mondtuk a Pythonnak, hogy hozzon létre egy változót („cipősdobozt”), amit var-nak neveztünk el, és pakolja bele a „2+2” eredményét. Ahhoz, hogy megnézhessük a var tartalmát a

```
print var
```

parancsot kell begépelni Enter-rel lezárva.

```
>>> print var
4
>>>
```

Mostantól újra meg újra fel tudjuk használni a var-t a 4-es szám helyett úgy, ahogy itt is:

```
>>> print var * 2
8
>>>
```

Ha ismét begépeljük a „print var”-t, akkor az alábbit kapjuk:

```
>>> print var
4
>>>
```

A var értéke nem változott. Még mindig a 2+2 eredményét, azaz a 4-et tárolja.

Mindez természetesen csak egy egyszerűbb programozási gyakorlat kezdők számára. A bonyolultság az elkövetkező cikkek folyamán nőni fog. Nézzünk most még néhány példát változókra.

Írjuk be az értelmezőbe:

```
>>> strng = 'The time has come
for all good men to come to the
aid of the party!'

>>> print strng

The time has come for all good
men to come to the aid of the
party!

>>>
```

Létrehoztunk egy „strng” (a string rövidítéseként) nevű változót melyben a „The time has come for all good men to come to the aid of the party!” értéket (magyarul: „Eljött az idő minden jó ember számá-

ra, hogy bulizni induljon!”) tároljuk. Mostantól (amíg a fordítónak ebben a futtatásában vagyunk), a strng változónk tartalma ugyanaz lesz, hacsak meg nem változtatjuk. Mi fog történni, ha ezt a változót 4-gyel megszorozzuk?

```
>>> print strng * 4

The time has come for all good men to come to the aid of the party!The time has come for all good men to come to the aid of the party!The time has come for all good men to come to the aid of the party!The time has come for all good men to come to the aid of the party!

>>>
```

Nos, nem éppen az, amit vártunk, ugye? Kiíratta az strng értékét négyszer. Hogy miért? Hát, a fordító tudta, hogy az strng egy karakterlánc és nem egy megszámlálható érték. Matematikai műveleteket nem lehet sztringeken végrehajtani.

Mi történne akkor, ha lenne egy s nevű változónk, ami a „4”-et tartalmazza, ahogy itt is:

```
>>> s = '4'
>>> print s
4
```

Úgy tűnik, mintha az s az integer (azaz egész) 4-et tárolná, de valójában annak a karakteres reprezentációjáról van szó. Tehát, ha begépeljük a „print s * 4”-et, akkor az alábbi kimenetet fogjuk kapni:

```
>>> print s*4
4444
>>>
```

Ismét csak tudta a fordító, hogy az s egy sztring, nem pedig egy szám. Tudja, mert szimpla idézőjelek (') közét tettük a 4-et, ezzel hozva létre egy karaktert.

Ezt be is tudjuk bizonyítani a print type(s) begépelésével. Ekkor láthatjuk, hogy mit is gondol a rendszer egy változó típusáról.

```
>>> print type(s)
<type 'str'>
>>>
```

Pont úgy van, ahogy mondtam. Ez egy sztring típus. Ha számértékként akarjuk használni, akkor a következőképpen járhatunk el:

```
>>> print int(s) * 4
16
>>>
```

Az (s) sztringet, ami a „4”, most már átváltotta egy integerré, és meg lehet szorozni 4-el, hogy 16-ot kapjunk.

Most már ismerjük a print és a raw_input parancsokat, illetve változókhoz tudunk értéket rendelni, továbbá tudjuk azt is, hogy mi a különbség a sztringek és az integerek között.

Lépjünk még egy kicsit tovább. A Python fordítójába gépeljük be a quit()-et a parancssorba való visszatéréshez.

Egyszerű For ciklus

Eljött az idő arra, hogy megismerkedjünk egy egyszerű programozási ciklussal. Térjünk vissza a szövegszerkesztőnkbe és írjuk meg az alábbi programot:

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

for cntr in range(0,10):

    print cntr
```

Fontos, hogy tabulátort használjuk a „print cntr” sorban, mivel a Python sem zárójeleket „(”, sem kapcsos zárójeleket „{” nem használ a blokkok elválasztásához, mint ahogy azt más hasonló nyelvek teszik. Ehelyett itt indentálást (behúzást) kell használunk.

Mentsük el a programot „for_loop.py” néven. Mielőtt kipróbálnánk, meg kell beszélnünk, hogy mi is valójában egy ciklus.

A ciklus egy olyan kód, ami egy megadott utasítást, vagy utasítások egy halmazát többször végrehajtja. Programunk ebben az esetben 10-szer fogja megismételni a cntr (mint counter, azaz számláló) kiíratását. Magyarul, azt mondtuk a rendszernek, hogy „add a cntr változónak a 0 értéket, majd ismételd meg 10-szer a cntr változó tartalmának kiíratását úgy, hogy mindig egyet adj hozzá, és ismételd ezt újra”. Elég egyszerűnek tűnik. A kód „range(0,10)” része azt jelenti, hogy kezdjen 0-val és hajtsa végre újra addig, amíg a cntr értéke 10 nem lesz, majd lépjen ki.

Most is, mint az előbb, használjuk a

```
chmod +x for_loop.py
```

parancsot, majd futtassuk a

```
./for_loop.py
```

fájlt a terminálban.

```
greg@earth:~/python_examples$
./for_loop.py
0
1
2
3
4
5
6
7
8
9
greg@earth:~/python_examples$
```

Úgy tűnik, ez működik is, de miért csak 9-ig számol és nem 10-ig? Nézzük meg a kimenetet megint. Tíz szám sorozatát íratjuk ki, amely nullával kezdődik és a kilencessel ér véget. Éppen ezt akartuk – írassa ki a cntr értékét 10-szer úgy, hogy mindig egyet ad a változóhoz, majd lépjen ki, ahogy az értéke 10 lesz.

Most, hogy mindezt már értjük, láthatjuk, hogy a programozás lehet egyszerű, de ugyanakkor bonyolult is. Ezért is tudnunk kell mindig, hogy mit kérünk éppen a géptől. Ha megváltoztatnánk a range parancsot mondjuk „range(1,10)”-re, akkor 1-től kezdene el számolni, de 9-nél ismét csak megállna, mivel amint a cntr értéke 10 lesz, kilép. Ahhoz, hogy az „1,2,3,4,5,6,7,8,9,10” számsort írja ki, a range(1,11) parancsot kell használnunk – mivel a for-ciklus akkor lép ki, amikor a felső határt elérte. Vegyük továbbá észre az állítás szintakszisát: „for változó in range(kezdőérték,végérték):”. A „:” azt jelenti, hogy egy olyan kódblokkot kezdünk, amit indentálni kell. Nagyon fontos, hogy emlékezzünk a kettőspontra, mindig addig kell indentálni a kódot, amíg a blokknak vége nem lesz.

Ha az alábbi módon módosítjuk programunkat:

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

for cntr in range(1,11):

      print cntr

print 'All Done'
```

akkor egy ilyen kimenetet kapunk:

```
greg@earth:~/python_examples$
./for_loop.py
1
2
3
4
5
6
7
8
9
10
All Done
greg@earth:~/python_examples$
```

Ne felejtsünk el helyesen indentálni, mivel ily módon tudatjuk a géppel, hogy egy új blokkról van szó. A következő alkalommal jobban bele fogunk mélyedni a blokkok indentációjába.

Ez minden, amit az első alkalomra tartogattam. Legközelebb egy kis ismétlés után továbblépünk néhány újabb python programozási útmutatóval. Addig is jó lenne, ha feltelepítenénk egy python specifikus szerkesztőt, mint pl a Dr. Python, vagy az SPE (Stani's Python Editor), melyek a Synapticból is elérhetők.

Greg Walters a RainyDay Solutions tulajdonosa, amely korlátolt felelősségű tanácsadó cég a Colorado-i Aurorában. Programozással 1972 óta foglalkozik. Szeret főzni, hegyeket mászni, zenét hallgatni és szabadidejét családja körében tölteni.

Programozzunk Pythonban – 2. rész

Írta: Greg Walters

ELŐZŐ SZÁMOK:
FCM 27. szám
Python – 1. rész

ITT HASZNÁLHATÓ:
ubuntu kubuntu xubuntu

KATEGÓRIÁK:
Fejlesztés Grafika Internet M/média Rendszer

ESZKÖZÖK:
CD/DVD Merevlemez USB eszköz Laptop Vezeték nélküli

Helyesbítés az első részhez

David Turnertől kaptam egy e-mailt, amiben azt javasolta, hogy a Tab billentyű indentálásra (behúzásra) való használata félrevezető lehet, mivel néhány szerkesztő négynél több, vagy kevesebb szóközt is használhat behúzásonként. Ez természetesen igaz. Sok Python programozó (köztük én is) a Tab billentyű négy szóközre való beállításával spórol az idején. A gond azonban az, hogy mások szerkesztője nem feltétlenül fogja ugyanezt a beállítást használni, ami pedig ronda kódhoz és más hasonló hibákhoz vezethet. Magyarul, inkább a szóközök használatára szokj rá, mint a Tab-okéra.

Előző számunkban egy olyan egyszerű programot néztünk meg, ami a „raw_input”-ot használta a felhasználói visszajelzés beolvasásához, illetve néhány egyszerű változótípust, meg egy egyszerű „for” ciklust is tartalmazott. Ez alkalommal mélyebbre fogunk ásni a változók világában, plusz írunk még néhány programot is.

LISTÁK

Először is kezdjük a listáknak nevezett változótípusokkal. Más nyelvekben egy listát tömbnek tekintenénk. Ha visszatérünk a cipősdoboz analógiájához, akkor egy tömb (vagy lista) sok, tárgyakat tartalmazó doboz egymás oldalához ragasztásának felelne meg. Például: a villákat az egyik dobozban tárolnánk, a késeket a másikban és a kanalakat a harmadikban. Lássunk is egy egyszerű listát. A könnyű értelmezés végett ez most a hónapok neveit fogja tárolni. Ennek a kódja így néz ki:

```
months =
['Jan','Feb','Mar','Apr','May'
,'Jun','Jul','Aug','Sep','Oct'
,'Nov','Dec']
```

A lista létrehozásához annak összes értékét szögletes nyitó- és zárójelek („[” és „]”) közé helyezzük. A listánkat „months” (hónapok) néven hoztuk létre. Ahhoz, hogy használjuk, mondjuk a months[0] vagy months[1] parancsot írnánk be (ami a „Jan” vagy a „Feb” szavakat adná eredményül). Fontos megjegyezni, hogy mindig nullától számolunk. A lista hosszának megállapításához az alábbi parancsot kell használnunk:

```
print len(months)
```

ami 12-vel tér vissza.

A listákra egy másik példa a receptkönyvek kategóriái lennének. Mondjuk így:

```
categories = ['Main
dish','Meat','Fish','So-
up','Cookies']
```

Ekkor a categories[0] a „Main dish” (főétel), a categories[4] pedig „Cookies” (sütik) lennének. Ezek megint csak elég nyilvánvalóak. Biztos vagyok benne, hogy sok olyan dolgot ki tudtok találni, amire egy listát lehet használni.

Mindeddig olyan listákat készítettünk, amik karakterláncokat tároltak. Olyan listát is létre tudunk hozni, ami egész számokat tárol. Ha visszagondolunk a hónapos listára, olyat is tudnánk csinálni, ami a hónapok napjainak a számát tárolná:

```
DaysInMonth =
[31,28,31,30,31,30,31,31,30,31
,30,31]
```

Ha kiíratnánk a DaysInMonth[1] (Február) elemet, akkor a 28-at kapnánk vissza. Észrevehetjük, hogy a listanév a DaysInMonth (hónap napjai). Ugyanilyen egyszerűen használhattam volna „daysinmonth”-ot is, vagy csak egyszerűen „X”-et... De akkor nem lenne olyan jól olvasható a kód. A helyes programozási szokások azt diktálják (ami értelmezés kérdése is), hogy a változónevek könnyen olvashatóak

legyenek. Arról, hogy miért is olyan fontos ez, majd a későbbiekben lesz szó. A listákkal még eljátszogatunk egy kicsit később.

Mielőtt azonban rátérnénk a következő példaprogramra, nézzünk még néhány pythonos dolgot.

A sztringek rejtelmei

Még az első részben röviden bemutattam a karakterláncokat. Most nézzük meg őket közelebbről. Egy sztring karakterek sorozata. Nem sokkal több ennél. Ami azt illeti, tekintsünk úgy egy karakterláncra, mint karakterek tömbjére. Például, ha a „The time has come” (eljött az idő) szringet egy strng nevű változóhoz rendeljük hozzá, és meg szeretnénk tudni, hogy mi a második karaktere, akkor a

```
strng = 'The time has come'
print strng[1]
```

utasításokat írnánk be. Az eredmény egy „h” lenne. Emlékezzünk, hogy mindig nullától számolunk, tehát az első karakter a [0], a második az [1], a harmadik a [2], és így tovább. Ha meg szeretnénk keresni a negyedik pozíciótól nyolcadikig tartó karaktereket, akkor a

```
print strng[4:8]
```

parancsot írnánk be, ami a „time” szóval térne vissza. Mint az első cikkben bemutatott „for” ciklusban, a számlálás nyolcnál megáll, de a nyolcadik karakter már nem adódik vissza, ami itt a szóköz lenne a „time” szó után.

Sztringünk hosszúságának lekérdezéséhez használhatjuk a len() függvényt:

```
print len(strng)
```

ami 17-el tér vissza. Ha viszont azt akarnánk megtudni, hogy a karakterláncunkban a „time” szó hol található, akkor a

```
pos = strng.find('time')
```

parancsot alkalmazhatjuk. Itt a pos (mint position, azaz pozíció) változó a 4-et tartalmazza, ami azt jelenti, hogy a „time” szó a negyedik pozíción kezdődik. Ha olyan szóra vagy szavakra keresnénk rá, amik nincsenek a sztringben, mint ahogy itt is:

```
pos = strng.find('apples')
```

a posba bekerülő érték -1 lenne. Azt is meg tudjuk csinálni, hogy a „split” paranccsal minden egyes szót külön kiíratunk. Ezzel több részre szedjük (vagy bontjuk) a karakterláncot minden egyes szóköznél:

```
print strng.split(' ')
```

A parancs a ['The','time','has', 'come'] listával tér vissza. A split egy igen jól használható dolog. Még sok ilyen beépített sztringkezelő függvény áll rendelkezésünkre, amiket a későbbiekben is használni fogunk.

Változó helyettesítés

Van még egy dolog, amit be kell mutatnunk, mielőtt rátérnénk a következő programozási példára. Amikor egy olyan dolgot szeretnénk kiíratni, ami konstans és változó szöveget is tartalmaz, akkor a változóhelyettesítés nevű eszközt használhatjuk. Alkalmazása igen egyszerű. Ha helyettesíteni akarunk egy sztringet, akkor a '%s'-et használjuk, majd a Pythonnak megadjuk, hogy mire kell lecserélnie. Például ahhoz, hogy egy hónapot kiírassunk a fenti listából, a

```
print 'Month = %s' % month[0]
```

parancsot használhatjuk, ami kiírja a „Month = Jan” szöveget. Ha egy integert szeretnénk helyettesíteni, akkor a '%d'-t használjuk. Nézzük meg az alábbi esetet:

```
Months =
['Jan','Feb','Mar','Apr','May'
,'Jun','Jul','Aug','Sep','Oct'
,'Nov','Dec']
DaysInMonth =
[31,28,31,30,31,30,31,31,30,31
,30,31]
for cntr in range(0,12):
      print '%s has %d days.'
% (Months[cntr],DaysIn-
Month[cntr])
```

Ennek a kódnak az eredménye:

```
Jan has 31 days.
Feb has 28 days.
Mar has 31 days.
Apr has 30 days.
May has 31 days.
Jun has 30 days.
Jul has 31 days.
Aug has 31 days.
Sep has 30 days.
Oct has 31 days.
Nov has 30 days.
Dec has 31 days.
```

Nagyon fontos, hogy megértsük az egyszeres és kétszeres idézőjelek használatát. Ha egy

változóhoz az alábbi karakterláncokat rendeljük:

```
st = 'The time has come'
```

vagy

```
st = “The time has come”
```

akkor az eredmény ugyanaz. Azonban, ha aposztrófokat kell használni a szövegen belül, mint itt is:

```
st = 'He said he's on his way'
```

akkor szintaktikai hibát kapunk. Helyesen ezt így kell csinálni:

```
st = “He said he's on his way”
```

Gondoljunk erre úgy, hogy egy sztring meghatározásánál a szöveget mindig valamilyen idézőjelek közé kell tenni – egyet az elejére, egyet a végére –, mindezt úgy, hogy azok megegyezőek legyenek. Ha keverni kell az idézőjeleket, akkor mindig azt használjuk a külső helyeken, amelyből nincs a szövegben. Viszont felmerülhet olyan is, amikor a sztring valami ilyesmi: “She said “Don't Worry”” (Azt mondta, hogy ne aggódjak). Ebben az esetben ezt így is meg lehet adni:

```
st = 'She said “Don\'t Worry”'
```

Vegyük észre, hogy egy backslash van a „Don't” szóban lévő egyes aposztróf előtt. Ez egy úgynevezett escape karakter. Ez mondja meg a Pythonnak, hogy írasson ki (ebben az esetben) egy egyszeres idézőjelet – anélkül, hogy figyelembe venné a sztring határolójeleit. Néhány másik escape szekvencia még (a teljesség igénye nélkül) a „\n”, ami az új sor megfelelője, vagy a „\t”, ami pedig a tabulátoré. Egy későbbi példakódban ezek még elő fognak kerülni.

Hozzárendelés kontra egyenlőség

Még mindig van néhány dolog, amit meg kell beszélnünk, hogy a következő példát értelmezni tudjuk. Az első ilyen dolog a hozzárendelés és az egyenlőség közötti különbség. A hozzárendelést már sokszor használtuk a példákban. Amikor egy változóhoz értéket akarunk rendelni, akkor a hozzárendelés operátort használjuk, vagyis az „=” szimbólumot (egyenlőségjelet):

```
variable = value
```

Viszont abban az esetben, amikor meg akarjuk állapítani egy változó egyenlőségét valamivel, akkor az összehasonlítás operátort kell alkalmaznunk. Mondjuk le szeretnénk ellenőrizni, hogy egy változó egyenlő-e egy értékkel. Ekkor a „==” jeleket (két egyenlőségjel) használnánk:

```
variable == value
```

Tegyük fel, hogy van egy loop (ciklus) nevezetű változónk és meg akarjuk tudni róla, hogy egyenlő-e például 12-vel:

```
if loop == 12:
```

Nem szükséges értenünk még az if és a kettőspont jelentését. Csak arra kell emlékeznünk, hogy az egyenlőség megállapításához két egyenlőségjelet használunk.

Kommentek

A következő témánk a megjegyzéseké. A kommentek sok minden miatt fontosak. Segítségükkel nem csak értelmezni tudjuk, hogy valaki mit is akar végrehajtani a kódban, hanem arra is, ha mondjuk hat hónap után előveszed egy programodat, emlékeztetnek majd arra, hogy mit is akartál csinálni. Amikor elkezdesz rengeteg programot készíteni, ez nagyon fontos lesz. A megjegyzések arra is jók, hogy használatukkal a Python figyelmen kívül hagyhat bizonyos sorokat. Egy sor kikommentelésére a „#” jelet használjuk:

```
# Ez egy komment
```

Bárhova rakhatunk kommenteket egy sorban, de ha ezt teszszük, akkor a Python minden „#” mögötti dolgot figyelmen kívül fog hagyni.

If utasítások

Most visszatérünk ahhoz az „if” utasításhoz, amiről már az előbb volt szó. Ha valamit egy dolog értékének függvényében akarunk eldönteni, akkor az „if” kulcsszót kell használnunk:

```
if loop == 12:
```

Ezzel leellenőrizzük a „loop” változó értékét, ha annak tartalma 12, akkor végrehajtjuk az

alatta lévő indentált blokk tartalmát. A legtöbb esetben ez épp elég is, de mi van akkor, ha azt akarjuk mondani, hogy „ha ez a változó valami, akkor csináld ezt, különben pedig azt”. Pszeudokódban ezt így írhatnánk:

```
if x == y then
      csinálj valamit
else
      csinálj valami mást
```

Pythonba ugyanez:

```
if x == y:
      csinálj valamit
else:
      csinálj valami mást
      meg még pár dolgot
```

A fontosabb dolgok:

1. Az 'if' vagy az 'else' utasítást kettősponttal kell lezárni.

2. INDENTÁLD a kódsorokat.

Ha egymás után több dolgot is le kell ellenőriznünk, akkor használhatjuk az if/elif/else formátumot. Például:

```
x = 5
if x == 1:
      print 'X is 1'
elif x < 6:
      print 'X is less than 6'
elif x < 10:
      print 'X is less than 10'
else:
      print 'X is 10 or greater'
```

Figyeljük meg, hogy a „<” operátort használtuk ahhoz, hogy megállapítsuk, vajon az x kisebb-e egy megadott értéknél – ebben az esetben 6-nál vagy 10-nél. Más gyakori összehasonlítási műveletek a nagyobb mint („>”), a kisebb vagy egyenlő mint („<=”), a nagyobb vagy egyenlő mint („>=”) és a nem egyenlő („!=”) operátorok.

While utasítások

Végezetül megnézünk egy egyszerű példát a „while” ciklusokra. A „while” utasítás segítségével egy olyan ciklust tudunk létrehozni, ami addig ismétlődik, amíg egy megadott küszöbértéket el nem ér. Egy könnyen érthető példa az lenne, ha a „loop” változóhoz 1-et rendelnénk úgy, hogy a „while” ciklus addig íratná ki a loop változó értékét, amíg az kisebb vagy egyenlő, mint tíz, és minden iterációban eggyel növeli a loop értékét. Ha a loop tíznél nagyobb, akkor kilép:

```
loop = 1
while loop <= 10:
      print loop
      loop = loop + 1
```

A futtatás eredménye terminálban:

```
1
2
3
4
5
6
7
8
9
10
```

Éppen az, amit vártunk. Az első ábrán (jobbra fent) egy hasonló, de valamivel bonyolultabb példa látható, ami azért még mindig elég egyszerű.

Ebben a példában, az „if” utasítást a „while” ciklussal, a „raw_input” utasítással, újsor escape szekvenciával, hozzárendelés és összehasonlítás operátorokkal kevertük – összesen nyolc sornyi programban.

```
loop = 1
while loop == 1:
      response = raw_input("Enter something or 'quit' to end => ")
      if response == 'quit':
            print 'quitting'
            loop = 0
      else:
            print 'You typed %s' % response
```

1. ábra

A példa végrehajtása az alábbi kimenetet hozná létre:

```
Enter something or 'quit' to
end
 => FROG
You typed FROG
Enter something or 'quit' to
end
 => bird
You typed bird
Enter something or 'quit' to
end
 => 42
You typed 42
Enter something or 'quit' to
end
 => QUIT
You typed QUIT
Enter something or 'quit' to
end
 => quit
quitting
```

Figyeljük meg, hogy amikor a „QUIT”-et írtuk be, a program nem állt meg. Ez azért van, mert a response (válasz) változót a

„quit”-el hasonlítottuk össze (response == 'quit'). A „QUIT” pedig NEM egyenlő a „quit”-tel.

Még egy gyors példa, mielőtt lezárnánk e havi adagunkat. Tegyük fel, hogy meg akarjuk nézni, vajon egy felhasználó jogosult-e programunk használatára. Ez a példa pont jó arra, hogy megmutassa, mit is tanultunk eddig, habár éppenséggel nem a legmegfelelőbb módja a probléma megoldásának. Egyszerűen el fogjuk kérni a felhasználótól a nevét és jelszavát, majd összehasonlítjuk azzal, ami a kódunkban van, és döntünk az eredmény függvényében. Két listát fogunk használni – az egyikben a jogosult felhasználók, a másikban azok jelszavai lesznek. Ezt követően a raw_inputtal bekérjük a felhasználó adatait, végezetül pedig az if/elif/else utasításokkal leellenőrizzük a felhasználót. Tartsuk azonban szem előtt, hogy a gyakorlatban nem így szoktunk eljárni. A soron következő cikkekben rátérünk más módszerekre is. A kód a jobbra levő szövegdobozban látható.

Mentsük is „python_test.py” néven, majd futtassuk különböző bemenetekre.

```
#-----------------------------------------------
#password_test.py
#    example of if/else, lists, assignments,raw_input,
#    comments and evaluations
#-----------------------------------------------
# Assign the users and passwords
users = ['Fred','John','Steve','Ann','Mary']
passwords = ['access','dog','12345','kids','qwerty']
#-----------------------------------------------
# Get username and password
usrname = raw_input('Enter your username => ')
pwd  = raw_input('Enter your password => ')
#-----------------------------------------------
# Check to see if user is in the list
if usrname in users:
     position = users.index(usrname) #Get the position in the list of the users
     if pwd == passwords[position]:  #Find the password at position
          print 'Hi there, %s.  Access granted.' % usrname
     else:
          print 'Password incorrect.  Access denied.'
else:
     print "Sorry...I don't recognize you.  Access denied."
```

2. ábra

Az egyedüli dolog, amiről még nem beszéltünk, az a leellenőrző rutin, ami az 'if usrname in users:' utasítással kezdődik. Amit itt csinálunk, az az, hogy megnézzük, vajon a begépelt felhasználónév benne van-e a listánkban. Majd használjuk a users.index(usrname) függvényt a listában elfoglalt helyének meghatározásához, hogy ezzel ki tudjuk olvasni a jelszavát is, ami ugyanezen a pozíción van a másik listában. Például: John az első pozíción van a felhasználónév listában. Jelszava a „dog”, ami szintén az első pozíción található a jelszavak listájában. Ezzel a módszerrel meg tudjuk találni az egyes párokat. Innentől már a többi egyértelmű.

Ennyi elég is volt erre a hónapra. A következő alkalommal a függvényekről és modulokról fogunk tanulni. Addig is játszadozzatok el a tanultakkal, és érezzétek jól magatokat.

Greg Walters a RainyDay Solutions tulajdonosa, ami egy korlátolt felelősségű tanácsadó cég a coloradói Aurorában. Programozással 1972 óta foglalkozik. Szeret főzni, túrázni, zenét hallgatni és szabadidejét családja körében eltölteni.

Programozzunk Pythonban – 3. rész

Írta: Greg Walters

ELŐZŐ SZÁMOK:
FCM 27–28. szám
Python – 1–2. rész

ITT HASZNÁLHATÓ:
ubuntu kubuntu xubuntu

KATEGÓRIÁK:
Fejlesztés Grafika Internet M/média Rendszer

ESZKÖZÖK:
CD/DVD Merevlemez USB eszköz Laptop Vezeték nélküli

Az előző cikkben megtanulhattunk változókat behelyettesíteni, megjegyzéseket elhelyezni, és azt is tudjuk, hogy mi a hozzárendelés és az egyenlőség közötti különbség, valamint ismerjük az if, illetve a while utasításokat is. Továbbá megígértem, hogy ebben a részben a modulokról és a függvényekről egyaránt szót fogunk ejteni.

Modulok

Modulok használatával a Python nyelvet bővíthetjük, akár újakat létrehozva, vagy felhasználva a Pythonnal együtt telepítetteket, illetve más felhasználók által létrehozottakat. A Python maga, sok száz különböző modullal rendelkezik, amik a programozási folyamatot nagyban megkönnyítik. A Python általános moduljainak listáját a http://docs.python.org/modindex.html címen találhatjátok meg. Néhány modul operációs rendszertől függő, de a legtöbb teljesen platformfüggetlen (ugyanúgy lehet Linux, Mac és Windows alatt használni). Ahhoz, hogy egy külső modult használjunk, először be kell importálni azt a programunkba. A Pythonban lévő modulok egyike az úgynevezett „random”. Segítségével pszeudovéletlen számokat tudunk generálni. A jobbra fenn látható modult fogjuk az első példában használni.

Vizsgáljunk meg minden egyes kódsort. Az első négy sor megjegyzés. Ezekről már szó volt az előző cikkben. Az ötödik sor megmondja a Pythonnak, hogy használja a random modult. Ezt mindig külön meg kell mondani a Pythonnak.

```
#=======================================
# random_example.py
# Module example using the random module
#=======================================
import random
# print 14 random integers
for cntr in range(1,15):
      print random.randint(1,10)
```

A hetedik sor beállít egy „for” ciklust 14 véletlengenerált szám kiírásához. A nyolcadik sor használja a randint() függvényt, hogy 0 és 10 között egy véletlen számot megjelenítsen. Figyeljük meg, hogy jelezni kell a függvény moduljának nevét a Pythonban. Ezt úgy tesszük (ebben az esetben), hogy random.randint. Felmerülhet azonban a kérdés, hogy egyáltalán miért jó modulokat készíteni? Nos, ha minden egyes függvényt belevennénk a Pythonba, irgalmatlanul nagy és lassú lenne, a debuggolás egy rémálommá válna. A modulok használatával részekre tudjuk bontani kódunkat úgy, hogy minden egyes darab egy bizonyos feladathoz van rendelve. Ha például nincs szükséged adatbáziskezelő funkciókra, nem kell tudnod róla, hogy van egy SQLite nevű modul. De ha kell, akkor már ott is van. (Ami azt illeti, fogunk is használni adatbázis modulokat a jövőben.)

Amint igazán belelendülsz a Python programozásába, valószínűleg saját modulokat fogsz létrehozni, így téve lehetővé kódod későbbi felhasználását anélkül, hogy újra be kellene azt gépelned. Ha esetleg valamit változtatnod kellene a kódban, anélkül teheted meg, hogy túl nagy kockázatot vállalnál a fő programrész elrontására. Vannak azonban bizonyos korlátok is, de ebben majd később mélyedünk el. Amikor korábban az „import random” utasítást használtuk, azt mondtuk meg a Pythonnak, hogy használjon minden függvényt a random modulon belül. Ha azon-

ban csak a randint()-re van szükségünk, átírhatjuk az import utasítást így:

```
from random import randint
```

Ha most hívjuk meg a függvényt, akkor már nem kell használnunk a „random” azonosítót. A kódunk tehát így fog kinézni:

```
from random import randint
# print 14 random integers
for cntr in range(1,15):
      print randint(1,10)
```

Függvények

Amikor a „random” modult importáltuk, a „randint()” függvényt használtuk. A függvény egy olyan kódblokk, amit – általában – többszöri meghívásra terveztek, ezáltal könnyebb a kezelése: megóv minket ugyanannak a kódnak az újbóli begépelésétől. Nagyon erős általánosítással élve, amikor ugyanazt a kódot meg kell írnunk egynél többször, akkor az egy jó alkalom a függvényírásra. Bár a következő két példa nagyon egyszerű, mégis jól bemutatja a függvények használatát. Tegyük fel, hogy szeretnénk két számot összeadni, megszorozni őket és kivonni egymásból, végül megjeleníteni a megadott számokat és az eredményeket minden alkalommal. Nem minden habostorta: mindezt háromszor kell megismételnünk, három eltérő számhalmazzal. Leegyszerűsített példánk végül úgy néz ki, amint az a jobb oldali szövegben látható.

```
#silly example
print 'Adding the two numbers %d and %d = %d ' % (1,2,1+2)
print 'Multiplying the two numbers %d and %d = %d ' % (1,2,1*2)
print 'Subtracting the two numbers %d and %d = %d ' % (1,2,1-2)
print '\n'
print 'Adding the two numbers %d and %d = %d ' % (1,4,1+4)
print 'Multiplying the two numbers %d and %d = %d ' % (1,4,1*4)
print 'Subtracting the two numbers %d and %d = %d ' % (1,4,1-4)
print '\n'
print 'Adding the two numbers %d and %d = %d ' % (10,5,10+5)
print 'Multiplying the two numbers %d and %d = %d ' % (10,5,10*5)
print 'Subtracting the two numbers %d and %d = %d ' % (10,5,10-5)
print '\n'
```

A függvények alkalmazása nélküli nagy, ömlesztett kódhalmaz nemcsak a több gépelés miatt vezethet sok hibához, de a későbbi változtatások miatt is, hiszen ugyanazt a változtatást egyszerre több helyen is ugyanúgy el kell végezni a program kódjában. Ehelyett készíteni fogunk egy „DoTwo” nevű függvényt, amely a megadott két számmal elvégzi a szükséges matematikai műveleteket és az eredményt kiírja. A függvényt a „def” kulcsszó beírásával kezdjük (ami megmondja a programnak, hogy definiálni fogunk egy függvényt). A „def” után hozzáadjuk a függvény nevét és a paraméterek listáját (ha vannak) zárójelek között. A sort egy kettősponttal (:) zárjuk. A következő sortól a függvény törzsét képező kódot be kell húzni (beljebb kell írni, indentálni). Továbbfejlesztett, második kis példánk az alábbiakban tekinthető meg.

Mint látható, sokkal kevesebb gépelésre volt szükség: egész pontosan 12 sor helyett csak nyolcra. Ha később valamilyen változtatásra van szükség függvényünkben, könnyebben megtehetjük, anélkül, hogy a program fő részében bármit megváltoztatnánk. Függvényünk hasz-

```
#silly example 2...still silly, but better
def DoTwo(num1,num2):
      print 'Adding the two numbers %d and %d = %d ' % (num1,num2,num1+num2)
      print 'Multiplying the two numbers %d and %d = %d ' % (num1,num2,num1*num2)
      print 'Subtracting the two numbers %d and %d = %d ' % (num1,num2,num1-num2)
      print '\n'

DoTwo(1,2)
DoTwo(1,4)
DoTwo(10,5)
```

nálatakor a meghívás a függvény nevével és az utána írt argumentumokkal történik.

A függvény további alkalmazására tekintsük a következőket.

Szeretnénk egy olyan programot írni, mely ki fogja írni a megvásárolt termékek listáját szép, formázott módon, az alábbi szövegkiíráshoz hasonlóan.

Az árucikkek egyenkénti és összegzett ára dollárban és centben lesz formázva. A megjelenítés szélessége változtatható kell hogy legyen. A bal- és jobboldali értékek szintén változóból kell hogy jöjjenek. Három függvényt fogunk elkészíteni a feladat megoldására. Az első kiírja a legfelső és a legalsó sort, a második kiírja az egyes cikkek részleteit az összegző sorral együtt, a harmadik pedig az elválasztó vonalat jeleníti meg. Szerencsére a Python sok olyan beépített jellemzővel bír, amely könnyebbé teszi a feladatunkat. Ha visszaemlékezünk, programunkban kiírattuk a szöveget négyszer, és az ugyanannak a sztringnek a négy másolatával tért vissza.

```
'+==============================+'
'| Item 1                 X.XX  |'
'| Item 2                 X.XX  |'
'|------------------------------|'
'| Total                  X.XX  |'
'+==============================+'
```

Ezt mi kihasználhatjuk a saját javunkra. A legfelső, vagy a legalsó sor kiíratásához vegyük a kívánt szélességet, vonjunk ki belőle kettőt a két '+' jel miatt és használjuk a „ '=' * (width-2)” formulát. A dolgok még könnyebbé tételéhez a változóbehelyettesítést fogjuk használni, hogy az összes elemet egy sorba tegyük. Tehát kiírandó sztringünket a ('+',('=' * width-2)),'+') karaktersorral lehet lekódolni. Függvényünk közvetlenül is írhatna a képernyőre, azonban ehelyett a return kulcsszóval visszaadjuk a létrehozott sztringet a hívó félnek. Függvényünket nevezzük el „TopOrBottom”-nak, és a kódját írjuk a következők szerint:

```
def TopOrBottom(width):
  # width is total width of returned line
  return '%s%s%s' % ('+',('=' * (width-2)),'+')
```

A megjegyzést kihagyhatnánk, de mégis jó, ha megtudhatjuk belőle egy pillanat alatt a „width” paraméter rendeltetését. A függvény meghívása a „print TopOrBottom(40)” módon történik, amelyben szélességnek természetesen más értéket is beállíthatunk. Most tehát már van egy függvényünk, mely a két említett sorral foglalkozik. Létrehozhatunk egy új függvényt, hogy az elválasztó vonallal foglalkozzon ugyanannak a kódnak a felhasználásával... VAGY módosíthatnánk a már meglevő függvényünket is, hogy egy további paraméterrel az alkalmazandó karaktert meghatározhassuk. Tegyük inkább ezt, neve pedig még mindig maradhat „TopOrBottom”.

```
def TopOrBottom(character,width):
      # width is total width of returned line
      # character is the character to be placed between the '+' characters
      return '%s%s%s' % ('+',(character * (width-2)),'+')
```

Most látható, hogy a megjegyzések milyen hasznosak. Ne feledjük, hogy a generált sztringgel visszatérünk, tehát nekünk kell valamit tennünk vele, hogy visszakapjuk, amikor meghívjuk. Ahelyett, hogy hozzárendelnénk egy másik sztringhez, egyszerűen csak kiírjuk a képernyőre. Használata így néz ki:

```
print TopOrBottom('=',40)
```

Tehát most nem csak három sorról gondoskodtunk, hanem a szükséges rutinok számát is csökkentettük háromról kettőre. Tehát innentől már csak a középső rész kiírásával kell törődnünk.

Az ehhez szükséges új függvényt nevezzük „Fmt”-nek. Négy paramétert fogunk neki átadni:
val1 – a bal oldali kiírandó érték,
leftbit – ennek az „oszlop”-nak a szélessége,
val2 – a jobb oldali kiírandó érték (lebegőpontos),
rightbit – ennek az „oszlop”-nak a szélessége.

Az első feladat az, hogy az információkat a jobb oldalon megformázzuk. Mivel dollár és cent értéket akarunk megformázni, használhatunk egy speciális változóbehelyettesítési függvényt, amely azt az utasítást adja ki, hogy az érték lebegőpontos számként íródjon ki, az n. számú

helyen, a tizedes ponttól jobbra. A parancs pedig a „%2.f”. Ezt a „part2” változóhoz fogjuk hozzárendelni. Ehhez kódunk a „part2 = '%.2 f' % val2” lesz. Használhatunk olyan függvényeket is még, amik a Python sztringekbe alapértelmezetten be vannak építve „ljust” és „rjust” néven. Az „ljust” balra igazítja a sztringet, kitöltve azt a jobb oldalon, bármilyen kívánt karakterrel. Az „rjust” ugyanazt teszi, csak a kitöltés a bal oldalon van. Ez már így szép darab. A helyettesítések segítségével összedobtunk egy nagy sztringet és visszaadtuk azt a hívó kódnak. Következő sorunk így néz ki:

```
return 'ss' % ('| 
',val1.ljust(leftbit-2,' 
'),part2.rjust(rightbit-2,' 
'),' |')
```

Miközben ez inkább ijesztőnek tűnik elsőre, elemezzük ki egy kicsit, és nézzük meg, valójában milyen egyszerű is:

return – az elkészült sztring, melyet visszaküldünk a hívó kódnak.

'ss' – 4 értéket fogunk tárolni egy sztringben. Mindegyik „%s” egy helyőrző.

% (– Elindítja a változó listát.

'| ', – Kiírja ezeket literálisan.

val1.ljust(leftbit-2,' ') – Veszi a 'val1' változót, melyet átadtunk, balra igazítja szóköz karakterekkel (leftbit-2). Ki kell vonni 2-t ahhoz, hogy a "|" a bal oldalon megfelelően legyen.

part2.rjust(rightbit-2,' ') – Jobbra igazítja az ár formázott sztringjét rightbit-2 mennyiségű szóközzel. A ' |' fejezi be a sztringet.

Ez minden, amit tennünk kellett. Még valójában egyes hibák ellenőrzését kellene elvégeznünk, vehetjük azt saját játékunknak is. Tehát az Fmt függvényünk valójában csak két sornyi kód a definíció során és a megjegyzéseken kívül. A meghívás az alábbiak szerint történik:

```
print Fmt('Item 
1',30,item1,10)
```

A visszatérési értéket ismét egy másik szöveghez rendelhetnénk, de mi csak ki fogjuk íratni.

```
+====================================+
| Item 1                        3.00 |
| Item 2                       15.00 |
+------------------------------------+
| Total                        18.00 |
+====================================+
```

Figyeljük meg, hogy adunk át 30-at a bal oldali rész, és 10-et a jobb oldal szélességének. Ez megegyezik a 40-nel, melyet a TopOrBottom rutinunknak korábban megadtunk. Majd indítsuk el szerkesztőprogramunkat és írjuk be az alábbi kódot.

Mentsük el a kódot „pprint1.py” néven és futtassuk

```
#pprint1.py
#Example of semi-useful functions

def TopOrBottom(character,width):
     # width is total width of returned line
     return '%s%s%s' % ('+',(character * (width-2)),'+')

def Fmt(val1,leftbit,val2,rightbit):
     # prints two values padded with spaces
     # val1 is thing to print on left, val2 is thing to print on right
     # leftbit is width of left portion, rightbit is width of right portion
     part2 = '%.2f' % val2
     return '%s%s%s%s' % ('| ',val1.ljust(leftbit-2,' '),part2.rjust(rightbit-2,' '),' |')
# Define the prices of each item
item1 = 3.00
item2 = 15.00
# Now print everything out...
print TopOrBottom('=',40)
print Fmt('Item 1',30,item1,10)
print Fmt('Item 2',30,item2,10)
print TopOrBottom('-',40)
print Fmt('Total',30,item1+item2,10)
print TopOrBottom('=',40)
```

le. A kimenet valahogy úgy néz ki, ahogy az előző oldalon látható jobbra fent.

Noha ez egy nagyon egyszerű példa, mégis tippeket ad, hogy miért és hogyan kell használni a függvényeket. Most pedig fejlesszük tovább egy kicsit, és tudjunk meg többet a listákról. Emlékszünk a második részre, amikor először tárgyaltuk a listákat? Egy dolog, amit még nem mondtam, hogy egy lista tartalmazhat szinte bármit, beleértve a listákat is. Definiáljunk programunkban egy új „itms” nevű listát és töltsük fel így:

```
itms = [['Soda',1.45],['Can-
dy',.75],['Bre-
ad',1.95],['Milk',2.59]]
```

Ha ezt a szokásos listaként kezelnénk, akkor a „print itms[0]” módon érnénk el az elemeit. Azonban erre a ['Soda', 1,45] térne vissza, ami nem igazán az, amit normális körülmények között kerestünk. El szeretnénk érni minden egyes elemet az első listában. Tehát a „print itms[0][0]” formát kellene használnunk, hogy megszerezzük a „Soda”-t és [0][1]-et, hogy megkapjuk a költségeket, vagyis az 1,45-öt. Szóval, most már négy elemünk van, amit már megvásároltunk, és ezt az információt szeretnénk használni szép kiíró rutinunkban. Az egyetlen dolog, amit meg kell változtatni, a program alja. Mentsük el a legutóbbi programot „pprint2.py”-ként, majd tegyük megjegyzésbe a két itemx definíciót, és tegyük bele a fenti listát. Mindennek valahogy így kell kinéznie :

```
#item1 = 3.00
#item2 = 15.00
itms = [['Soda',1.45],['Can-
dy',.75],['Bre-
ad',1.95],['Milk',2.59]]
```

Ezután távolítsuk el az összes sort, mely az Fmt()-t hívja meg. Utána adjuk hozzá az alábbi sorokat (#NEW LINE segítségével a végére), hogy a kód úgy nézzen ki, mint a képen jobbra látható.

```
itms = [['Soda',1.45],['Candy',.75],['Bread',1.95],['Milk',2.59]]

print TopOrBottom('=',40)

total = 0 #NEW LINE
for cntr in range(0,4): #NEW LINE
      print Fmt(itms[cntr][0],30,itms[cntr][1],10) #NEW LINE
      total += itms[cntr][1] #NEW LINE
print TopOrBottom('-',40)
print Fmt('Total',30,total,10) #CHANGED LINE
print TopOrBottom('=',40)
```

Vegyünk fel még egy számláló változót, amely végigmegy a lista minden elemén a ciklus végéig. Figyeljük meg a „total” nevű változó alkalmazását is. A „total”-t 0-ra állítjuk, mielőtt a ciklus elkezdődne. Aztán ahogy az eladott tételeket kiírjuk, hozzáadjuk a költséget a „total”-hoz. Végül kiíratjuk a „total”-t közvetlenül az elválasztó vonal után. Mentsük el a programot és futtassuk le azt. Ilyesmit kell látnunk, mint ami az alábbi képen látható.

```
+======================================+
| Soda                           1.45  |
| Candy                          0.75  |
| Bread                          1.95  |
| Milk                           2.59  |
+--------------------------------------+
| Total                          6.74  |
+======================================+
```

Ha nagyon elvetemültek akarunk lenni, akkor hozzáadhatunk egy sort az adó számára is. Közel azonos módon kezeljük, mint azt tettük a „total” sorában, de használjuk a (total * .086)-ot költségként.

```
print Fmt('Tax:',30,
total*.086,10)
```

Több elemet is felvehetünk a listára, és megnézhetjük, hogyan működik.

Ennyi volt mára. A következő alkalommal az osztályokkal fogunk foglalkozni.

Jó szórakozást!

Greg Walters a RainyDay Solutions tulajdonosa, ami egy korlátolt felelősségű tanácsadó cég a coloradói Aurorában. Programozással 1972 óta foglalkozik. Szeret főzni, túrázni, zenét hallgatni és szabadidejét családja körében eltölteni.

Programozzunk Pythonban – 4. rész

Írta: Greg Walters

ELŐZŐ SZÁMOK:
FCM 27–29. szám
Python – 1–3. rész

ITT HASZNÁLHATÓ:
ubuntu kubuntu xubuntu

KATEGÓRIÁK:
Fejlesztés Grafika Internet M/média Rendszer

ESZKÖZÖK:
CD/DVD Merevlemez USB eszköz Laptop Vezeték nélküli

```
class Dog():
    def __init__(self,dogname,dogcolor,dogheight,dogbuild,dogmood,dogage):
        #here we setup the attributes of our dog
        self.name = dogname
        self.color = dogcolor
        self.height = dogheight
        self.build = dogbuild
        self.mood = dogmood
        self.age = dogage
        self.Hungry = False
        self.Tired = False
```

Előző alkalommal megígértem, hogy az osztályokról fogunk beszélni, tehát erre fogunk most koncentrálni. De mik is az osztályok és mire valók?

Az osztály az egyik módja annak, hogy objektumokat hozzunk létre. Egy objektum pedig egy módszer arra, hogy állapotokat és viselkedéseket egy csoportként kezelhessünk. Tudom, hogy egy kissé kuszának tűnik, de el fogom magyarázni. Gondoljunk erre csak úgy, mintha az objektumok a valós élet modelljei lennének. Az osztályok pedig a módszerek, amivel mindezeket implementáljuk. Például, legyen otthon három kutyánk. Egy beagle, egy labrador és egy német/ausztrál juhász keverék. Mindhárom kutya, de mindegyik különbözik. Vannak közös tulajdonságaik, de mindegyiknek van még ezenkívül csak rá jellemző tulajdonságai is. Például, a kopó egy kicsi, pufók, barna és morcos kutya. A labrador közepes méretű, fekete és nagyon nyugodt természetű. A német/ausztrál juhász keverék pedig magas, sovány, fekete és a kelleténél egy kicsit bolondabb. Első ránézésre vannak nyilvánvaló tulajdonságaik. A kicsi/közepes/magas méret például a magasságukra vonatkozik. A morcos, kényelmes és bolond meg a természetüket jellemzi. Ha a cselekedeteik felől nézzük a dolgokat, akkor figyelembe vehetjük az evést, az alvást, a játékot és hasonló dolgokat.

Mindhárom a „Dog” (Kutya) osztályba tartozik. Visszatérve az előbb használt tulajdonságokhoz, vannak Dog.Name (Kutya.Név), Dog.Height (Kutya.Magasság), Dog.Build (Kutya.Testalkat, sovány, pufók stb.) és Dog.Color (Kutya.Szín) jellemzőik. Vannak viselkedéseink is, mint a Dog.Bark (Kutya.Ugat), Dog.Eat (Kutya.Eszik), Dog.Sleep (Kutya.Alszik) és így tovább.

Mint ahogy előzőleg is mondtam, mindegyik kutya különbözik is a fajtára vonatkozó tulajdonságokban. Mindegyik külön alosztálya lenne a Dog (Kutya) osztálynak. Egy diagramon ezt így ábrázolhatnánk:

```
          /--Beagle
Dog ---|-- Lab
          \--Shepherd/Heeler
```

Mindegyik alosztály örökli a Dog osztály tulajdonságait. Ezért, ha létrehoznánk egy példányt a Beagle osztályból, megkapná a szülő osztály összes tulajdonságát is, amik ebben az esetben ugye a Dog-ra vonatkozók lennének.

```
Beagle = Dog()
Beagle.Name = 'Archie'
Beagle.Height = 'Short'
Beagle.Build = 'Chubby'
Beagle.Color = 'Brown'
```

Kezdenek értelmet nyerni a dolgok? Készítsük akkor el a durván felvázolt Dog osztályunkat (fenn). A „class” kulcsszóval és az osztály nevével kezdünk.

Mielőtt még továbblépnénk kódunkban, vegyük észre, hogy a függvény, amit itt definiáltunk, az __init__ metódus (két aláhúzás + 'init' + két aláhúzás) egy inicializációs függvény, mely bármelyik osztálynál működik. Amint kódunkban meghívjuk az osztályunkat, ez a függvény le fog futni. Jelen esetben beállítunk néhány alap paramétert: van egy név, szín, testalkat, hangulat, kor, és még néhány változó, mint Hungry (Éhes) és Tired (Fáradt). Még vissza fogunk térni ezekre. Írjunk még néhány sornyi kódot.

```
Beagle = Dog('Ar-
chie','Brown','Short','Chub-
by','Grumpy',12)
print Beagle.name
print Beagle.color
print Beagle.mood
print Beagle.Hungry
```

Ez egy INDENTÁLATLAN kód, ami az osztályon kívül helyezkedik el. Ez az a kód, ami használja az osztályunkat. Az első sor létrehoz egy példányt a kutya osztályból, amit Beagle-nek neveztünk el. Ezt nevezzük példányosításnak. Amikor ezt tettük, továbbítottunk még bizonyos információkat az osztály példánya felé, mint a Beagle neve, színe, stb. A következő négy sor csak egyszerű lekérdezésekből áll, melyek a Beagle objektumról adnak információt. Itt az idő még egy kis kódra. Adjuk még hozzá a jobb felső dobozban látható kódot az __init__ függvény után.

Most pedig hívjuk meg a Beagle.Eat()-tel vagy Beagle.Sleep()-pel. Adjunk hozzá egy újabb metódust. Hívjuk ezt Bark-nak (Ugatás). A kódja jobbra látható.

Ezt egy kissé rugalmasabbá tettem. A kutya hangulatától függően az ugatás meg fog változni. A következő oldalon látható az eddigi teljes kód.

Nos, ha ezt futtatjuk, akkor a következőt kellene kapnunk:

```
My name is Archie
My color is Brown
My mood is Grumpy
```

```
    def Eat(self):
        if self.Hungry:
            print 'Yum Yum...Num Num'
            self.Hungry = False
        else:
            print 'Sniff Sniff...Not Hungry'
    def Sleep(self):
        print 'ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ'
        self.Tired = False
```

```
    def Bark(self):
        if self.mood == 'Grumpy':
            print 'GRRRRR...Woof Woof'
        elif self.mood == 'Laid Back':
            print 'Yawn...ok...Woof'
        elif self.mood == 'Crazy':
            print 'Bark Bark Bark Bark Bark Bark Bark'
        else:
            print 'Woof Woof'
```

```
I am hungry = False
Sniff Sniff...Not Hungry
Yum Yum...Num Num
GRRRRR...Woof Woof
```

Ezzel le is tudtuk a morcos kis kopónkat. De előbb azt mondtam, hogy három kutyánk van. Mivel az osztályunkat előretekintően kódoltuk, annyi lesz csak a dolgunk, hogy még kétszer példányosítjuk a kutya osztályt.

```
Lab = Dog('Nina','Black','Me-
dium','Heavy','Laid Back',7)
Heeler =
Dog('Bear','Black','Tall','Ski
nny','Crazy',9)
print 'My Name is %s' %
Lab.name
print 'My color is %s' %
Lab.color
print 'My Mood is %s' %
Lab.mood
print 'I am hungry = %s' %
Lab.Hungry
Lab.Bark()
Heeler.Bark()
```

Figyeljük meg, hogy mindkét kutya példányát a kiíratások előtt hoztuk létre. Ez nem gond, mivel a példányokat a metódusok hívását megelőzően definiáltam. Itt a teljes kimenete a kutya osztályos programunknak.

```
My name is Archie
My color is Brown
My mood is Grumpy
I am hungry = False
Sniff Sniff...Not Hungry
Yum Yum...Num Num
GRRRRR...Woof Woof
My Name is Nina
My color is Black
My Mood is Laid Back
I am hungry = False
Yawn...ok...Woof
Bark Bark Bark Bark Bark Bark
Bark
```

Mivel már ismerjük az alapokat, a házi feladat az lenne, hogy bővítsük a kutya osztályt több metódussal, mint a Play (Játszik) vagy EncounterStrangeDog (TalálkozásIdegenKutyával) vagy valami hasonlóval.

A következő alkalommal belekezdünk a GUI, avagy a grafikus kezelőfelület programozásába. Ehhez a ***Boa Constructor***-t fogjuk használni.

Greg Walters a RainyDay Solutions, LLC, az Aurora Colorado-i tanácsadó cég tulajdonosa, és már 1972 óta foglalkozik programozással. Szeret főzni, túrázni, szereti a zenét és családjával tölteni szabadidejét.

```
class Dog():
    def __init__(self,dogname,dogcolor,dogheight,dogbuild,dogmood,dogage):
        #here we setup the attributes of our dog
        self.name = dogname
        self.color = dogcolor
        self.height = dogheight
        self.build = dogbuild
        self.mood = dogmood
        self.age = dogage
        self.Hungry = False
        self.Tired = False

    def Eat(self):
        if self.Hungry:
            print 'Yum Yum...Num Num'
            self.Hungry = False
        else:
            print 'Sniff Sniff...Not Hungry'

    def Sleep(self):
        print 'ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ'
        self.Tired = False

    def Bark(self):
        if self.mood == 'Grumpy':
            print 'GRRRRR...Woof Woof'
        elif self.mood == 'Laid Back':
            print 'Yawn...ok...Woof'
        elif self.mood == 'Crazy':
            print 'Bark Bark Bark Bark Bark Bark Bark'
        else:
            print 'Woof Woof'

Beagle = Dog('Archie','Brown','Short','Chubby','Grumpy',12)
print 'My name is %s' % Beagle.name
print 'My color is %s' % Beagle.color
print 'My mood is %s' % Beagle.mood
print 'I am hungry = %s' % Beagle.Hungry
Beagle.Eat()
Beagle.Hungry = True
Beagle.Eat()
Beagle.Bark()
```

Programozzunk Pythonban - 5. rész

Írta: Greg Walters

ELŐZŐ SZÁMOK:
FCM 27-30. szám
Python - 1-4. rész

ITT HASZNÁLHATÓ:
ubuntu kubuntu xubuntu

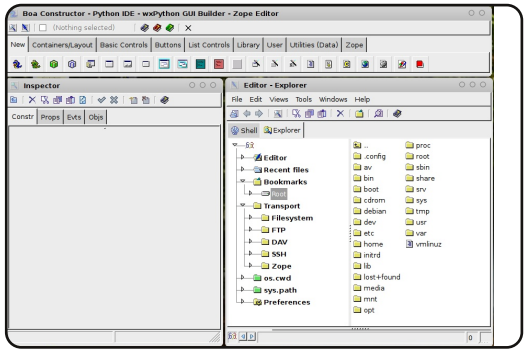
KATEGÓRIÁK:
Fejlesztés Grafika Internet M/média Rendszer

ESZKÖZÖK:
CD/DVD Merevlemez USB Eszköz Laptop Vezeték nélküli

Ha olyan vagy mint én, akkor UTÁLNI fogod ennek a cikknek az első felét. Mert én KI NEM ÁLLHATOM, amikor a szerző azt írja, hogy minden szót kétszer is végig kell olvasnom a könyvében/fejezetében/cikkében, mivel már ekkor tudom, hogy dögunalmas lesz - még úgy is, hogy valószínűleg igaza van, és az én érdekemet szolgálná, meg a végén mégiscsak végig fogok menni rajta újra.

Én figyelmeztettelek! Most arra KÉRLEK, hogy figyelmesen olvasd el a következő unalmas dolgokat. Hamarosan rátérünk az izgalmasabb részekre, de mielőtt akárcsak beszélhetnénk is programozásról, egy kis alapozásra lesz szükségünk.

ELŐSZÖR is fel kellene telepíteni a Boa Constructor és wxPython programokat. Használd a Synaptic-ot és válaszd ki mind a wxPythont, mind a Boa Constructort. Amikor ezeket feltelepítetted, a Boát az Applications/Programming/Boa Constructor menüben találhatjuk meg. Indítsuk is el rögtön. Így sokkal tisztábbak lesznek a dolgok. Amint az alkalmazás elindult, három különböző ablakot (vagy frame-et) figyelhetünk meg: egyet felül és kettőt alul. Valószínűleg át kell majd mozgatni és méretezni őket, de jussunk el arra a pontra, hogy a képen láthatóan nézzen ki.



A felső ablakot tool frame-nek (eszköz panel) nevezzük. A bal alsó az inspector frame (felügyelő panel) és a jobb alsó az editor frame (szerkesztő panel). Az editor frame-en több fajta fül van (New, Containers/Layout, stb.), melyek segítségével új projekteket tudunk létrehozni, további frame-eket tudunk egy már létező projekthez hozzáadni, illetve különböző vezérlőelemeket az alkalmazásunk frame-jeiben elhelyezni. Az inspector frame igen fontos lesz, amint vezérlőemeket kezdünk az alkalmazásunkhoz adni. Az editor frame-en tudjuk a kódunkat szerkeszteni, elmenteni a projektünket, stb. Visszatérve a tool frame-re, nézzük meg mindegyik fület - a „New” füllel kezdve. Annak ellenére, hogy sok-sok választási lehetőség áll itt rendelkezésünkre, csak kettővel fogunk foglalkozni. Ezek balról az ötödik és hatodik gombok: a wx.App és a wx.Frame. A wx.App segítségével egy teljes alkalmazást tudunk létrehozni két automatikusan legenerált fájllal. Az egyik egy frame fájl, a másik pedig az alkalmazás állománya. Én ezt a módszert szeretem használni. A wx.Frame további frame-ek alkalmazásunkhoz való hozzáadására szolgál és/vagy egy különálló alkalmazás egyetlen forráskódból való létrehozására. Erről még később ejtünk szót.

Most nézzük meg a Containers/Layout fület. Sok finomságot találhatunk itt. A legtöbbször használt a wx.Panel (balról az első) és a sizerek (méretezők - 2,3,4,5 és 6 jobbról). A Basic Controls alatt találhatjuk a statikus szöveget (label), szövegdobozokat, jelölődobozokat, rádiógombokat, stb. A Buttons alatt pedig a különféle gombok vannak. A List Controls-ban vannak a táblázatok és a lenyíló listák. Most ugorjunk a Utilities-re, ahol sok időzítőt és menüelemet láthatunk.

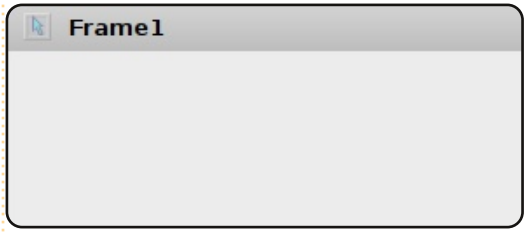
Van néhány dolog, amit nem árt, ha észben tartunk, mielőtt első alkalmazásunkhoz hozzálátnánk. Van néhány bug a Li-

nuxos verzióban. Az egyik az, hogy NÉHÁNY vezérlőelemet nem tudunk a designer nézetben mozgatni. Használd a <Ctrl>+Nyíl gombokat pozíciójuk megváltoztatásához és finomhangolásához. Egy másik hibával akkor találkozhatunk, amikor a Boa Constructor beépített tutorialjait akarjuk kipróbálni - amikor egy panelt próbálunk elhelyezni, nem nagyon lehet azt látni. Ilyenkor a kis dobozkákat kell keresni (hamarosan lesz szó róla). Használhatnánk az Inspector frame Objs fülét is a kiválasztásához.

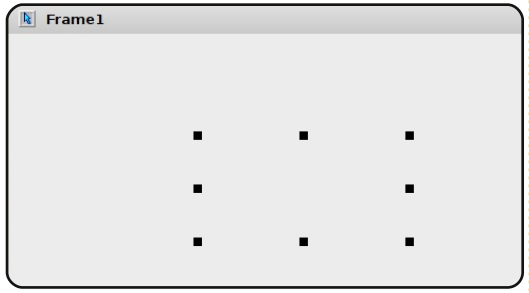
Akkor hát, indul a móka. Az editor frame-en a „New” fül alatt válasszuk ki a wx.Appot (5. balról). Ezzel létrehoztunk két új fület a szerkesztő panelen: az egyik „*(App1)*” nevű, a másik pedig „*(Frame1)*”. Ha hiszed, ha nem, a LEGELSŐ dolog, amit tenni fogunk, hogy elmentjük a két új fájlunkat a Frame1-el kezdve. A Mentés gomb az editor frame-en az ötödik balról. A felugró „Save As” ablak azt kéri, hogy adjuk meg a fájl mentési helyét, illetve nevét. Készítsünk egy GuiTests nevű mappát a home mappánkban és mentsük el bele a fájlt „Frame1.py” néven. Figyeljük meg, hogy a „*(Frame1)*” fül mostmár „Frame1” lett. (A „*(” jelzi, hogy a fájl még nincs elmentve.) Csináljuk meg ugyanezt az App1 fülre is.

Most vizsgáljunk meg néhány gombot az Editor eszköztáron. A fontosabbak a Save (5. balról) és a Run (Sárga nyíl, 7. balról). Ha egy frame fülön vagy (Frame1 például), akkor lesz még pár gomb, amiről illik tudnunk. Egyelőre beszéljünk csak a Designer gombról: 

Ez egy fontos gomb. Segítségével tudjuk a GUI-nkat megtervezni - épp az, amit csinálni szeretnénk. Amikor rákattintunk, egy üres frame-et fogunk kapni:



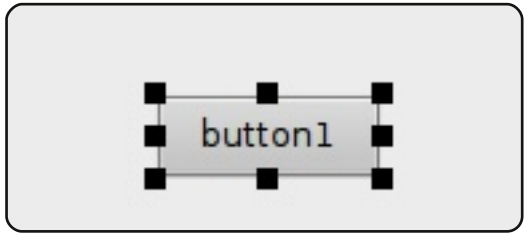
Ez egy olyan üres háttérfelület (canvas), amire bármilyen vezérlőelemet rá lehet pakolni (természetesen az észszerűség határain belül). Az első dolog, amit meg kell tennünk, hogy helyezzünk el egy wx.panel elemet. Majdnem minden cikk, amit olvastam, azt javasolja, hogy vezérlőelemeket ne helyezzünk el (a wx.panel kivételével) közvetlenül a frame-en. Kattintsunk hát a Tool Frame-en a Containers/Layout fülre, majd pedig a wx.Panel gombra. Utána menjünk vissza az új frame-re, amin épp dolgozunk és kattintsunk bele valahova. Akkor fogjuk tudni, hogy működött, ha valami ilyesmit kapunk:



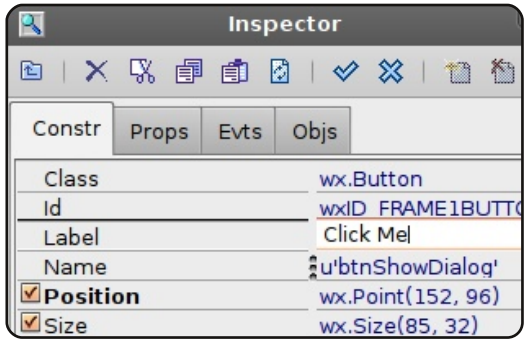
Emlékszünk még a figyelmeztetésemre a bugokkal kapcsolatban? Nos, ez pont egy ilyen eset. De ne aggódjunk! Látjuk a nyolc kis fekete négyzetet? Ezek jelzik a panel határait. Ha akarnánk, akkor rájuk kattintva át tudnánk méretezni a panelt, azonban most mi azt akarjuk, hogy az egész frame-et befedje a panel. Ehhez egyszerűen méretezzük át a FRAME-et. Most már van egy panelünk, amire a többi vezérlőeszközöket pakolhatjuk. Mozgassuk a frame-et amin dolgozunk úgy, hogy láthassuk az Editor frame eszközsorát. Két új gomb is megjelent: egy pipa és egy „X”. Az X megnyomásával eldobhatjuk a változtatásainkat.

A pipa gomb:  amit „Post”-nak hívnak. Megnyomásával a változtatások beleíródnak a frame fájljába. Még mindig el kell majd mentenünk azt, de ezzel helyezzük el magukat az új dolgokat a fájlban. Akkor nyomjuk is meg a Post gombot. Van egy másik post gomb az Inspector frame-en is, de ezzel később fogunk foglalkozni. Most mentsük el a fájt.

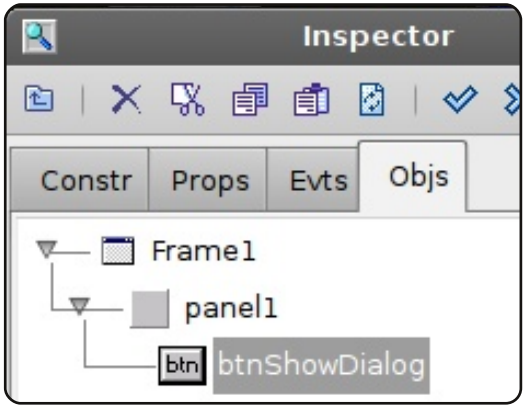
Ugorjunk vissza Design módba. Klikkeljünk a „Buttons” fülre a Tool frame-en, majd az első gombra balról, ami a wx.Button nevet viseli. Helyezzük el valahol a frame-ünk közepén, hogy valami ilyesmit kapjunk:



Figyeljük meg, hogy akár csak a panelnél, nyolc kis négyzet van körülötte. Ezek méretező fogantyúk. Továbbá azt is megmutatják, hogy melyik vezérlőelem van kijelölve éppen. Ahhoz, hogy közelebb vigyük a frame közepéhez, nyomjuk le a Control gombot (Ctrl), és amíg az le van nyomva, használjuk a nyíl billentyűket a mozgatásához. Most nézzük meg az Inspector frame-et. Négy fülünk van. Kattintsunk a „Constr” fülre. Itt tudjuk megváltoztatni a gomb címkéjét, nevét, pozícióját, méretét és stílusát. Egyelőre csak változtassuk meg a gomb nevét „btnShowDialog”-ra és a Label property-t (címke tulajdonság) „Click Me”-re (Kattints Rám).

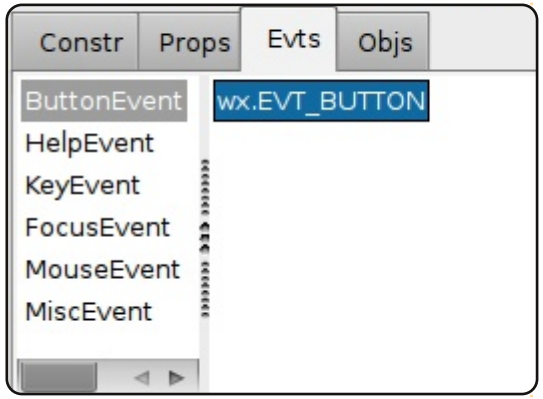


Most ugorjunk át a fül maradékán és menjünk az Objs fülre. Ez a fül mutatja az összes általunk használt vezérlőelemet, illetve a köztük lévő szülő/gyerek kapcsolatokat. Mint ahogy láthatjuk, a gomb a panel1 leszármazottja, ami pedig a Frame1 gyereke.



Postoljunk (pipa gomb) és mentsük el a változtatásokat. Ismét menjünk vissza designer nézetbe és figyeljük meg, hogy (feltételezem, hogy még mindig az „Objs” fül van kiválasztva az inspector frame-en) a Frame1 van most kijelölve. Ez jó, mert épp rá lesz szükségünk. Menjünk vissza a „Constr” fülre és változtassuk meg a „Frame1” címet „Our First GUI”-ra (Első GUI-nk). Postoljuk és mentsünk még egyszer. Futtassuk az alkalmazást. Nyomjuk meg a sárga Run gombot az Editor frame-en.

Ám kattintgathatunk annyiszor a gombra, amennyiszer jól esik, semmi sem fog történni. Hogy miért? Hát, nem mondtuk meg a gombnak, hogy mit is kellene csinálnia. Ezért kell egy eseményt beregisztrálni, ami akkor aktiválódik, amikor a felhasználó a gombra kattint. Klikkeljünk a frame jobb-felső sarkában lévő X-re, hogy befejezzük a frame futtatását. Ezután térjünk vissza designer nézetbe, válasszuk ki a gombot és menjünk az „Evts” fülre az inspector frame-n. Kattintsunk a ButtonEventre, majd kétszer a felbukkanó wx.EVT_BUTTON szövegre. Vegyük észre, hogy az alsó ablakban lesz egy „OnBtnShowDialogButton” nevű gomb eseményünk. Postoljunk és mentsünk.



Mielőtt továbblépnénk, lássuk, hogy mit is csináltunk kód szinten (23. oldal).

Az első sor egy olyan komment, ami megmondja a Boa Constructornak, hogy ez egy boa fájl. A Python fordító ezt figyelmen kívül fogja hagyni, de a Boa nem. A következő sor beimportájla a wxPythont. Most ugorjunk le az osztálydefinícióhoz.

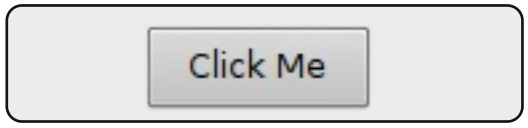
Legfelül van az __init_ctrl metódus. Láthatjuk, hogy van egy komment közvetlenül a leírás sora alatt. Ne szerkesszük az ebben a részben lévő kódot. Ha ezt tennénk, akkor később megbánnánk. De bárhol ALATTA már biztonságos. Ebben a szubrutinban található a frame-ünk összes vezérlőelemének leírása.

Most nézzük meg az __init__ metódust. Ide tetszőleges inicializációs kódot helyezhetünk el. Végezetül az OnBtnShowDialogButton függvényt vizsgáljuk meg. Ide fogjuk beírni azt a kódot, ami akkor hajtódik végre, amikor a felhasználó a gombra kattint. Láthatjuk, hogy egyelőre ez egy event.Skip() sort tartalmaz. Egyszerűen megfogalmazva, ezzel a paranccsal lépünk ki, amikor az esemény aktiválódik.

Most pedig létre fogunk hozni egy felugró ablakot valamilyen szöveggel. Ez egy elég gyakori módja annak, hogy a programozó valamilyen információt közöljön a felhasználóval - akár egy hibaüzenetet, vagy a folyamat befejeződésének tényét. Mi a wx.MessageBox beépített függvényt fogjuk meghívni. Ennek a függvénynek két paramétere van. Az első az ablak által közlendő üzenet, a második az ablak címe. Kommenteljük ki az event.Skip()-et és helyezzük el a következő sort:

```
wx.MessageBox('You Clicked
the button', 'Info')
```

Mentsünk és kattintsunk a futtatás gombra (sárga nyíl). Valami ilyesmit kellene látnunk:



A kattintás után pedig:

Jó ha tudjuk, hogy ez a legpuritánabb módja egy felbukkanó ablak létrehozásának. Még néhány egyéb paramétert is megadhatnánk neki.

Itt van egy rövidke összefoglaló arról, hogy hogyan lehet megváltoztatni az ablak ikonjának kinézetét (többről majd a következő alkalommal lesz szó).

wx.ICON_QUESTION - egy kérdőjel ikon mutatása

wx.ICON_EXCLAMATION - figyelmeztetés ikon kirakása

wx.ICON_ERROR - hiba ikon mutatása

wx.ICON_INFORMATION - info ikon kirajzolása

Ekkor a függvényhívást így fogalmazhatnánk meg:

```
wx.MessageBox('You Clicked
the button', 'Info',
wx.ICON_INFORMATION)
```

```
#Boa:Frame:Frame1
import wx
def create(parent):
    return Frame1(parent)
[wxID_FRAME1, wxID_FRAME1BTNSHOWDIALOG, wxID_FRAME1PANEL1,
] = [wx.NewId() for _init_ctrls in range(3)]

class Frame1(wx.Frame):
    def _init_ctrls(self, prnt):
        # generated method, don't edit
        wx.Frame.__init__(self, id=wxID_FRAME1, name='', parent=prnt,
              pos=wx.Point(543, 330), size=wx.Size(458, 253),
              style=wx.DEFAULT_FRAME_STYLE, title=u'Our First GUI')
        self.SetClientSize(wx.Size(458, 253))
        self.panel1 = wx.Panel(id=wxID_FRAME1PANEL1, name='panel1', parent=self,
              pos=wx.Point(0, 0), size=wx.Size(458, 253),
              style=wx.TAB_TRAVERSAL)
        self.btnShowDialog = wx.Button(id=wxID_FRAME1BTNSHOWDIALOG,
              label=u'Click Me', name=u'btnShowDialog', parent=self.panel1,
              pos=wx.Point(185, 99), size=wx.Size(85, 32), style=0)
        self.btnShowDialog.Bind(wx.EVT_BUTTON, self.OnBtnShowDialogButton,
              id=wxID_FRAME1BTNSHOWDIALOG)

    def __init__(self, parent):
        self._init_ctrls(parent)
    def OnBtnShowDialogButton(self, event):
        event.Skip()
```

vagy valami hasonló módon, attól függően, hogy milyen ikont akarnánk megjeleníteni. Vannak még továbbá különféle gombelrendezések, amiről majd a következő alkalommal lesz szó.

Addig is játszadozzatok el a különböző vezérlőelemekkel és azok elhelyezésével. Jó szórakozást!

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

Programozzunk Pythonban - 6. rész

Írta: Greg Walters

ELŐZŐ SZÁMOK:
FCM 27-31. szám
Python 1-5. rész

ITT HASZNÁLHATÓ:
ubuntu kubuntu xubuntu

KATEGÓRIÁK:
Fejlesztés Grafika Internet M/média Rendszer

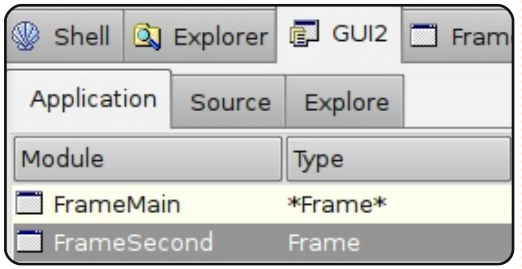
ESZKÖZÖK:
CD/DVD Merevlemez USB Eszköz Laptop Vezeték nélküli

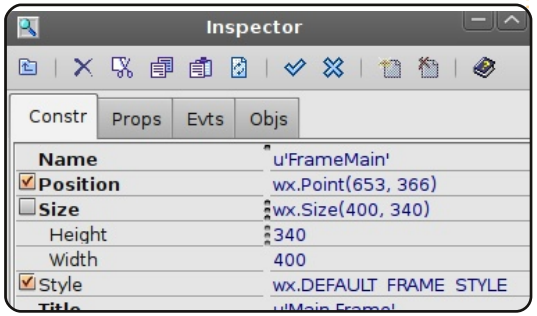
Remélem, hogy legutóbbi találkozásunk óta eljátszadoztatok a Boa Constructorral. Először egy igen egyszerű programmal fogunk foglalkozni, ami egy frame-et rajzol ki, melyen egy gomb megnyomásával meg lehet jeleníteni egy másikat. Előző alkalommal ezt egy üzenőablakkal csináltuk. Most tejesen különálló frame-et készítünk. Ez akkor hasznos, amikor olyan alkalmazást készítünk, ami több frame-mel és ablakkal dolgozik. Csapjunk is a lovak közé!

Indítsuk el a Boa Constructort és az Editor ablakban a Shell és az Explorer kivételével zárjunk be minden fület a (Ctrl-W) billenytűkombinációt használva. Ezzel biztosítjuk, hogy tiszta lappal kezdjünk. Most hozzunk létre egy új projektet a wx.App gombra való kattintással (lásd az előző cikket, ha szükséges).

Mielőtt bármi mást csinálnánk, mentsük el a Frame1-et „FrameMain.py”-ként , illetve az App1-et „Gui2.py”-ként. Ez nagyon fontos. Mialatt a GUI2 fül van kijelölve az Editorban, menjünk vissza a Toolbar ablak New fülére, és adjunk még egy frame-et a projectünkhöz a wx.Frame-re való kattintással (közvetlenül a wx.App gomb mellett). Bizonyosodjunk meg arról, hogy az Application fül Module oszlopában mindkét frame látszik. Most ugorjunk vissza az új frame-re és mentsük el „FrameSecond.py”-ként.

Ezután nyissuk meg a FrameMaint a tervezőben. Adjunk egy wx.Panelt a frame-hez. Méretezzük úgy, hogy lefedje a frame-et. Ezután megváltoztatunk néhány beállítást – ezzel nem foglalkoztunk múltkor. Az inspector ablakban ellenőrizzük, hogy a Constr fül van kiválasztva és állítsuk a címet (title) „Main Frame”-re, majd a nevet (name) „FrameMain”-re. Hamarosan az elnevezési konvenciókra is ki fogunk térni. A Size jelölődobozra való kattintással állítsuk a méretet (size) 400x340-re. Ez le fog nyílni, hogy megmutassa a magasság és szélesség értékeket. A magasságnak 400-nak, a szélességnek 340-nek kell lennie:



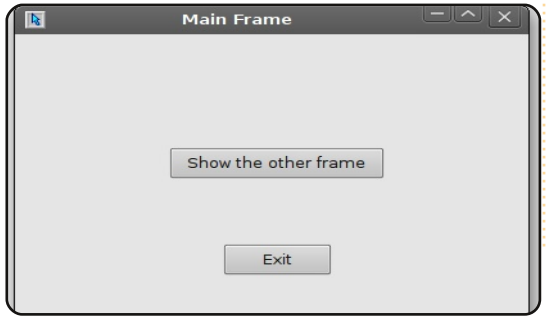


Most kattintsunk a Props fülre. Klikkeljünk a Centered (középre igazított) tulajdonságra, és állítsuk azt wx.BOTH-ra. Nyomjuk meg a post pipa-jelét, majd mentsük munkánkat. Most futtassuk az alkalmazásunkat a sárga nyílra való kattintással. A képernyőnk közepén megjelenik a „Main Frame” nevű programunk. Most zárjuk be az alkalmazást a jobb-felső sarkában lévő „X” gombra való kattintással.

Hozzuk vissza a tervező nézetbe a FrameMaint-t. Helyezzünk el két wx.Buttont, egyiket a másik fölé rakva valahol a frame közepén. Válasszuk ki a felső gombot, nevezzük ezt „btnShowNew”-nak és állítsuk a címkéjét (label) az Inspector ablak Constr fülén „Show the other frame”-re. Méretezzük át a gombot a Shift+Nyilak billentyűkombinációval úgy, hogy az egész szöveg látszódjon, majd használjuk a Ctrl+Nyilak kombinációt ahhoz, hogy visszavigyük a frame közepére. Válasszuk ki az alsó gombot és ne-

vezzük el „btnExit”-nek, a címkéje pedig legyen „Exit”. Postoljuk, mentsünk és futtasuk a programot a változtatások kipróbálásához. Lépjünk ki és menjünk vissza a tervező nézetbe. Most a gombnyomáskor aktiválódó eseményeket fogjuk beállítani. Válasszuk ki a felső gombot, illetve az inspector ablakban az Evts fület. Kattintsunk a ButtonEventre, majd kétszer a wx.Evt_BUTTON-ra. Vegyük észre, hogy az „OnBtnShowNewButton” lent feltűnik. Utána válasszuk ki a btnExit gombot. Csináljuk ugyanazt, mint előbb és bizonyosodjunk meg róla, hogy az „OnBtnExitButton”-t mutat. Postoljuk és mentsünk. Most menjünk az Editor ablakba és görgessunk le az aljáig.

Ellenőrizzük, hogy mindkét létrehozott eseménymetódus megvan-e. Eddig valahogy így kellene kinéznie az ablaknak:



Itt az idő, hogy foglalkozzunk a másik frame-ünkkel is. Nyissuk meg a „FrameSecond”-öt tervező nézetben. Állítsuk a nevét „FrameSecond”-re, a címét meg „Second Frame”-re. Mindkettőnél állítsuk a centeringet wx.BOTH-ra. Helyezzünk el egy wx.Buttont valahova a frame alsó felébe. A neve legyen „btnFSExit” és a címe „Exit”. Állítsuk be a gomb eseményét is. Következőnek tegyünk még egy wx.StaticTextet a frame felső-közép részére. Nevezzük ezt „stHiThere”-re, a címet meg „Hi there... I'm the second form!”-ra, a betűtípust tegyük 14 pont magas wx.BOLD Sans-ra. Most állítsuk vissza a pozícióját úgy, hogy jobbról és balról középre legyen igazítva. Ezt a Position (pozíció) tulajdonság kikapcsolásával tudjuk megtenni, használjuk az X pozíciót a jobb és bal, az Y-t a fent és lent értékek megadásához, amíg kielégítő eredményt nem érünk el. Postoljunk és mentsünk.

Mivel az ablakaink készen vannak, már csak össze kellene valahogy „ragasztani” őket.

Az Editor ablaban kattintsunk a GUI2 fülre, majd ez alatt a Source fülre. Az „import FrameMain” sor alá helyezzük el az „import FrameSecond” parancsot. Mentsük el a változásokat. Ezután válasszuk ki a „FrameMain” fület. Az „import wx” sor után helyezzük el megint az „import FrameSecond” utasítást. Most görgessünk le egészen addig, amíg meg nem látjuk a „def __init__(self,parent):” tartalmú sort. Helyezzünk el a „self.__init__ctrls(parent)” sor alá a „self.Fs = FrameSecond.FrameSecond(self)” sort. A „def OnBtnShowNewButton(self, event):” eseménynél kommenteljük ki az „event.Skip()” hívást, és írjuk be az alábbi két sort:

```
self.Fs.Show()
self.Hide()
```

Végezetül, az „OnBtnExitButton” metódusnál is kommenteljük ki az „event.Skip()”-et, és helyezzük el a „self.Close()”-t.

De mire volt jó mindez? Rendben. Amit először csináltunk, az az volt, hogy megismertettük az alkalmazásunkkal az általunk használni kívánt ablakokat. Ezért importáltuk be mind a FrameMaint és a FrameSecondöt a GUI2 fájlba. Ezután a FrameSecond referenciáját a FrameMainbe is beimportáltuk, mivel majd később itt akarunk hivatkozni rá. Inicializáltuk az „_init_” metódust. Az „OnBtn ShowNewButton” eseménynek megadtuk, hogy amikor a gombot megnyomjuk, először meg szeretnénk jeleníteni a második ablakot, majd eltüntetni az elsőt. Legutoljára az alkalmazást bezáró parancsot helyeztük el az Exit gomb megnyomásához.

Most váltsunk át a FrameSecond kódjára. A változtatások itt viszonylag kicsik voltak. Az „_init_” metódusba beírt „self.parent = parent” sor egy self.parent változót hozott létre. Végezetül, az FSExitButton eseménynél is kommenteljük ki az „event.Skip()” sort, és helyezzük el az alábbi két sort:

```
self.parent.Show()
self.Hide()
```

Emlékezzünk arra, hogy mi-

vel elrejtettük az első ablakot, amikor megjelenítettük a másodikat, ezért azt újra meg kell jeleníteni. Végül a másodikat is el kell rejtenünk. Mentsük el a változásokat.

Itt van ellenőrzésképpen a teljes kód (ez az oldal és az utána lévő):

Most már futtathatjuk az alkalmazásunkat. Ha minden rendben ment, akkor rá tudunk kattintani a btnShowNew-ra és láthatjuk, ahogy az első ablak eltűnik és a második megjelenik. Az Exit gombra való kattintás eltünteti az ablakot és újra megjeleníti az elsőt. Ha a fő ablakban kattintunk az Exit

GUI2 code:

```
#!/usr/bin/env python
#Boa:App:BoaApp

import wx

import FrameMain
import FrameSecond

modules ={u'FrameMain': [1, 'Main frame of Application',
u'FrameMain.py'],
 u'FrameSecond': [0, '', u'FrameSecond.py']}

class BoaApp(wx.App):
    def OnInit(self):
        self.main = FrameMain.create(None)
        self.main.Show()
        self.SetTopWindow(self.main)
        return True

def main():
    application = BoaApp(0)
    application.MainLoop()

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

FrameMain code:

```
#Boa:Frame:FrameMain

import wx
import FrameSecond

def create(parent):
    return FrameMain(parent)

[wxID_FRAMEMAIN, wxID_FRAMEMAINBTNEXIT,
wxID_FRAMEMAINBTNSHOWNEW,
 wxID_FRAMEMAINPANEL1,
] = [wx.NewId() for _init_ctrls in range(4)]

class FrameMain(wx.Frame):
    def _init_ctrls(self, prnt):
        # generated method, don't edit
        wx.Frame.__init__(self, id=wxID_FRAMEMAIN,
name=u'FrameMain',
              parent=prnt, pos=wx.Point(846, 177),
size=wx.Size(400, 340),
              style=wx.DEFAULT_FRAME_STYLE, title=u'Main
Frame')
        self.SetClientSize(wx.Size(400, 340))
        self.Center(wx.BOTH)

        self.panel1 = wx.Panel(id=wxID_FRAMEMAINPANEL1,
name='panel1',
              parent=self, pos=wx.Point(0, 0),
size=wx.Size(400, 340),
              style=wx.TAB_TRAVERSAL)

        self.btnShowNew =
wx.Button(id=wxID_FRAMEMAINBTNSHOWNEW,
              label=u'Show the other frame',
name=u'btnShowNew',
              parent=self.panel1, pos=wx.Point(120, 103),
size=wx.Size(168, 29),
              style=0)
        self.btnShowNew.SetBackgroundColour(wx.Colour(25,
175, 23))
        self.btnShowNew.Bind(wx.EVT_BUTTON,
self.OnBtnShowNewButton,
              id=wxID_FRAMEMAINBTNSHOWNEW)
```

FrameMain Code (cont.):

```
        self.btnExit =
wx.Button(id=wxID_FRAMEMAINBTNEXIT, label=u'Exit',
              name=u'btnExit', parent=self.panel1,
pos=wx.Point(162, 191),
              size=wx.Size(85, 29), style=0)
        self.btnExit.SetBackgroundColour(wx.Colour(225,
218, 91))
        self.btnExit.Bind(wx.EVT_BUTTON,
self.OnBtnExitButton,
              id=wxID_FRAMEMAINBTNEXIT)

    def __init__(self, parent):
        self._init_ctrls(parent)
        self.Fs = FrameSecond.FrameSecond(self)

    def OnBtnShowNewButton(self, event):
        #event.Skip()
        self.Fs.Show()
        self.Hide()

    def OnBtnExitButton(self, event):
        #event.Skip()
        self.Close()
```

FrameSecond code:

```
#Boa:Frame:FrameSecond

import wx

def create(parent):
    return FrameSecond(parent)

[wxID_FRAMESECOND, wxID_FRAMESECONDBTNFSEXIT,
wxID_FRAMESECONDPANEL1,
 wxID_FRAMESECONDSTATICTEXT1,
] = [wx.NewId() for _init_ctrls in range(4)]

class FrameSecond(wx.Frame):
    def _init_ctrls(self, prnt):
        # generated method, don't edit
        wx.Frame.__init__(self, id=wxID_FRAMESECOND,
name=u'FrameSecond',
              parent=prnt, pos=wx.Point(849, 457),
size=wx.Size(419, 236),
              style=wx.DEFAULT_FRAME_STYLE, title=u'Second
Frame')
        self.SetClientSize(wx.Size(419, 236))
        self.Center(wx.BOTH)
        self.SetBackgroundStyle(wx.BG_STYLE_COLOUR)

        self.panel1 = wx.Panel(id=wxID_FRAMESECONDPANEL1,
name='panel1',
              parent=self, pos=wx.Point(0, 0),
size=wx.Size(419, 236),
              style=wx.TAB_TRAVERSAL)

        self.btnFSExit =
wx.Button(id=wxID_FRAMESECONDBTNFSEXIT, label=u'Exit',
              name=u'btnFSExit', parent=self.panel1,
pos=wx.Point(174, 180),
              size=wx.Size(85, 29), style=0)
        self.btnFSExit.Bind(wx.EVT_BUTTON,
self.OnBtnFSExitButton,
              id=wxID_FRAMESECONDBTNFSEXIT)

        self.staticText1 =
wx.StaticText(id=wxID_FRAMESECONDSTATICTEXT1,
              label=u"Hi there...I'm the second form!",
name='staticText1',
              parent=self.panel1, pos=wx.Point(45, 49),
size=wx.Size(336, 23),
              style=0)
        self.staticText1.SetFont(wx.Font(14, wx.SWISS,
wx.NORMAL, wx.BOLD,
              False, u'Sans'))

    def __init__(self, parent):
        self._init_ctrls(parent)
        self.parent = parent

    def OnBtnFSExitButton(self, event):
        #event.Skip()
        self.parent.Show()
        self.Hide()
```

gombra, akkor bezárjuk az alkalmazást.

Megígértem, hogy megtárgyaljuk az elnevezési konvenciókat. Emlékszünk még, hogy valamikor régen beszéltem a kód kommentelésről? Nos, jól megfogalmazott GUI vezérlőelemnevek használatával a kódunk éppen eléggé öndokumentáló lesz.

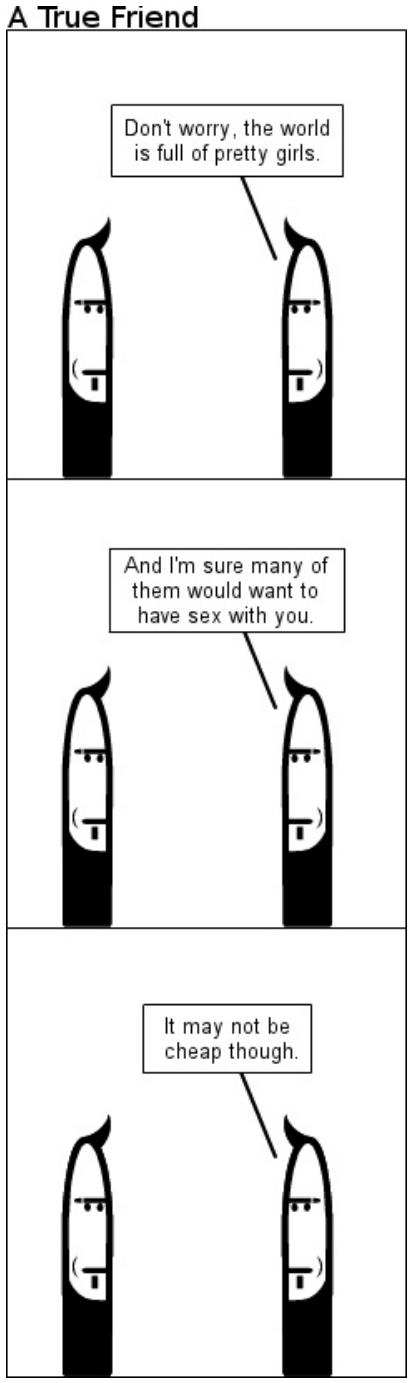
Ha staticText1, button1 vagy hasonló neveket használtunk volna, akkor egy összetettebb ablak létrehozásakor - mely rengeteg vezérlőelemet tartalmaz, kiváltképpen sok szövegdobozt és gombot - sok problémánk adódhatott volna. Ilyen esetben a megfelelő jelentésű elnevezés igen fontos. Talán nem annyira lényeges, ha te vagy az egyetlen, aki valaha is látni fogja a kódot, de ha esetleg valaki majd átveszi, akkor a jó elnevezés sokat fogja segíteni a munkáját. Éppen ezért, használjunk valami ilyesmit:

```
Control type - névprefix
Statikus szöveg - st_
Gomb - btn_
Szövegdoboz - txt_
Jelölődoboz - chk_
Rádió gomb - rb_
Frame - Frm_ vagy Frame_
```

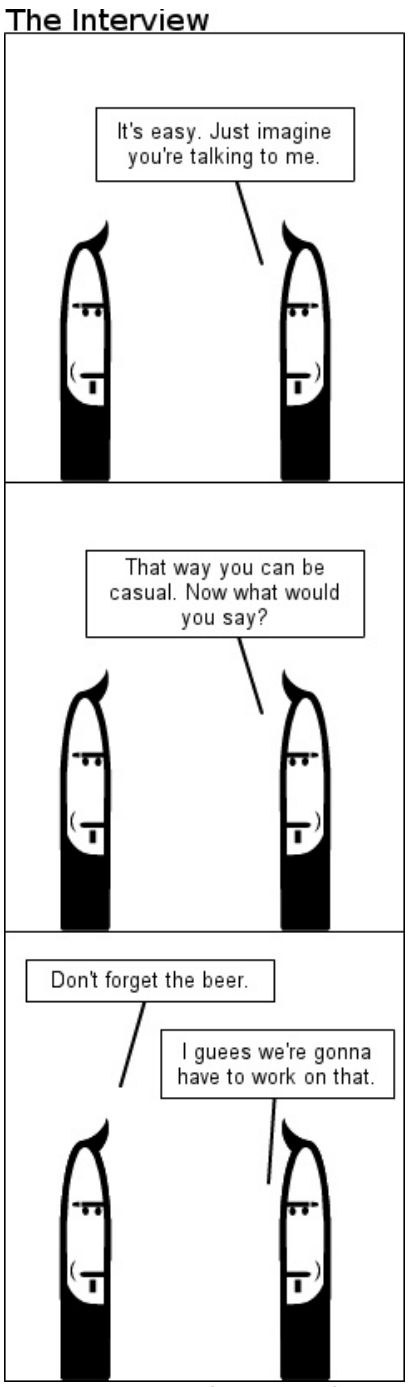
Ahogy fejlődünk a programozás terén, kitalálhatjuk a saját elnevezési szabályainkat is, és néhány esetben a munkahelyeden meg is szabják azokat helyetted.

Következő alkalommal félretesszük a GUI programozást egy kicsit, és az adatbázisokra fogunk koncentrálni. Addig is, tegyétek fel a python-apsw-t és a python-mysqldb-t a rendszeretekre. Továbbá szükségünk lesz az SQLite-unkhoz az sqlitera és az sqlitebrowser-re. Ha szeretnétek a MySql-lel is kísérletezni, akkor az is elérhető a Synaptic-ból.

Greg Walters a *RainyDay Solutions* tulajdonosa, ez egy korlátolt felelősségű tanácsadó cég a coloradoi Aurorában. Programozással 1972 óta foglalkozik. Szeret főzni, túrázni, zenét hallgatni és szabadidejét családja körében eltölteni.



Rédei Richárd



Rédei Richárd

Programozzunk Pythonban - 7. rész

Írta: Greg Walters

ELŐZŐ SZÁMOK:
FCM 27-32. szám
Python - 1-6. rész

ITT HASZNÁLHATÓ:
ubuntu kubuntu xubuntu

KATEGÓRIÁK:
Fejlesztés Grafika Internet M/media Rendszer

ESZKÖZÖK:
CD/DVD Merevlemez USB Eszköz Laptop Vezeték nélküli

Jó reggelt fiúk-lányok! Mese idő van. Mindenki helyezze kényelembe magát. Készen álltok? Rendben!

Egyszer régen a világot a papír uralta. Mindenhol papír és papír. Még saját otthonokat is készíteni kellett a papíroknak. Ezeket hívtuk kartotékszekrényeknek, melyek nagy, termeket, termeket és termeket felölelő fém dolgok voltak az üzleti életben a papírok eltárolásához. Minden kartotékszekrényben volt egy mappa, amivel a fontos papírokat próbálták összefogni. Egy idő után túltöltődtek és szétestek, mivel már nagyon régiek voltak, vagy sokszor nyitották ki őket.

Ezeknek a kartotékszekrényeknek a használatához főiskolai végzettség kellett. Napok kellettek ahhoz, hogy megtaláljunk minden papírt a különböző szekrényekben. Az üzlet iszonyúan meg is szenvedte. Ez egy nagyon sötét időszaka volt az emberiségnek.

Ekkor egy nap, valahonnan a hegy tetejéről (személyszerint úgy gondolom, hogy Colorádóból, de ebben nem vagyok biztos) előjött egy bájos tündér. Ez a tündér kék és ezüst színű volt - 35 centiméter magas, csodaszép szárnyakkal és fehér hajjal. A neve, hiszed vagy sem, Szíkvill volt. Hát nem egy vicces kis név? Mindenesetre, Szíkvill azt ígérte, hogy megold minden problémát, amit az összes papír, kartoték és elpocsékolt idő okozott, azzal az egy feltétellel, ha az emberek bíznak majd a számítógépekben és benne. Ezt a hatalmat Ő „Adatbázisoknak” nevezte. Azt mondta, hogy az „Adatbázis” az egész kartotékrendszert le tudja váltani. Néhányan hittek benne és hamarosan boldog életük lett. Azonban voltak olyanok akik nem és számukra minden nap ugyanolyan papírhegyekben kutakodós maradt.

Azonban minden tündérígéret valamilyen feltétellel jár. Ez a feltétel az volt, hogy mindenki, aki Szíkvill bűbáját akarta használni, meg kellett tanulnjon egy kicsit egy másik nyelvből. Nem lenne túl nehezen tanulható nyelv. Ami azt illeti, nagyon hasonló volt ahhoz, amit az emberek eddig is használtak. Egyszerűen csak máshogy lehet rajta kifejezni magunkat, és igencsak el kellett gondolkodni a dolgokon, MIELŐTT kimondtad őket ahhoz, hogy Szíkvill varázsa hasson.

Egy nap egy fiatal fiú, akit érdekes módon Felhasználónak hívtak, eljött, hogy csodájára járjon Szíkvillnek. Teljesen elbűvölte szépsége és azt mondta: „Szíkvill, kérlek taníts meg az erőd használatára”. Mire Szíkvill megígérte, hogy megteszi. Szíkvill ezután így szólt: „Először ismerned kell az információid szerkezetét. Mutasd meg a papírjaidat.”

Mivel Felhasználó még igen fiatal volt, ezért csak néhány darab papírt birtokolt. Szíkvill erre azt mondta: „Felhasználó, jelenleg még el tudsz éldegélni papírokkal és kartotékokkal, de látom, hogy a jövőben olyan sok papírod lesz, hogy ha egymás tetejére raknánk őket, akkor 15-ször magasabb lenne, mint te vagy. Használjuk hát a varázserőmet.”

Így történt hát, hogy Felhasználó és Szíkvill megalkotta az „adatbázis bigyót” (ami egy tündér szakkifejezés), és Felhasználó boldogan élt, amíg meg nem halt.

Itt a vége, fuss el véle.

Természetesen a történet teljes egészében nem igaz. Mindazonáltal az adatbázisok és az SQL használata könnyebbé teheti az életünket. Ez alkalommal egy kis SQL lekérdezésről tanulunk, illetve arról, hogy hogyan használjuk őket a programokban. Néhány ember úgy gondolhatná, hogy ez nem teljesen a „legkorrektebb” vagy „legjobb” módszer, de ennek ellenére nagyon is elfogatható. Kezdjük hát el.

Az adatbázisok olyasmik, mint a kartotékszekrényeink a fenti történetben. Az adattáblák olyanok, mint a mappák. Az gyes bejegyzések (avagy rekordok) pedig hasonlóak a papírlapokhoz. Minden egyes információdarabot mezőnek nevezünk. Igen egybevágóak, igaz? Az SQL (amit Szíkvillnek ejtünk) utasításokat használjuk az adatok manipulálására. Az SQL maga a Structured Query Language (struktúrált lekérdező nyelv) rövidítése és alapjaiban véve az adatbázisok egyszerű használatára találták ki. A gyakolatban azonban igen bonyolulttá válhat. Ebben a részben csak egyszerű dolgokkal fogunk foglalkozni.

Először is szügségünk van egy tervre, ugyanúgy, mint egy építkezés kezdetén. Gondoljunk most receptkártyákra, melyen azért jó elmélkedni, mivel egy receptadatbázis-programot fogunk készíteni. Az én házam táján a receptek különböző formában vannak jelen: 3x5-ös kártya, 8x10-es papírdarab, szalvétán lévő irka-firka, magazinból való lapok, és még ennél is különösebb dolgok. Mindezek könyvekben, dobozokban, kötegekben és hasonló helyeken találhatóak meg. Ennek ellenére legtöbbször van egy közös tulajdonságuk: a formátumuk. A legtöbb esetben a felső részen van a recept neve és talán az, hogy hány személyre való, illetve, hogy honnan származik. A középső rész tartalmazza a hozzávalók listáját, a legalsó pedig az elkészítéséhez tartozó utasításokat - az egyes műveletek sorrendje, az elkészítés ideje és így tovább. Ezt az általános formátumot fogjuk sablonként használni az adatbázis projektünkben. Először is, két részre fogjuk osztani a dolgokat. Az adatbázist most fogjuk elkészíteni, az alkalmazást, ami olvasni és frissíteni fogja, a következő alkalommal.

Itt egy példa. Tegyük fel, hogy a jobbra lévő receptünk van.

Vegyük észre a sorrendet, amiről az előbb beszéltünk. Most mikor az adatbázisunkat tervezzük, nagyon nagyot is készíthetnénk és lehetne egy külön rekordja minden receptnek. Ez azonban kimondottan esetlen és nehezen kezelhető lenne. Éppen ezért, a receptkártyákat fogjuk sablonként használni. Egy tábla a kártya felső részét fogja kezelni, azaz az általános információkat a receptről, egy másik tábla a kártya közepét fogja tárolni, avagy a hozzávalókat, és egy megint másik az alsó rész, ami az instrukciónak felel meg.

Bizonyosodjunk meg róla, hogy az SQLite és az APSW fel vannak telepítve gépünkre. Az SQLite egy apró adatbáziskezelő motor, mely nem követeli meg, hogy egy külön adatbázisszerverünk legyen, éppen ez teszi ideálissá az alkalmazásunk számára.

Spanyol rizs

Adagok: 4

Forrás: Greg Walters

Hozzávalók:
1 bögre opál rizs
1 font hamburgerhús
2 bögre víz
1 8 unciás sűrített paradicsom
1 kicsi darabolt hagyma
1 gerezd darabolt fokhagyma
1 kanál őrölt kömény
1 kiskanál őrölt oregano
só és paprika ízlés szerint
Salsa ízlés szerint

Instrukciók:
Süssük barnára a hamburgerhúst.

Addjuk hozzá az összes alapanyagot.

Forraljuk fel.

Keverjük, kis lángon főzzük és fedjük le.
Húsz percig főzzük.

Ne nézegessük és ne nyúljunk hozzá.

Kavarjuk meg és szolgáljuk fel.

Minden, amit itt tanulsz, felhasználható nagyobb adatbázisrendszereknél is, mint amilyen a MySQL és a többiek. A másik jó dolog az SQLite-ban, hogy előre meghatározott adattípusokat használ. Ezek a Text (szöveg), Numeric (szám), Blob és Integer Primary Key (egész értékű elsődleges kulcs). Mint ahogy azt már korábban megtanultuk, szöveg gyakorlatilag bármi lehet. A hozzávalóink, az instrukcióink és a receptünk címe mind szöveg típusú - még annak ellenére is, hogy számokat tartalmazhatnak. A numerikus adattípusok számokat tárolnak. Ezek lehetnek egészek, lebegőpontosak vagy valós értékűek. A Blob-ok bináris adatok, melyekben képeket és hasonló dolgokat lehet eltárolni. Az egészértékű elsődleges kulcs különleges. Az SQLite adatbázismotor mindig automatikusan egy egyedi egész értéket generál számunkra. Ez a későbbiek folyamán fontos lesz. Az APSW az Another Python SQLite Wrapper-ből (Másik Python SQLite Csomag) ered, egy gyors módszere az SQLite-tal való kommunikációnak. Most fussuk át az SQL utasítások létrehozásának néhány lehetőségét.

Ahhoz, hogy rekordokat tudjunk kiolvasni az adatbázisból, a SELECT utasítást kell használnunk. A szerkezete így néz ki:

```
SELECT [mit] FROM [melyik táblá(k)ból] WHERE [megszorítások]
```

Tehát, ha minden mezőt ki szeretnénk olvasni a Recipes (receptek) táblából, akkor a következőt írnánk:

```
SELECT * FROM Recipes
```

Ha csak az elsődleges kulcs alapján akarunk egy rekordot kiolvasni, akkor tudnunk kell mi az értéke (ebben az esetben a pkID értékéről van szó) és a WHERE parancsot is bele kell rakni az utasításba. Használjuk ezt:

```
SELECT * FROM REcipes WHERE pkID = 2
```

Egyszerű... igaz? Igen világos egy nyelv. Tegyük fel, hogy most csak a recept nevére (name), illetve az adagok számára (serving) van szügségünk, minden receptből. Könnyű. Mindössze csak annyit kell csinálnunk, hogy megadjuk azokat a mezőket, amiket a SELECT utasítással le akarunk kérdezni:

```
SELECT name, servings FROM Recipes
```

Ahhoz, hogy rekordokat szúrjunk be, az INSERT INTO utasítást használjuk. Szintaxisa:

```
INSERT INTO [tábla neve] (mezőlista) VALUES (beszúrandó értékek)
```

Tehát ahhoz, hogy egy receptet adjunk hozzá a recept táblához, az alábbi parancsot használhatnánk:

```
INSERT INTO Recipes (name,servings,source) VALUES ("Tacos",4,"Greg")
```

Rekordok törléséhez ezt használhatjuk:

```
DELETE FROM Recipes WHERE pkID = 10
```

Van még egy UPDATE utasítás is, de ezt egy másik alkalomra hagyom.

Még több a SELECT-ről

Adatbázisunknak három táblája lesz, mindegyik a recipeID használatával kapcsolódik egymáshoz, mely a recipe tábla pkID-jére mutat. Tegyük fel, hogy meg szeretnénk szerezni egy adott recept összes instrukcióját. Ezt így tehetjük meg:

```
SELECT Recipes.name, Recipes.servings, Recipes.source, Instructions.Instructions FROM Recipes LEFT JOIN instructions ON (Recipes.pkid = Instructions.recipeid) WHERE Recipes.pkid = 1
```

De ez nagyon sok gépeléssel és redundanciával jár. Ehelyett használhatjuk a helyettesítésnek nevezett technikát. Ezt így csinálhatjuk:

```
SELECT r.name, r.servings, r.source, i.Instructions FROM Recipes r LEFT JOIN instructions i ON (r.pkid = i.recipeid) WHERE r.pkid = 1
```

Így rövidebb és olvashatóbb lett a lekérdezés. Most egy olyan aprócska programot fogunk készíteni, ami létrehozza az adatbázisunkat, megcsinálja a tábláinkat, és néhány egyszerű adattal feltölti azokat, hogy legyen mivel dolgoznunk. Beleírhatnánk a teljes programunkba, de - a példa kedvéért - egy különálló alkalmazást fogunk készíteni. Ez egy egyetlen futásra

tervezett program - ha másodjára is elindítanánk, akkor a táblalétrehozó utasításnál hibába ütközne. Befoglalhatnánk egy try...catch blokkba is, de ezt majd a következő alkalommal tesszük.

Először beimportáljuk az APSW-t.

```
import apsw
```

Majd létre kell hoznunk egy kapcsolatot az adatbázisunkkal. Ugyanabban a könyvtárban lesz elérhető, amiben a programunk is van. Amikor létrehozzuk a kapcsolatot, az SQLite automatikusan meg fogja nézni, hogy létezik-e az adatbázis. Ha létezik, megnyitja. Ha nem, akkor pedig létrehozza azt. Amikor van már egy kapcsolatunk, szükségünk lesz egy cursornak (mutató) nevezett dologra. Ez egy olyan konstrukció, amit az adatbázissal való munkához használhatunk. Amit meg kell jegyezni, az az, hogy egy kapcsolatra és egy cursorra van szügségünk. Ezeket így hozzuk létre:

```
# Opening/creating database
```

```
RECIPES
------------
pkID (Integer Primary Key)
name (Text)
source (Text)
serves (Text)
```

```
INSTRUCTIONS
----------------------
pkID(Integer Primary Key)
recipeID (Integer)
instructions (Text)
```

```
INGREDIENTS
--------------------
pkID (Integer Primary Key)
recipeID (Integer)
ingredients (Text)
```

```
connection=apsw.Connection("cookbook1.db3")
cursor=connection.cursor()
```

Rendben - megvan a kapcsolatunk és a cursorunk. Most egy táblát kellene létrehozni. Összesen három tábla lesz az alkalmazásban. Egy az általános receptinformációknak, egy az instrukcióknak és egy a hozzávalók listájának. Azonban miért nem használunk mindenhez egyetlen táblát? Hát, lehetne, de ahogy látni fogjuk, így túl nagy lenne és sok azonos információt tartalmazna.

A tábla szerkezetét így is el tudnánk képzelni. Minden oszlop egy külön tábla, mint ahogy azt a fent látható ábra is mutatja.

Minden táblának van egy pkID nevű mezője. Ez lesz az elsődleges kulcs, ami minden táblában egyedi. Ez fontos, mivel így az adattáblákban soha nem lesznek teljesen ugyanolyan rekordok. Ez egy egész értékű adattípus és automatikusan kiosztja az adatbázisrendszer. De kell ez nekünk? Igen kell, mert megvan annak az esélye, hogy ugyanazt a rekordazonosítót kétszer hozzuk létre. Az Recipes tábla esetében ezt a számot arra fogjuk használni, hogy beazonosítsuk az egyes receptek instrukcióit és hozzávalóit.

Először az adatbázis recept táblájába fogjuk beszúrni az étel nevét, forrását és adagját. A pkID automatikusan ki lesz osztva. A példa kedvéért tegyünk úgy, mintha ez lenne a legelső rekord adatbázisunkban, tehát az adatbáziskezelő az 1 értéket adná a pkID-nek. Ezt az értéket fogjuk használni a többi információ összekapcsolására az adott receptnél. Az instrukciók táblája nagyon egyszerű. Csak az elkészítési utasítások hosszú szövegét fogja tartalmazni, meg a saját pkID-jét, ami a recipe táblában lévő receptre mutat. A hozzávalók tábla egy kicsit bonyolultabb annyiban, hogy minden hozzávalónak egy külön rekordja van a saját pkID-je mellett, ami megintcsak a recipe táblára mutat vissza.

Tehát, a recipe tábla létrehozásához először egy sql azonosítójú string változót kell definiálnunk, amihez a tábla létrehozása utasítást rendeljük hozzá:

```
sql = 'CREATE TABLE Recipes (pkiD INTEGER PRIMARY KEY, name TEXT, servings TEXT, source TEXT)'
```

Ezután meg kell mondani az ASPW-nek, hogy hajtsa végre az utasítást:

```
cursor.execute(sql)
```

Most létrehozzuk a táblákat:

```
sql = 'CREATE TABLE Instruc-
tions (pkID INTEGER PRIMARY
KEY, instructions TEXT, reci-
peID NUMERIC)'

cursor.execute(sql)

sql = 'CREATE TABLE Ingredi-
ents (pkID INTEGER PRIMARY
KEY, ingredients TEXT, reci-
peID NUMERIC)'

cursor.execute(sql)
```

Amint létrejöttek a táblák, az INSERT INTO utasítást fogjuk használni, hogy minden adatot a megfelelő táblába felvigyünk.

Emlékezzünk arra, hogy a pkID automatikusan be lesz írva számunkra, ezért nem veszszük bele az insert utasításba. Mivel mezőneveket fogunk használni, ezért tetszőleges sorrendben lehetnek, nem csak abban a sorrendben, amiben létrehoztuk őket. Mindaddig, amíg tudjuk a mezők neveit, minden helyesen fog működni. A recipe tábla insert utasítása ez:

```
INSERT INTO Recipes (name,
serves, source) VALUES ("Spa-
nish Rice",4,"Greg Walters")
```

Következő lépésként ki kell találnunk a recipe táblához rendelt pkID-t. Ezt egy egyszerű utasítással tehetjük meg:

```
SELECT last_insert_rowid()
```

De ez nem egy olyan dolog, amit csak úgy használhatunk. Egy sor hasonló utasítást kell használnunk, mint itt:

```
sql = "SELECT last_insert_-
rowid()"

cursor.execute(sql)

for x in cursor.execute(sql):
    lastid = x[0]
```

Hogy mért is van ez? Nos, amikor adatot szeretnénk visszaszerezni az ASPW-ből, az tuple-ként (kb. vektor) jön vissza. Erről még nem beszéltünk. A rövid magyarázat az lenne, hogy a tuple (ha megnézzük a fenti kódot) olyan, mint egy lista, csak nem lehet módosítani. Sokan csak ritkán használják a tuple-öket, vannak olyanok is, akik gyakran, ez egyedül rajtad múlik. Az a lényeg, hogy az első visszaadott értéket akarjuk felhasználni. A for ciklust arra használjuk, hogy az értéket eltároljuk az x tuple változóban.

Érthető? Rendben. Folytassuk...

Most, létrehozzuk az instrukciók insert utasítását:

```
sql = 'INSERT INTO Instructi-
ons (recipeID,instructions)
VALUES( %s,"Brown hamburger.
Stir in all other ingredi-
ents. Bring to a boil. Stir.
Lower to simmer. Cover and
cook for 20 minutes or until
all liquid is absorbed.")' %
lastid

cursor.execute(sql)
```

Figyeljük meg, hogy egy változó behelyettesítést (%s) használunk a recept pkID-jének (lastid) elhelyezéséhez az sql utasításban. Végül, minden egyes hozzávalót el kell helyeznünk az ingredient táblában. Egyet meg is mutatok:

```
sql = 'INSERT INTO Ingredi-
ents (recipeID,ingredients)
VALUES ( %s,"1 cup parboiled
Rice (uncooked)")' % lastid

cursor.execute(sql)
```

Ezen a ponton ezt már nem túl nehéz megérteni. Következő alkalommal egy kicsit bonyolódni fognak a dolgok.

Ha el szeretnéd érni a teljes forráskódot, akkor azt a honlapomon megtalálod. Menj a www.thedesignatedgeek.com oldalra és töltsd le.

Legközelebb fel fogjuk használni a recept programunkhoz mindazt, amit a sorozat alatt tanultunk a menüközpontú felületek létrehozásáról - segítségével megnézhetjük a receptek listáját, egyetlen egy receptet, kereshetünk a receptek között és bővithetjük, illetve törölhetjük őket.

Azt javaslom, hogy egy kis időt szánjunk az SQL programozásról szóló dolgok utánolvasására. Jól fogsz járni, ha megteszed.

Greg Walters a RainyDay Solutions tulajdonosa, ez egy korlátolt felelősségű tanácsadó cég a colorado-i Aurorában. Programozással 1972 óta foglalkozik. Szeret főzni, túrázni, zenét hallgatni és szabadidejét családja körében eltölteni.

Programozzunk Pythonban - 8. rész

Írta: Greg Walters

ELŐZŐ SZÁMOK:
FCM 27-33 - Python 1-7. rész

ITT HASZNÁLHATÓ:
ubuntu kubuntu xubuntu

KATEGÓRIÁK:
Fejlesztés Grafika Internet M/média Rendszer

ESZKÖZÖK:
CD/DVD Merevlemez USB eszköz Laptop Vezeték nélküli

Folytassuk a hetedik részben elkezdett recept adatbázisunk programozását. Ez a cikk hosszú lesz, és sok-sok kódot fog tartalmazni. Szóval kössétek fel a gatyákat, és ne felejtsétek a kezeket és lábakat mindvégig a kocsiban tartani. Az adatbázisunkat már létrehoztuk. Most már szeretnénk megjeleníteni a tartalmát, hozzáadni, illetve törölni belőle. De hogyan is fognak ezek működni? Kezdésként egy konzolos alkalmazást fogunk elkészíteni, ehhez pedig egy menüt kell összeraknunk. Mindezen felül az adatbázishoz tartozó eljárásokat tartalmazó osztályt is létre kell hoznunk. Vágjunk is neki a jobbra fent található programrészlettel.

Most pedig alakítsuk ki a menü szerkezetét. Erre azért van szükségünk, mert így el tudjuk helyezni a program szerkezetét meghatározó kódtöredékeket. A menünk egy igen nagy ciklus lesz, ami kiíratja a felhasználó által végrehajtható opciók listáját. Ehhez egy while ciklust fogunk használni. Módosítsuk a menu eljárásunkat úgy, hogy a jobb oldalon, lent lévő kódra hasonlítson.

A menut egy if|elif|else struktúrával töltjük fel (következő oldal tetején látható).

Gyorsan fussuk át a menu metódusunkon. A felhasználó által választható opciók kiíratásával kezdünk. Egy változót (loop) True-ra (igaz) állítunk és a while ciklusban addig iterálunk amíg a loop=False nem lesz. A raw_input() parancsot használva várunk arra, hogy a felhasználó

```
#!/usr/bin/python
#-------------------------------------------------------
# Cookbook.py
# Created for Beginning Programming Using Python #8
# and Full Circle Magazine
#-------------------------------------------------------
import apsw
import string
import webbrowser

class Cookbook:

def Menu():
    cbk = Cookbook() # Initialize the class

Menu()
```

```
def Menu():
    cbk = Cookbook() # Initialize the class
    loop = True
    while loop == True:
        print
'=================================================='
        print '                RECIPE DATABASE'
        print
'=================================================='
        print ' 1 - Show All Recipes'
        print ' 2 - Search for a recipe'
        print ' 3 - Show a Recipe'
        print ' 4 - Delete a recipe'
        print ' 5 - Add a recipe'
        print ' 6 - Print a recipe'
        print ' 0 - Exit'
        print
'=================================================='
        response = raw_input('Enter a selection -> ')
```

```
        if response == '1': # Show all recipes
            pass
        elif response == '2': # Search for a recipe
            pass
        elif response == '3': # Show a single recipe
            pass
        elif response == '4': # Delete Recipe
            pass
        elif response == '5': # Add a recipe
            pass
        elif response == '6': # Print a recipe
            pass
        elif response == '0': # Exit the program
            print 'Goodbye'
            loop = False
        else:
            print 'Unrecognized command.  Try again.'
```

kiválassszon egy menüelemet. Ezt követően az if lekezeli a kiválasztott műveletet. Mielőtt azonban tesztelni tudnánk mindezt, létre kell hoznunk az osztályunk __init__ eljárását:

```
def __init__(self):
    pass
```

A programunkat mentsük ugyanoda, ahol múltkor az adatbázist hoztuk létre és futtassuk. A jobbra fent található képhez hasonlót kellene látnunk.

Egész egyszerűen csak ki kell íratnia a menüt újra és újra, addig, amíg meg nem nyomjuk a „0”-t. Ekkor kiíratjuk a „Goodbye” szöveget és kilépünk. Ezen a ponton már elkezdhetjük a Cookbook osztályban eljárásaink vázának létrehozását. Szükségünk lesz egy olyan függvényre, ami megjeleníti a Recipes tábla információit, egy olyanra amivel kereshetünk a receptek között, egy másikra, ami megjelenít egy teljes receptet a három táblából, egyre ami töröl egy receptet, még egyre, amivel új receptet tudunk felvenni, és végül egy olyanra ami az alapértelmezett nyomtatóval kinyomtatja a receptet. A PrintAllRecipes rutinnak nincs szüksége a (self) paraméteren kívül másra, akárcsak a SearchforRecipe-nek és az EnterNew eljárásnak sem. A PrintSingleRecipe, DeleteRecipe és PrintOut metódusoknak tudniuk kell melyik recepttel van dolguk, tehát szükségük van egy olyan paraméterre, amit most „which”-nek fogunk nevezni. Egyelőre mindegyik kódtörzsben használjuk a pass parancsot. A Cookbook osztály alatt helyezzük el az eljárástörzseket:

```
def PrintAllRecipes(self):
  pass
def SearchForRecipe(self):
  pass
def
PrintSingleRecipe(self,which)
:
  pass
def DeleteRecipe(self,which):
  pass
def EnterNew(self):
  pass
def PrintOut(self,which):
  pass
```

```
/usr/bin/python -u
"/home/greg/python_examples/APSW/cookbook/cookbook_stub.py"
==================================================
                RECIPE DATABASE
==================================================
1 - Show All Recipes
2 - Search for a recipe
3 - Show a Recipe
4 - Delete a recipe
5 - Add a recipe
6 - Print a recipe
0 - Exit
==================================================
Enter a selection ->
```

Néhány menüelemnél ki szeretnénk majd íratni a Recipe táblában lévő összes receptet, így a felhasználó képes lesz választani a listából. Ezek az 1-es, 3-as, 4-es és 6-os opciók lesznek. Módosítsuk hát ezeket a menüelmeket a pass parancs cbk.PrintAllRecipes()-re való cseréjével. A felhasználói visszajelzést ellenőrző rutinunk a következő oldal tetején látható.

Van még egy dolog, amit meg kell tennünk: be kell állítanunk az __init__ metódust. A kódtöredéket helyettesítsük az alábbi sorokkal:

```
def __init__(self):
  global connection
  global cursor
  self.totalcount = 0
  connection=apsw.Connection
("cookbook.db3")
cursor=connection.cursor()
```

```
        if response == '1': # Show all recipes
            cbk.PrintAllRecipes()
        elif response == '2': # Search for a recipe
            pass
        elif response == '3': # Show a single recipe
            cbk.PrintAllRecipes()
        elif response == '4': # Delete Recipe
            cbk.PrintAllRecipes()
        elif response == '5': # Add a recipe
            pass
        elif response == '6': # Print a recipe
            cbk.PrintAllRecipes()
        elif response == '0': # Exit the program
            print 'Goodbye'
            loop = False
        else:
            print 'Unrecognized command.  Try again.'
```

Először létrehozunk két globális változót a kapcsolathoz (connection) és a kurzorhoz (cursor). Mindkettőt bárhonnan el tudjuk érni a cookbook (szakácskönyv) osztályból. Ezután létrehozunk egy self.totalcount változót, amivel megszámoljuk a recepteket. Ezt a változót a későbbiekben is használni fogjuk. Végezetül létrehozzuk a kapcsolatot és a kurzort.

A következő lépésben összerakjuk a Cookbook osztály PrintAllRecipes() eljárását. Mivel a connection és a cursor változók globálisak, ezért nem kell minden metódusban újra létrehozni őket. Ezt követően, szeretnénk a képernyőre „csinosan kiíratni” a receptlista fejlécét. A megfelelő szóközölés eléréséhez használni fogjuk a „%s” formázási parancsot, és a balra zárást. Valami ilyesmit szeretnénk kapni:

```
Item Name    Serves    Source
-------------------------
```

Végül, létre kell hoznunk az SQL utasítást, ami lekérdezi az adatbázist és kiíratja a találatokat. A múlthavi cikkben ennek nagyját már tárgyaltuk.

```
    sql = 'SELECT * FROM
Recipes'
    cntr = 0
    for x in
cursor.execute(sql):
        cntr += 1
        print '%s %s %s %s'
%(str(x[0]).rjust(5),x[1].lju
st(30),x[2].ljust(20),x[3].lj
ust(30))
    print '-------------'
    self.totalcount = cntr
```

A cntr változóban fogjuk számolni a képernyőn megjelenített receptek számát. Ezen a ponton már készen is vagyunk a metódussal. Ha esetleg kihagytál volna valamit, lentebb megtalálod a szubrutin teljes kódját.

Figyeljük meg hogy használjuk az ASPW cursor.execute eljárás által visszaadott vektort. A pkID-t, mint itemet írjuk ki, így a későbbiekben majd ki tudjuk választani a megfelelő receptet. Amikor a programot lefuttatjuk, meg kellene jelennie a menünek, illetve amikor kiválasztjuk az 1-est, akkor a következő oldal tetején levő képet fogjuk látni.

Épp ezt akartuk elérni. Ha Dr. Pythont vagy hasonlót használunk, akkor csak annyi a gond, hogy az alkalmazásunk nem áll meg. Egyszerűen várakozzunk, amíg a felhasználó meg nem nyom egy gombot, így lesz pár perce a kimenet végigbogarászására. Ha már itt vagyunk, akkor írassuk is ki az előbb beállított változóból az összes recept számát. Helyezzük a menu 1-es választási lehetősége alá:

```
def PrintAllRecipes(self):
        print '%s %s %s %s'
%('Item'.ljust(5),'Name'.ljust(30),'Serves'.ljust(20),
'Source'.ljust(30))
        print '---------------------------------'
        sql = 'SELECT * FROM Recipes'
        cntr = 0
        for x in cursor.execute(sql):
            cntr += 1
            print '%s %s %s %s'
%(str(x[0]).rjust(5),x[1].ljust(30),x[2].ljust(20),x[3
].ljust(30))
        print '---------------------------------'
        self.totalcount = cntr
```

```
Enter a selection -> 1
Item  Name                              Serves              Source
-------------------------------------------------------------------------------
    1 Spanish Rice                      4                   Greg
    2 Pickled Pepper-Onion Relish       9 half pints        Complete Guide to Home Canning
-------------------------------------------------------------------------------
================================================
               RECIPE DATABASE
================================================
1 - Show All Recipes
2 - Search for a recipe
3 - Show a Recipe
4 - Delete a recipe
5 - Add a recipe
6 - Print a recipe
0 - Exit
================================================
Enter a selection ->
```

```
print 'Total Recipes - %s'
%cbk.totalcount

print '----------------------
-------------------------
-'

res = raw_input('Press A Key
-> ')
```

A második elemet (Recept keresése) ugorjuk egyelőre át, és foglalkozzunk a hármassal (egyetlen recept kiíratása). Először foglalkozzunk a menü részével. Itt íratjuk ki a receptek listáját, akárcsak, az első opciónál, majd megkérjük a felhasználót, hogy válasszon. Ahhoz, hogy a hibás inputokkal szemben bebiztosítsuk magunkat, használni fogjuk a Try|Except vezérlési szerkezetet. Meg fogjuk jeleníteni a felhasználó felé a promptot (Select a recipe -->), majd, ha egy helyes választ gépelnek be, meghívjuk a PrintSingle Recipe() rutint a Recipe táblabeli pkID-vel a Cookbook osztályból. Ha a megadott érték nem szám, akkor egy ValueError kivétel dobódik, melyet a ValueError: fog majd elkapni.

Következőnek, a Cookbook osztályban lévő PrintSingleRecipe metóduson fogunk dolgozni. Ismét a kapcsolattal és a kurzorral kezdünk, majd ezután létrehozzuk az SQL utasítást is. Ebben az esetben a „SELECT * FROM Recipes WHERE pkID= %s"%str(which)” lekérdezést használjuk, ahol a where az az érték, amit meg szeretnénk találni. Ezt követően az outputot „csinosan kiíratjuk” az ASPW által visszaadott tuple-ből. Ebben az esetben az x-et egy általános változóként használjuk, és az egyes elemekre a zárójelezett indexel hivatkozunk a tuple-ban. Mivel a táblázat elrendezése pkID/name/servings/source, az x[0], x[1], x[2] és x[3] értékeket fogjuk megjeleníteni. Ezután a „wantot” fogjuk használni ahhoz, hogy mindent kiválasszunk a hozzávalók táblájából, ahol a recipeID (a recipe adattábla kulcsa) megegyezik azzal a pkID-vel, amit éppen használtunk. Végigiterálunk a visszaadott tuple-ön, mindegyik hozzávalót kiíratva, majd az instructions táblából kivesszük az instrukciókat - mint ahogy ezt az ingredients táblánál tettük. Végül, a felhasználó gombnyomására várakozunk, hogy a képernyőn meg tudjuk jeleníteni a receptet. A kód a következő oldalon látható.

```
try:
    res = int(raw_input('Select a Recipe -> '))
    if res <= cbk.totalcount:
        cbk.PrintSingleRecipe(res)
    elif res == cbk.totalcount + 1:
        print 'Back To Menu...'
    else:
        print 'Unrecognized command.  Returning to menu.'
except ValueError:
    print 'Not a number...back to menu.'
```

Most már a hatból két metódussal készen vagyunk. Foglalkozzunk akkor a keresés rutinnal, megint a menüvel kezdve. Szerencsénkre, csak a keresés utasítás meghívására van szükségünk az osztályban. Cseréljük le a pass parancsot:

```
cbk.SearchForRecipe()
```

Rakjuk most össze a keresés algoritmusát. A Cookbook osztályban, a SearchForRecipe autogenerált törzsét cseréljük le a következő oldalon látható kódra.

Jó sok minden van itt. Miután létrehoztuk a kapcsolatunkat és a kurzorunkat, megjelenítjük a keresés menüt. A felhasználónak három keresési lehetőséget, illetve egy kilépés rutint fogunk felkínálni. A felhasználó egy szót kereshet a recept nevében, forrásában, vagy a hozzávalók listájában. Emiatt nem tudjuk egyszerűen felhasználni az imént megírt megjelenítési metódust és létre kell hoznunk egy személyreszabott kiíratási rutint. Az első két opció egyszerű SELECT utasításokat használ - egy kis csavarral megtoldva. Mégpedig a „like” minősítőt használjuk. Ha egy „SQLite Database Browser” szerű lekérdezés-szerkesztőt használunk, akkor a like utasításaink a „%” wildcardot fogják használni. Tehát, amikor „rice”-t (rizs) tartalmazó receptnévre keresünk, akkor a lekérdezés ilyen lenne:

```
SELECT * FROM Recipes WHERE
name like '%rice%'
```

Mivel, a „%” karakter a sztringeknél egy helyettesítési karakter is, a %%-ot kell használnunk a szövegünkben. Ami még rosszabb, az az, hogy egy helyettesítő karaktert használunk a felhasználó által keresett szó beillesztésére is. Ezért, a „%%%s%%” karaktersorozathoz jutunk. Hát, nem éppen szép. A harmadik lekérdezést Join utasításnak nevezzük. Nézzük meg egy kicsit közelebbről:

```
sql = "SELECT
r.pkid,r.name,r.servings,r.so
urce,i.ingredients FROM
Recipes r Left Join
ingredients i on (r.pkid =
i.recipeid) WHERE
i.ingredients like '%%%s%%'
GROUP BY r.pkid" %response
```

Mindent kiválasztunk a recipe táblából és a hozzávalókat az ingredients táblából, a kapcsolatot, vagy a hozzárendelést a recipeID és a pkID alapján megvalósítva. Ezután a like utasítással megkeressük a hozzávalóinkat, majd végül az eredményeket csoportosítjuk a recepttáblabeli pkID szerint, ezzel elkerülve, hogy a többszörös adatok megjelenjenek. Emlékezzünk arra, hogy két paprika van a második receptünkben (hagyma és paprika fűszerek), egy zöld és egy piros. Ez megzavarhatja a felhasználónk fejét. A menünk a

```
  searchin = raw_input('Enter
Search Type -> ')

if searchin != '4':
```

megoldást használja, ami azt mondja: ha a searchin (a felhasználó által begépelt érték) NEM egyenlő 4-gyel, akkor az

```
  def PrintSingleRecipe(self,which):
        sql = 'SELECT * FROM Recipes WHERE pkID = %s' %
str(which)
        print
'~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~'
        for x in cursor.execute(sql):
            recipeid =x[0]
            print "Title: " + x[1]
            print "Serves: " + x[2]
            print "Source: " + x[3]
        print
'~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~'
        sql = 'SELECT * FROM Ingredients WHERE RecipeID =
%s' % recipeid
        print 'Ingredient List:'
        for x in cursor.execute(sql):
            print x[1]
        print ''
        print 'Instructions:'
        sql = 'SELECT * FROM Instructions WHERE RecipeID
= %s' % recipeid
        for x in cursor.execute(sql):
            print x[1]
        print
'~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~'
        resp = raw_input('Press A Key -> ')
```

```
    def SearchForRecipe(self):
        # print the search menu
        print '------------------------------'
        print ' Search in'
        print '------------------------------'
        print ' 1 - Recipe Name'
        print ' 2 - Recipe Source'
        print ' 3 - Ingredients'
        print ' 4 - Exit'
        searchin = raw_input('Enter Search Type -> ')
        if searchin != '4':
            if searchin == '1':
                search = 'Recipe Name'
            elif searchin == '2':
                search = 'Recipe Source'
            elif searchin == '3':
                search = 'Ingredients'
            parm = searchin
            response = raw_input('Search for what in %s (blank to exit) -> ' % search)
            if parm == '1': # Recipe Name
                sql = "SELECT pkid,name,source,servings FROM Recipes WHERE name like '%%%s%%'" %response
            elif parm == '2': # Recipe Source
                sql = "SELECT pkid,name,source,servings FROM Recipes WHERE source like '%%%s%%'" %response
            elif parm == '3': # Ingredients
                sql = "SELECT r.pkid,r.name,r.servings,r.source,i.ingredients FROM Recipes r Left Join ingredients i
on (r.pkid = i.recipeid) WHERE i.ingredients like '%%%s%%' GROUP BY r.pkid" %response
            try:
                if parm == '3':
                    print '%s %s %s %s %s'
%('Item'.ljust(5),'Name'.ljust(30),'Serves'.ljust(20),'Source'.ljust(30),'Ingredient'.ljust(30))
                    print '--------------------------------------------------------------------------------'
                else:
                    print '%s %s %s %s' %('Item'.ljust(5),'Name'.ljust(30),'Serves'.ljust(20),'Source'.ljust(30))
                    print '--------------------------------------------------------------------------------'
                for x in cursor.execute(sql):
                    if parm == '3':
                        print '%s %s %s %s %s'
%(str(x[0]).rjust(5),x[1].ljust(30),x[2].ljust(20),x[3].ljust(30),x[4].ljust(30))
                    else:
                        print '%s %s %s %s' %(str(x[0]).rjust(5),x[1].ljust(30),x[3].ljust(20),x[2].ljust(30))
            except:
                print 'An Error Occured'
            print '--------------------------------------------------------------------------------'
            inkey = raw_input('Press a key')
```

opciókkal foglalkozunk, ha 4, akkor nem csinálunk semmit, csak túllépünk rajta. Vegyük észre, hogy a „!=” (nem egyenlő) operátort használtuk a „<>” operátor helyett. Bármelyik működik Python 2.x alatt, de Python 3.x-ban szintaxishibát kapunk. A későbbiekben több Python 3.x-et érintő változással is fogunk foglalkozni. Egyelőre használjuk a „!=” operátort, hogy a jövőben megkönnyítsük az áttérést Python 3.x-ra. Végül, „csinosan kiíratjuk” az outputot. Jobbra láthatjuk, hogy mit fog kapni a felhasználó.

Láthatjuk, hogy programunk milyen szépen írja ki a kimenetet. Most a felhasználó már visszaléphet a menübe és használhatja a 3-as opciót bármelyik recept kiíratásához. Következőnek recepteket fogunk az adatbázisunkhoz hozzáadni. Ismét csak egyetlen sort kell a menu rutinunkban elhelyezni. Ez az EnterNew eljárás:

```
cbk.EnterNew()
```

Itt van a kód, melynek az EnterNew() kódtöredéket kell felváltania: http://pastebin.com/f1d868e63.

```
Enter a selection -> 2
------------------------------
Search in
------------------------------
1 - Recipe Name
2 - Recipe Source
3 - Ingredients
4 - Exit
Enter Search Type -> 1
Search for what in Recipe Name (blank to exit) -> rice
Item  Name                            Serves                Source
-------------------------------------------------------------------------------
    1 Spanish Rice                    4                     Greg
-------------------------------------------------------------------------------
Press a key
```

Elég egyértelmű. Most, ami a hozzávalók keresését illeti...

```
Enter a selection -> 2
------------------------------
Search in
------------------------------
1 - Recipe Name
2 - Recipe Source
3 - Ingredients
4 - Exit
Enter Search Type -> 3
Search for what in Ingredients (blank to exit) -> onion
Item  Name                  Serves            Source                          Ingredient

-------------------------------------------------------------------------------
1 Spanish Rice                4               Greg                            1 small
Onion chopped
2 Pickled Pepper-Onion Relish  9 half pints   Complete Guide to Home Canning 6 cups
finely chopped Onions
-------------------------------------------------------------------------------
Press a key
```

Az „ings” nevű lista definiálásával kezdünk - ami az ingredients rövidítése. Ezután megkérjük a felhasználót, hogy adja meg a nevet, forrást és adagok számát. Ezután egy ciklusba lépünk be, ahol mindegyik hozzávalót bekérjük, majd ezeket rendre az ing listához hozzáfűzzük. Ha a felhasználó 0-t ír be, akkor kilépünk a ciklusból és bekérjük az instrukciókat. Ezután a recept tartalmát újra megmutatjuk és megkérjük a felhasználót, hogy ellenőrizze, mielőtt elmentenénk. Akárcsak múltkor, megint INSERT INTO utasításokat használunk, mielőtt kilépünk a menübe. Egy dologra azonban ügyelnünk kell: a bejegyzéseknél lévő egyszeres idézőjelekre. ÁLTALÁBAN a hozzávalók listájánál, illetve az instrukcióknál ez nem okoz problémákat, de a név és forrás mezőknél előjöhetnek. Ezért minden egyes idézőjelnél egy escape szekvenciát kell elhelyeznünk. Ezt a string.replace eljárással tesszük meg. Ezért importáltuk be a string könyvtárat. Helyezzük el a menü rutinban a 4-es opció után a jobbra fent látható kódot.

Majd a Cookbook osztályban használjuk a jobbra lent látható kódot a DeleteRecipe() eljáráshoz.

Gyorsan fussunk át a törlő metóduson is. Először megkérdezzük a felhasználótól, hogy melyik receptet szeretné törölni (még a menüben), majd átadjuk ezt a pkID-t a törlő eljárásnak. Ezután megbizonyosodunk, hogy a felhasználó biztosan szeretné-e törlni a receptet. Ha a válasz „Y” (string.upper(resp) == 'Y'), akkor létrehozzuk a törlő sql utasítást. Vegyük észre, hogy ezúttal mind a három táblából törölnünk kell rekordokat. Természetesen törölhetnénk egyedül csak a recipe táblából, de ekkor árva rekordjaink maradnának a másik kettőben. Ez pedig nem lenne jó. A pkID mezőket használjuk a rekord recipe táblából való törlésekor, a másik két táblában pedig a recipeID mezőket.

És végül foglalkozzunk a recept kinyomtató eljárásával. Egy NAGYON egyszerű HTML fáljt hozunk majd létre. Ezt követően megnyitjuk az alapértelmezett böngészőt, ahol már ki tudják nyomtatni a receptet. A webbrowser könyvtárat ezért importáltuk be. A menü eljárás 6-os opciójában helyezzük el a következő oldalon fent látható kódot.

```
cbk.PrintAllRecipes()
     print '0 - Return To Menu'
     try:
         res = int(raw_input('Select a Recipe to DELETE
or 0 to exit -> '))
         if res != 0:
             cbk.DeleteRecipe(res)
         elif res == '0':
             print 'Back To Menu...'
         else:
             print 'Unrecognized command.  Returning to
menu.'
     except ValueError:
          print 'Not a number...back to menu.'
```

```
    def DeleteRecipe(self,which):
        resp = raw_input('Are You SURE you want to Delete
this record? (Y/n) -> ')
        if string.upper(resp) == 'Y':
            sql = "DELETE FROM Recipes WHERE pkID = %s" %
str(which)
            cursor.execute(sql)
            sql = "DELETE FROM Instructions WHERE
recipeID = %s" % str(which)
            cursor.execute(sql)
            sql = "DELETE FROM Ingredients WHERE recipeID
= %s" % str(which)
            cursor.execute(sql)
            print "Recipe information DELETED"
            resp = raw_input('Press A Key -> ')
        else:
            print "Delete Aborted - Returning to menu"
```

Ismét megjelenítjük a receptek listáját, és megengedjük, hogy kiválasszhassák azt, amelyiket ki szeretnék nyomtatni. A Cookbook osztály PrintOut eljárását hívjuk meg. (Következő oldalon lent látható.)

A fi = open([filename],'w') utasítással kezdünk, ami létrehoz egy fájlt. Ezután lekérdezzük a recipe tábla adatait, majd a fi.write-tal kiíratjuk a fájlba.

A <H1></H1> címsor tagokat használjuk a névhez, a <H2> tagot az adagokhoz és forrásokhoz. A tagot a hozzávalók és instrukciók listájánál használjuk. Ezektől eltekintve, csak egyszerű, már tanult lekérdezéseket tartalmaz. Végül, a fi.close() utasítással bezárjuk a fájlt, és meghívjuk a webbrowser.open([filename]) függvényt az előbb létrehozott fáljra. A felhasználó ezután a böngészőből ki tudja nyomtatni a receptet - ha szükséges.

```
cbk.PrintAllRecipes()
       print '0 - Return To Menu'
       try:
           res = int(raw_input('Select a Recipe to DELETE or 0 to exit -> '))
           if res != 0:
               cbk.PrintOut(res)
           elif res == '0':
               print 'Back To Menu...'
           else:
               print 'Unrecognized command.  Returning to menu.'
       except ValueError:
           print 'Not a number...back to menu.'
```

HUHH!! Ez volt eddigi legnagyobb alkalmazásunk. A teljes forráskódot kiraktam (a minta adatbázist is, ha kihagytad volna az előző hónapot) a weblapomra. Ha nem akarod az egészet begépelni, akkor ugorj el a www.thedesignatedgeek.com címre, és töltsd le a kódot.

```
    def PrintOut(self,which):
        fi = open('recipeprint.html','w')
        sql = "SELECT * FROM Recipes WHERE pkID = %s" % which
        for x in cursor.execute(sql):
            RecipeName = x[1]
            RecipeSource = x[3]
            RecipeServings = x[2]
        fi.write("<H1>%s</H1>" % RecipeName)
        fi.write("<H2>Source: %s</H2>" % RecipeSource)
        fi.write("<H2>Servings: %s</H2>" % RecipeServings)
        fi.write("<H3> Ingredient List: </H3>")
        sql = 'SELECT * FROM Ingredients WHERE RecipeID = %s' % which
        for x in cursor.execute(sql):
            fi.write("<li>%s</li>" % x[1])
        fi.write("<H3>Instructions:</H3>")
        sql = 'SELECT * FROM Instructions WHERE RecipeID = %s' % which
        for x in cursor.execute(sql):
            fi.write(x[1])
        fi.close()
        webbrowser.open('recipeprint.html')
        print "Done"
```

Greg Walters a RainyDay Solutions tulajdonosa, ez egy korlátolt felelősségű tanácsadó cég a colorado-i Aurorában. Programozással 1972 óta foglalkozik. Szeret főzni, túrázni, zenét hallgatni és szabadidejét családja körében eltölteni.

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 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
Imolai Gábor (külső felajánlás)

Szerkesztő, korrektor:
Heim Tibor

Köszönet a Canonical-nek és a fordítócsapatoknak világszerte.