Full Circle

AZ UBUNTU LINUX KÖZÖSSÉG FÜGGETLEN MAGAZINJA

Programozói sorozat – Különkiadás

Programozói sorozat Különkiadás

Programozzunk Pythonban

10. 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” 54–59. részének** az újabb kiadása (a magazin 85–87, 91, 95, 100. 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 – 54. rész

Írta: Greg D. Walters
Fordította: Sipos Zoltán

Sok évvel ezelőtt, magas vérnyomásban szenvedtem. A doktorom azt javasolta, hogy csináljak valamit ami lefoglal, elég hasznos, egyben hétköznapi is. Megfogadtam a tanácsát és kipróbáltam a számlált keresztöltést. Ez kreatív, összpontosítást kíván és segít elterelni a gondolataidat a zavaró dolgokról. Úgy érzem megint előjött a problémám, így előkerítettem a rámát meg a tűket és újra nekikezdtem.

Abban az esetben ha nem ismered a számlált keresztöltést, nagyvonalakban elmondom miről is van szó. A keresztöltés a kézimunka egy fajtája, amikor egy szálból formált apró „x” minták végül egy képet hoznak létre. A szál neve „fonal”, a használt szövet neve pedig „aida”. A Wikipedia szerint az aida egy speciális szövet, amin szabályos távolságokra apró lukak találhatók, amik apró négyzeteket formálnak. Ez megkönnyíti a képet alkotó „x” minták elhelyezését. A keresztöltésnek két típusa van. Az egyiknél egy kép van nyomtatva az aidára (egyfajta színezd ki a számokat), a másiknál teljesen üres az aida a minta kiöltéséhez. A második sokkal nehezebb mint az első. Menj csak el egy rövidáru boltba vagy a helyi lakberendezési boltba és máris megtudod miről van szó.

Nem is olyan régen elkezdtem írogatni egy programot, ami egy képből keresztöltésmintát állított elő. Egyik dolog jött a másik után és polcra kellett tennem a programot. De most leporoltam az ötletet és újra nekikezdtem.

A következő néhány cikket ennek a feladatnak fogjuk szentelni. El fog tartani egy darabig, mert néhány dolog elég összetett és sok részből tevődik össze. Itt van a „játékmenet”:

- Adatbázis létrehozása a pixelszínekhez és a fonalszínekhez.
- GUI létrehozása az alkalmazáshoz a Tkinter segítségével.
- Alkalmazás felruházása a képfájlok feldolgozásának képességével.
- PDF fájl létrehozása, ami a végső minta lesz a feladat számára.

Amivel találkozni fogsz

- Adatbázis és XML kezelés.
- Tkinter GUI programozás. Ha kihagytad az ezzel foglalkozó előző cikkeket, megtalálod őket az FCM 51-től 54 -ig tartó kiadásaiban.
- Képfeldolgozás PIL segítségével (http://pillow.readthedocs.org/en/latest/).
- PDF létrehozás pyFPDF segítségével (https://code.google.com/p/pyfpdf/).

Lássunk hozzá

Az első dolog az adatbázis létrehozása, ami tárolja a DMC(TM) fonalszíneket és hozzárendeli őket a legközelebbi RGB (Piros, Zöld, Kék) értékekhez, amiket a képekben használnak a számítógépeken. Ugyanakkor az adatbázis tárolni fogja a hexa és a HSV (Színezet, Telítettség, Világosság) értékeket is mindegyik fonalszín esetén. Úgy tűnik, hogy a HSV használatával a legegyszerűbb megtalálni a legközelebbi színértéket, ami a fonal színével fog megegyezni. Természetesen a végső döntéshozó az emberi szem. Ha nem mozogsz otthonosan a HSV színábrázolásban, a Wikipedián található egy elég összetettnek mondható leírás: http://en.wikipedia.org/wiki/HSL_and_HSV. Lehet, hogy segít, de lehet, hogy nem lesz érhetőbb.

Az első dolog ami kell nekünk, az egy XML fájl, ami tartalmazza a DMC fonalszíneket RGB formátumban. A legjobb amit találtam a http://sourceforge.net/p/kxstitch/feature-requests/9/ címen található. A fájl ami kell, az a dmc.xml. Töltsd le a fájlt és helyezd abba a mappába, ahol majd a Python-kód lesz tárolva.

Most az apsw-t fogjuk használni az adatbázisunk kezeléshez, amivel már kellett hogy találkozz, az XML feldogozásához pedig az ElementTree-t (ami a Python 2.7+ verzióban

```
# makedb.py
# DMC.xml to SQLite database
# For Full Circle Magazine #85

import apsw
from xml.etree import ElementTree as ET
tablename = "DMC"
```

található). Nézzük akkor a kódot.

Ahogy mindig, most is importálással kezdünk. Ebben a programban csak ez a kettő van. Beállítjuk a tábla nevét is.

Abban az esetben, ha állandó olvasója vagy a cikkeknek, a következő résznek ismerősnek kell lennie. Létrehozzuk a függvényt, ami az XML fájlt fogja olvasni és elemezni. Utána ezt az információt az adatbázis feltöltésére használhatjuk. Egy részlet az XML fájlból jobbra fent.

Megkeressük a <floss> tagot mindegyik adatelemhez. Ehhez a .findall('floss') parancsot használjuk. Amint megvan az adatelem, az adatbázisba helyezéshez mindegyik tagot (name, description, stb.) külön változóba mentjük. Amikor a <color> taghoz érünk, a .floss.findall('color') parancsot használjuk, hogy megkapjuk a piros, zöld, kék értékeket.

Azzal kezdjük, hogy megmondjuk a függvénynek, hogy a connection és a cursor nevű globális változókat fogjuk használni. Utána beállítjuk az XML fájl nevét, és kielemezzük az XML fájlt. Egy számlálóváltozót is használunk még, ami a feldolgozás és az adatbázis feltöltése alatt mutatja, hogy valami folyamatban van.

Most, hogy megvan az összes adatunk, meg kell írnunk az SQL utasítást majd futtatni azt. Vedd észre a „\"-t a VALUES után az SQL utasításban. Az egy sort-nem-törő karakter, ami megkönnyíti a megjelenést itt a magazinban. Mindjárt létre fogjuk hozni az adatbázist és a táblát is.

```
SQL = "INSERT INTO DMC
(DMC,Description,Red,Green,Bl
ue) VALUES \

      ('%s','%s',%s,%s,%s)"
% (name,desc,red,green,blue)

cursor.execute(SQL)
```

Most a terminálablakba írunk, hogy valami folyamatban van:

```
print "Working record
{0}".format(cntr)

cntr += 1
```

Most létrehozzuk és/vagy megnyitjuk az adatbázist az OpenDB rutinban. Ha olvastad a részt, amikor az adatbázissal foglalkoztunk, akkor fel fog tünni, hogy ez alkalommal két kurzort használunk. A „cursor" változó van használva a „normál" adatbevitelekhez és később a SELECT utasításban az „update"-hez a hexa és HSV értékek

```
<floss>
    <name>150</name>
    <description>Dusty Rose Ultra VDK</description>
    <color>
         <red>171</red>
         <green>2</green>
        <blue>73</blue>
    </color>
</floss>
```

```
def ReadXML():
    global connection
    global cursor
    fn = 'dmc.xml'
    tree = ET.parse(fn)
    root = tree.getroot()
    cntr = 0
    for floss in root.findall('floss'):
        name = floss.find('name').text
        desc = floss.find('description').text
        for colour in floss.findall('color'):
            red = colour.find('red').text
            green = colour.find('green').text
            blue = colour.find('blue').text
```

```
def OpenDB():
    global connection
    global cursor
    global ucursor
    global dbname
    connection = apsw.Connection("floss.db3")
    cursor = connection.cursor()
    ucursor = connection.cursor()
```

beállításánál. Két kurzort kell használnunk, mert ha a logikai utasítás közepén módosítod a kurzort, akkor mindent elvesztesz az új parancs miatt. Az „ucursor"-t viszont már használhatjuk az update utasításhoz. Ennek kivételével, ez a normál OpenDB rutinunk.

Most hogy az adatbázis létrejött és/vagy meg van nyitva, beállíthatjuk a táblánkat. Vedd észre hogy az SQL utasítás hármas aposztrófozást használ, hogy a sor szépen iga-

zodjon a megjelenéshez.

Az EmptyTables rutin azért van, hogy ha az alkalmazást egynél többször akarjuk vagy szükséges futtatni, akkor tiszta és üres táblával induljunk.

Ha itt megállnánk, már akkor is lenne egy értelmes, működő adatbázisunk a DMC színnel, a szín nevével és az RGB értékekkel. De ahogy már említettem, a HSV adat használatával könnyebb kiválasztani a legközelebbi fonalszínt.

Következőnek az RGB értékekből hexa értéket hozunk létre. A következő függvény a HSV értékeket hozza létre az RGB értékekből . Az algoritmust az interneten találtam. Felkutathatod ha akarod.

Végül megírjuk az UpdateDB függvényt. A SELECT * FROM DMC utasítást használjuk és az adat tárolásához a „standard” cursor változót. Utána végiglépkedünk a visszakapott adatokon, kiolvassuk az RGB értékeket és átadjuk az rgb2hex függvénynek egy értékként, majd a rgb2hsv függvénynek három különálló értékként. Amint megkapjuk a visszatérő értékeket, az update SQL utasítást használjuk, a megfelelő rekord kiválasztásához az elsődleges kulccsal együtt (pkID). Ahogy már említettem, az update utasításnál egy másik kurzort kell használnunk.

```
def MakeTables():
    sql = '''CREATE TABLE IF NOT EXISTS DMC
        (pkID INTEGER PRIMARY KEY, DMC INTEGER,
         Description TEXT, Red INTEGER, Green INTEGER, Blue INTEGER,
         HEX TEXT,H INTEGER,S INTEGER,V INTEGER)'''
    cursor.execute(sql)
```

```
def rgb2hex(rgb):
    return '%02x%02x%02x' % rgb
```

```
def EmptyTables():
    sql="DELETE FROM %s" % tablename
    cursor.execute(sql)
```

Az utolsó dolog amit csinálunk, hogy az adatbázis létrehozásához meghívjuk az összes függvényt és a végén kiírjuk hogy „Befejezve” így a felhasználó tudja hogy minden elkészült.

```
OpenDB()
MakeTables()
EmptyTables() # Just to be
safe
ReadXML()
UpdateDB()
print "Finished"
```

A programot „MakeDB”-nek neveztem el. Az adatbázisnak ugyanabban a mappában kellene létrejönnie, mint ahol a kód és az XML fájl található. Mint mindig a teljes kód megtalálható a http://pastebin.com/Zeggw3pi címen.

```
def rgb2hsv(r, g, b):
    r, g, b = r/255.0, g/255.0, b/255.0
    mx = max(r, g, b)
    mn = min(r, g, b)
    df = mx-mn
    if mx == mn:
        h = 0
    elif mx == r:
        h = (60 * ((g-b)/df) + 360) % 360
    elif mx == g:
        h = (60 * ((b-r)/df) + 120) % 360
    elif mx == b:
        h = (60 * ((r-g)/df) + 240) % 360
    if mx == 0:
        s = 0
    else:
        s = df/mx
        v = mx
    return int(round(h,0)), int(round(s*100,0)),
int(round(v*100,0))
```

Következőnek a GUI-val fogunk foglalkozni. A GUI-hoz a Tkintert használjuk, de addig is felfrissítheted az emlékeidet visszalapozva az FCM 51-től 54-ig tartó kiadásaiba, ahol a Tkintert ismertetem.

A következő találkozásig érezzétek jól magatokat.

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

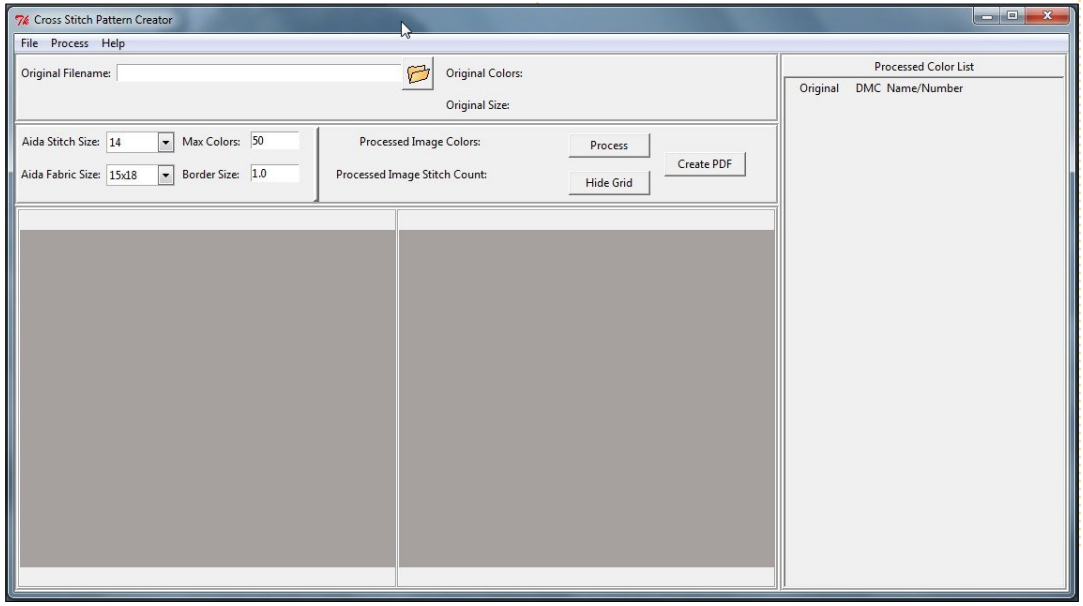
Programozzunk Pythonban – 55. rész

Írta: Greg D. Walters
Fordította: Sipos Zoltán

Ez a több részes oktatóanyagunk második része a hímzésminta generátorról. Az első részben (FCM85) létrehoztunk egy adatbázist mely a DMC(TM) fonal színeket és a hozzájuk legközelebb álló RGB értékeket tartalmazta. Ebben a részben a Tkinter segítségével létrehozzuk a GUI-t. Továbbá használni fogjuk még a PIL-t (Python Imaging Library) és a PMW-t (Python Mega Widgets). Ezeket a szoftverkönyvtárakat le kell majd töltened és telepítened kell őket, hogy egyáltalán jussunk valamire. A PIL-ért keresd fel a Pillow forkot a https://github.com/python-imaging/Pillow címen és töltsd le a legfrissebb verziót. A PMW-t a http://pmw.sourceforge.net/ címről tudod letölteni.

Szükséged lesz még 2 kép fájlra is. Az egyik egy egyszerű 500x400 pixel méretű szürke téglalap. A létrehozásához használhatod a GIMP-et vagy valamelyik másik képszerkesztő programot. Nevezd el default.jpg-nek és helyezd a forráskód könyvtárába az adatbázis mellé. A másik egy mappának a képe, a képet megnyitó gomb számára. Az open clipartról szereztem be egyet ahol a „mappa” szóra kerestem rá. Egy megfelelőnek tűnőt a https://openclipart.org/detail/177890/file-folder-by-thebyteman-177890 linken találtam. Nyisd meg a képet a GIMP-ben, méretezd át 30x30-ra és mentsd el open.gif néven ugyanabba a könyvtárba ahol a másik két fájl is található.

A lenti egy képernyőkép, ahogyan a kész GUI fog kinézni. A GUI-ben négy fő frame található. Három bal oldalt és egy a jobb oldalt. A widget létrehozása műveletnél úgy fogok rájuk hivatkozni mint Felső Frame, Középső Frame, Alsó Frame és Oldalsó Frame. A felső frame az eredeti képet kezeli. A középső frame a kép feldolgozásával foglalkozik. Az alsó frame megjeleníti az eredeti képet a bal oldalon, a feldolgozott képet a jobb oldalon és az oldalsó frame pedig a színeket és a szükséges fonalat jeleníti meg. Első ránézésre úgy tűnik, hogy itt egy csomó kihasználatlan hely van, de amikor már fut a program és túljutottunk a feldolgozási részen, akkor már nem igazán marad ilyen sok üres hely.



Most már készen állunk, hogy hozzákezdjünk a kódoláshoz. Itt van az importunk hosszú listája...

```
from Tkinter import *
import tkFileDialog
import tkCommonDialog
import tkMessageBox
import ttk

from PIL import
Image,ImageTk,ImageOps
import Pmw
import apsw # Database
Access
import math # Math library
import sys
```

Az importok teljes számából kiindulva azt mondhatod, hogy ez egy hosszú program lesz. Ténylegesen a kód UI része a kommentekkel együtt több mint 300 sor lesz. A „jó” hír az, hogy kb. 200 sornyi kód a program Tkinter részével foglalkozik, ami maga az aktuális GUI. Ebben a részben a hátralévő sorok többsége egy üres függvény, és a következő részben jutnak szerephez.

Létre fogunk hozni egy osztályt ami az összes UI feldolgozó kódot tartalmazza (következő oldal jobb felül).

Elsőnek definiálva van az osztály, majd az __init__ függvény aminek átadjuk a „root” TopLevel ablakot. A TopLevel ablakot a program utolsó négy sorában hozzuk létre. Az __init__ függvényen belül definiáljuk az összes globális változót és néhány kezdeti értékadást hajtunk végre mielőtt hozzákezdünk a többi függvényhez. Az első dolog amit csinálunk, hogy a kép fájlformátumok tárolásához létrehozunk egy Tuple listát amire akkor van szükségünk amikor meghívjuk az OpenFile dialógus ablakot. A következő két sor lent definiálja és előkészíti a két kép fájlt, amit most hoztunk létre (nyitott mappa GIF fájl és a szürke téglalap – ami majd kijelöli a helyet a minta létrehozásához használt képeink számára.

```
self.openimage =
PhotoImage(file='open.gif')

self.DefaultImage
=ImageTk.PhotoImage(self.Thum
bnail("default.jpg”,450,450))
```

Most belevetjük magunkat a globális definiciókba (jobb középen). Ha emlékszel, amikor a Tkinter-t használod és van egy widgeted mint egy szöveg beviteli mező vagy egy lenyíló lista és el akarod érni a beírt vagy a kiválasztott adatokat, akkor egy globális változót definiálsz és hozzárendeled egy Változó Osztályhoz (BooleanVar, DoubleVar, IntVar vagy StringVar). Ez követni fogja a widget értékekben bekövetkezett változásokat, így elérheted azokat a .get() vagy a .set() metódusokkal. A következő kódsorokban létrehozzuk a globális változó nevet, majd hozzárendeljük a megfelelő wrapper osztályhoz. Segítségképpen beillesztettem néhány kommentet a kódba, hogy követni tudd, hogy mit is csinálunk.

Ahogy látod, beállítunk egy OriginalFilename nevű változót a kép tárolásához, amiből a mintát akarjuk létrehozni, egy OriginalColorCount nevű változót, ami az eredeti képfájlban levő színek számát tárolja és egy OriginalSize nevű változót ami az eredeti kép pixelméretét tárolja. Ahogy a tv-ben modják... ***„DE VÁRJON! EZ MIND SEMMI!”*** (jobb lent):

Egy lenyíló lista beállítja a ComboStitch nevű változót ami a feladatban használt aida öltésméretét kezeli. A ComboSize nevű változót szintén egy lenyíló lista állítja be és az aida anyagméretét tárolja. A FabricHeight és a FabricWidth az aida méretének a felbontása. A MaxColors egy beviteli mezőből származó érték a színek számának a beállításához és a BorderSize egy lebegőpontos érték, ami megadja a ráma miatt nem használt részt az aida-ból.

```
class XStitch:
    def __init__(self, master):
        self.picFormats = [
        ('JPEG / JFIF','*.jpg'),
        ('Portable Network Graphics','*.png'),
        ('CompuServer GIF','*.gif'),
        ('Windows Bitmap','*.bmp'),
        ('All File Types *.*','*.*'),
        ]
```

```
        #-----------------------------------------
        #                Global Definitions
        #-----------------------------------------
        #                    UI Required
        global OriginalFilename
        OriginalFilename = StringVar()
        global OriginalColorCount
        OriginalColorCount = StringVar()
        global OriginalSize
        OriginalSize = StringVar()
```

```
        global ComboStitch
        ComboStitch = IntVar()
        global ComboSize
        ComboSize = StringVar()
        global FabricWidth
        FabricWidth = DoubleVar()
        global FabricHeight
        FabricHeight = DoubleVar()
        global MaxColors
        MaxColors = IntVar()
        global BorderSize
        BorderSize = DoubleVar()
```

```
global ProcessedColors
ProcessedColors = StringVar()
global ProcessedSize
ProcessedSize = StringVar()
global DmcColor
DmcColor = StringVar()
```

Az utolsó 'változó osztály'-ba tartozó változók tájékoztató jelleggel vannak használva miután az

eredeti képet a kívánt paraméterekké alakítottuk.

A global-ok következő csoportja a programból való könnyebb elérések kedvéért van bevezetve (jobb felül). A legtöbbjük szerepe nyilvánvaló a nevükből következően, vagy az lesz, amint használni kezdjük őket. Van azonban három nemannyira-nyilvánvaló változó is. A backgroundColor1 és a backgroundColor2 mind Tuple, ami a rácsozás folyamatában van használva és a ReadyToProcess változó ami azt jelöli, hogy az eredeti kép be van töltve és minden kész az indulásra – csak abban az esetben ha a felhasználó túl korán nyomja meg a Process gombot.

Végül készen vagyunk az összes globálunkkal és most a kód következik ami ténylegesen létrehozza a GUI-t. Megnyitjuk az adatbázist, létrehozzuk a menüt, beállítjuk a widgeteket és végül a megfelelő helyre helyezzük őket. Az igazításhoz a Grid geometria menedzsert fogjuk használni. Visszatérünk még rá egy kicsit később.

```
self.OpenDB()
self.MakeMenu(master)
frm =
self.BuildWidgets(master)
self.PlaceWidgets(frm)
```

A kódunk következő része (jobb középen) a menüt fogja beállítani. Próbáltam logikus módon elrendezni, így könnyű lesz megérteni.

Definiálunk egy függvényt MakeMenu névvel és berakjuk a TopLevel ablakba. Majd definiáljuk a három menüt amit létre fogunk hozni. Egyet a File, egyet a Process és egyet a Help számára.

```
menu.add_cascade(label="File"
, menu=filemenu)
menu.add_cascade(label="Proce
ss",menu=process)
menu.add_cascade(label="Help"
,menu=help)
```

Most beállítjuk a File menü elemeket (jobb lent). Open a képünket fogja megnyitni és meghívja a „GetFileName” függvényt. A Save a kimeneti PDF fájlt fogja létrehozni és a FileSave függvényt hívja meg. Hozzáadunk egy elválasztót és végül egy Exit függvényt.

Majd a Process elem és a Help függvények következnek (következő oldal jobb felül).

A menüelemek mindegyike különböző gombokon keresztül is elérhető a programból.

Most megírjuk a BuildWidgets függvényünket. Ez az ahol létrehozzuk a GUI-en használt összes widgetet.

```
        #---------------------------------------
        global ShowGrid
        ShowGrid = True
        global ProcessedImage
        ProcessedImage = ""
        global GridImage
        GridImage = ""
        global backgroundColor1
        backgroundColor1 = (120,)*3
        global backgroundColor2
        backgroundColor2 = (0,)*3
        global ReadyToProcess
        ReadyToProcess = False
```

```
#=========================================================
#                   BEGIN UI DEFINITION
#=========================================================
    def MakeMenu(self,master):
            menu = Menu(master)
            root.config(menu=menu)
            filemenu = Menu(menu, tearoff=0)
            process = Menu(menu,tearoff=0)
            help = Menu(menu,tearoff=0)
```

```
#-----------------------------------------
    #                   File Menu
    #-----------------------------------------
    filemenu.add_command(label="New")
    filemenu.add_command(label="Open", command=self.GetFileName)
    filemenu.add_command(label="Save", command=self.FileSave)
    filemenu.add_separator()
    filemenu.add_command(label="Exit", command=self.DoExit)
```

```
def
BuildWidgets(self,master):

self.frame =
Frame(master,width=900,height
=850)
```

A függvény definicióval kezdünk (jobb lent), átadjuk neki a TopLevel

ablakot (master) és elhelyezünk egy frame-et ami az összes többi widgetünket tárolja. Kommentekkel láttam el ami segít értelmezni, hogy melyik kódrészlet melyik frame-re vonatkozik. Elsőnek a felső frame-et részletezzük.

Feltételezve, hogy emlékszel vagy felfrissítetted az emlékeidet a Tkinter-el kapcsolatban, ennek a résznek elég egyszerűnek kellene lennie. Nézzük az első label-t mint értelmezendő dolgot.

```
self.label1 =
Label(self.frm1,text =
„Original Filename: ”)
```

Elsőnek definiáljuk a widget nevét (self.label1 =). Utána beállítjuk, hogy milyen widget típussal akarjuk használni a változót; jelen esetben Label. Végül beállítjuk a paramétereket amiket alkalmazni akarunk a widgettel kapcsolatban, elsőnek a szülő widgetet (self.frm1), majd jelen esetben a szöveget ami megjelenik a labelben. Most egy pillanat erejéig nézzük meg a self.btnGetFN gombot.

```
self.btnGetFN =
Button(self.frm1, width=28,
image=self.openimage,command=
self.GetFileName)
```

Az első észreveendő dolog, hogy ez két sorba van törve. Te nyugodtan írhatod az egészet egy sorba... ez csak túl hosszú, hogy beleférjen egy 72 karakteres sorba. Egész közelről is szemügyre fogjuk venni a paramétereket, amiket itt használunk. Első a szülő (frm1), a következő a width ami 28-ra van beállítva. Amikor egy widgetet használunk aminek a paramétere egy szöveg vagy egy kép, akkor oda kell figyelnünk a width beállításánál. Ha ez egy szöveget fog tartalmazni, akkor a width paraméter a karakterek számát fogja tárolni. Ha ez egy képet jelenít meg, akkor a pixelek számára lesz beállítva. Végül beállítjuk a command paramétert ami megmondja a rendszernek, hogy a gomb megnyomásakor melyik függvényt hívja meg.

```
#-----------------------------------------
#                   Process Menu
#-----------------------------------------
process.add_command(label="All",command=self.Process)
#-----------------------------------------
#                    Help Menu
#-----------------------------------------
help.add_command(label="Help",command=self.ShowHelp)
help.add_separator()
help.add_command(label="About",command=self.ShowAbout)
```

Még egy dolgot meg kell nézni, mégpedig a textvariable paramétert. Ez azt mondja meg nekünk, hogy melyik változó fogja tárolni az információt, ami a widgetben lesz megjelenítve. Ezeket már beállítottuk korábban az __init__ függvényben. Van még egy dolog amit meg kell említeni, talán emlékszel, hogy magának a frame-nek két paramétere van. A Relief paraméter ami a

```
# ---------------Middle Frame --------------------
self.frm2 = Frame(self.frame,width=900,height=160,bd=4,relief=GROOVE)
self.lbl4 = Label(self.frm2,text="Aida Stitch Size: ")
self.lbl5 = Label(self.frm2,text="Aida Fabric Size: ")
self.TCombobox1 = ttk.Combobox(self.frm2,textvariable=ComboStitch,width=8)
self.TCombobox1.bind('<<ComboboxSelected>>', self.StitchSizeSelect)
self.TCombobox1['values'] = (7,10,11,12,14,16,18,22)
self.TCombobox2 = ttk.Combobox(self.frm2,textvariable=ComboSize,width = 8)
self.TCombobox2.bind('<<ComboboxSelected>>',self.AidaSizeSelect)
self.TCombobox2['values'] = ("12x18","15x18","30")
```

```
# ---------------- TOP FRAME --------------------
self.frm1 = Frame(self.frame,width=900,height=100,bd=4,relief=GROOVE)
self.label1 = Label(self.frm1,text = "Original Filename: ")
self.entFileName = Entry(self.frm1,width=50,textvariable=OriginalFilename)
self.btnGetFN = Button(self.frm1, width=28, image=self.openimage,
    command=self.GetFileName)
self.label2 = Label(self.frm1,text = "Original Colors: ")
self.lblOriginalColorCount = Label(self.frm1,text="",width=10,
    textvariable=OriginalColorCount)
self.label3 = Label(self.frm1,text = "Original Size: ")
self.lblOriginalSize = Label(self.frm1,text="",width=10,
    textvariable=OriginalSize)
```

frame szélének a típusát állítja be és ebben az esetben GROOVE és a bd paraméter, ami a szélességét állítja. A szélesség alapesetben 0, így ha látni akarod a hatását akkor be kell állítanod a szélességet (bd egy rövidítés).

Most a középső frame-mel fogunk foglalkozni.

Ezen szakasz utolsó hat sora (előző oldal jobb középen) az UI-ben lévő két lenyíló listát kezeli. Mindegyik lenyíló lista három sorból áll (így programoztam, hogy könnyebb legyen megértni). Az első sorban az alap paramétereket állítjuk be. A következő sorban összekötjük a lenyíló lista selection-changed eseményét a StitchSizeSelect függvénnyel és az utolsó sorban az értékek vannak felsorolva, amik a lenyitáskor lesznek elérhetők.

Minden más ezek előtt elég „normál” dolog. Most beállítjuk a widgetek számára a szükséges alapértelmezett értékeinket. Megint a globális változókat használjuk amit az __init__ függvényben állítottunk be és összekapcsoltuk a widget változó osztállyal.

```
ComboStitch.set(14)
ComboSize.set(„15x18”)
FabricWidth.set(15)
FabricHeight.set(18)
MaxColors.set(50)
BorderSize.set(1.0)
```

Most nézzük az alsó frame-et. Ez igazán egyszerű mert csak a frame-et és a két label-t kell beállítani amit a kép tárolására fogunk használni.

Végül az oldalsó frame-mel foglalkozunk. Az oldalsó frame egy ScrolledFrame-et fog tárolni a PMW library-ből. Ez egy barátságos felületet biztosít a használandó fonal információk számára és használni sem olyan nehéz. Tanulmányozd a ScrolledFrame-et egyedül, mert még nagyon sok dologról kell itt beszélnünk.

```
        self.lbl6 = Label(self.frm2,text="Max Colors: ")
        self.entMaxColors = Entry(self.frm2,textvariable=MaxColors,width=3)
        self.lbl7 = Label(self.frm2,text="Border Size: ")
        self.entBorderSize = Entry(self.frm2,textvariable=BorderSize,width = 8)
        self.frmLine = Frame(self.frm2,width=6,height=80,bd=3,relief="raised")
            self.lbl8 = Label(self.frm2,text="          Processed Image Colors: ")
        self.lbl9 = Label(self.frm2,text="Processed Image Stitch Count: ")
        self.lblProcessedColors = Label(self.frm2, width=10,textvariable=ProcessedColors,
             justify=LEFT)
        self.lblProcessedSize = Label(self.frm2, width=10, textvariable=ProcessedSize,
             justify=LEFT)
        self.btnDoIt = Button(self.frm2,text="Process",width=11,command = self.Process)
        self.btnShowGrid = Button(self.frm2,text="Hide Grid", width=11,
             command=self.ShowHideGrid)
        self.btnCreatePDF = Button(self.frm2, text="Create PDF", width=11,
             command=self.CreatePDF)
```

```
        # --------------- Bottom Frame ------------------
        self.frm3 = Frame(self.frame,width=450,height=450,bd=4,relief=GROOVE)
        self.lblImageL = Label(self.frm3, image=self.DefaultImage,
             height=400, width=400, borderwidth=2, relief=GROOVE)
        self.lblImageR = Label(self.frm3, image=self.DefaultImage, height=400,
             width=400,borderwidth=2, relief=GROOVE)
```

```
        #---------------- Side Frame ------------------
        self.frm4 = Frame(self.frame,width = 300,height=580,bd=4,relief=GROOVE)
        # Create the ScrolledFrame.
        self.sf = Pmw.ScrolledFrame(self.frm4,
            labelpos = 'n', label_text = 'Processed Color List',
            usehullsize = 1,
            hull_width = 300,
            hull_height = 567,)
        return self.frame
```

Ennyit a widgetekkel kapcsolatban. Most el kell őket helyeznünk. Ahogy korábban említettem inkább a Grid geometria menedzsert fogjuk használni mint az absolute vagy a pack menedzsert.

A grid metódus belehelyezi a widgeteket egy rácsba amelyre sor és oszlop értékekkel lehet hivatkozni. Példaként lássuk a felső frame-et (jobb felül).

Elsőnek elhelyezzük a frame-et.

Láthatod, hogy a widgeteket a {widgetnev}.grid parancs a sor és az oszlop poziciók használatával helyezzük el. Mint látod, az entry widgetnek megmondjuk, hogy 5 oszlopot foglaljon el. A padx és pady értékek egy extra távolságot adnak a jobb és bal oldalhoz (padx) vagy a felső és alsó részhez (pady). A sticky egy hasonló paraméter mint a text-nél a justify parancs.

A középső frame egy kicsit bonyolultabb de alapvetően ugyanaz mint a felső frame. A kód közepén egy extra frame-et is észrevehetsz (self.frmLine). Ez egy elválasztót biztosít számunkra a beállítások rész és a megjelenítő rész között. Mivel nincs vizszintes vagy függőleges vonal widget, ezért csaltam egy kicsit és egy frame-et használtam 6 pixel szélességben és 3 pixelnyi széllel, hogy úgy nézzen ki mint egy vastag vonal.

```
ROW | Col 0  | Col 1 - Col 6          |Col 7   | Col 9 | Col 10                  |
    ------------------------------------------------------------------------------
 0  | Label1 | entFileName            |btnGenFN| Label2|lblOriginalColorCount    |
 1  |                                 |        | Label3|lblOriginalSize          |
    ------------------------------------------------------------------------------
```

```
    def PlaceWidgets(self,frame):
        frame.grid(column = 0, row = 0)
        # ---------------- TOP FRAME ---------------------
        self.frm1.grid(column=0,row=0,rowspan=2,sticky="new")
        self.label1.grid(column=0,row=0,sticky='w')
        self.entFileName.grid(column=1,row=0,sticky='w',columnspan = 5)
        self.btnGetFN.grid(column=7,row = 0,sticky='w')
        self.label2.grid(column=9,row=0,sticky='w',padx=10)
        self.lblOriginalColorCount.grid(column=10,row=0,sticky='w')
        self.label3.grid(column=9,row=1,sticky='w',padx=10,pady=5)
        self.lblOriginalSize.grid(column=10,row=1,sticky='w')
```

```
        # ---------------- MIDDLE FRAME ---------------------
        self.frm2.grid(column=0,row=2,rowspan=2,sticky="new")
        self.lbl4.grid(column=0,row=0,sticky="new",pady=5)
        self.lbl5.grid(column=0,row=1,sticky="new")
        self.TCombobox1.grid(column=1,row=0,sticky="new",pady=5)
        self.TCombobox2.grid(column=1,row=1,sticky="new")
        self.lbl6.grid(column=2,row = 0,sticky="new",padx=5,pady=5)
        self.entMaxColors.grid(column=3,row=0,sticky="new",pady=5)
        self.lbl7.grid(column=2,row=1,sticky='new',padx=5)
        self.entBorderSize.grid(column=3,row=1,sticky='new')
        self.frmLine.grid(column=4,row=0,rowspan=2,sticky='new',padx=15)
        self.lbl8.grid(column=5,row=0,sticky='new',pady=5)
        self.lbl9.grid(column=5,row=1,sticky='new')
        self.lblProcessedColors.grid(column=6,row=0,sticky='w')
        self.lblProcessedSize.grid(column=6,row=1,sticky='new')
        self.btnDoIt.grid(column=7,row=0,sticky='e',padx=5,pady = 5)
        self.btnShowGrid.grid(column=7,row=1,sticky='e',padx=5,pady = 5)
        self.btnCreatePDF.grid(column=8,row=0,rowspan=2,sticky='ew',padx=10)
```

```
        # ---------------- BOTTOM FRAME ---------------------
        self.frm3.grid(column=0,row=4,sticky="nsew")
        self.lblImageL.grid(column=0,row=0,sticky="w")
        self.lblImageR.grid(column=1,row=0,sticky="e")
```

Az alsó frame egyszerű, mert csak a frame-ünk és a két label-ünk van, amikben a képeket tároljuk.

Az oldalsó frame majdnem ugyanígy néz ki, kivéve, hogy a ScrolledFrame lehetővé teszi egy frame-nek, hogy az hozzá legyen rendelve a ScrolledFrame widget belsejéhez. Itt létrehozunk három widgetet és elhelyezzük őket a rácsba mint oszlop fejléceket. De csak azután, miután itt hozzárendeltük a belső frame-et a görgető frame-hez, majd hozzá kell rendelnünk a szülőt (self.sfFrame) a widgetekhez, miután létrehoztuk azt.

Ennyi volt a kemény munka mára. Ennél a pontnál létre fogjuk hozni az összes függvényt ami ahhoz kell, hogy a GUI működjön de a legtöbbnek üresen hagyjuk a függvénytörzsét a következő hónapig. Van egy kevés amit mégis befejezünk, de ezek elég rövidek.

Az első függvény az Exit elem lesz a menüből. Ez a File menü alatt található.

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

Az egyetlen másik a Thumbnail függvény. Ez ahhoz kell, hogy az alsó frame-ben betöltsük a szürke téglalapokat a label-ekbe. Átadjuk a fájlnevet és azt a szélességet és magasságot amekkorának az előnézet képet kívánjuk látni.

Mert ez a cikk ilyen hosszúra nyúlt, adni fogok egy függvénynevekből álló listát és annyit kell csinálnod, hogy a pass parancsot írod a függvények törzsébe. A kódokat majd a következő hónapban fogjuk megírni. Példaként meg fogom adni az elsőt, de már tudni kellene, hogy hogyan csináld.

```
def GetFileName(self) :
    pass
```

A maradék függvényeknél csak a def sorokat adom meg. Körültekintően szúrd be mindegyiket a kódodba.

Láthatod, hogy nagy mennyiségű munka vár még ránk a következő hónapban. Van ám még négy sorunk amit meg kell írni és befejeztük erre a hónapra. Ez az osztály kódból való.

```
# ---------------- SIDE FRAME ---------------------
self.frm4.grid(column=2,row=0,rowspan=12,sticky="new")
self.sf.grid(column=0,row=1)
self.sfFrame = self.sf.interior()
self.lblch1 = Label(self.sfFrame,text="        Original")
self.lblch2 = Label(self.sfFrame,text="        DMC")
self.lblch3 = Label(self.sfFrame,text="Name/Number")
self.lblch1.grid(column=0,row=0,sticky='w')
self.lblch2.grid(column=1,row=0,sticky='w')
self.lblch3.grid(column=2,row=0,sticky="w")
```

```
    def Thumbnail(self,file,hsize,wsize):
        size = hsize,wsize
        extpos = file.rfind(".")
        outfile = file[:extpos] + ".thumbnail"
        im = Image.open(file)
        im.thumbnail(size)
        im.save(outfile,"JPEG")
        return im
```

```
root = Tk()
root.title(„Cross Stitch Pattern Creator”)
test = XStitch(root)
root.mainloop()
```

Az első sor a root TopLevel ablakot állítja be. A következő sor pedig a címsort. A harmadik sor példányosítja az XStitch osztályunkat és az utolsó sor elindítja a fő ciklust ami megjeleníti az UI-t és átadja neki a vezérlést.

Nos nem volt kevés ez erre a hónapra, de végül a végére értünk. Futtathatod is akár a programot, hogy lásd a GUI-t.

Mint mindig, a kód elérhető a Pastebin-en a http://pastebin.com/XtBawJps címen.

A következő hónapban meg fogjuk írni a kódot. Találkozunk akkor.

```
    def ShowHelp(self):, def ShowAbout(self):, def OpenDB(self):, def ShowHideGrid(self):
    def StitchSizeSelect(self,p):, def AidaSizeSelect(self,p):, def Process(self):
    def CreatePDF(self):, def OriginalInfo(self,file):, def GetColorCount(self,file):
    def GetHW(self,file):, def GetHW2(self,file):, def GetColors(self,image):
    def Pixelate(self,im,pixelSize):, def ReduceColours(self,ImageName):
    def MakeLines(self,im,pixelSize):, def MakeLines2(self,im,pixelSize):
    def Rgb2Hex(self,rgb):, def FillScrolledList(self,filename):
    def GetBestDistance(self,r1,g1,b1):
```

Programozzunk Pythonban – 56. rész

Írta: Greg D. Walters
Fordította: Sipos Zoltán

Egy hímzésmint-generátoron munkálkodunk. A múlt hónapban megcsináltuk az UI részt és most elérkezett az idő, hogy megírjuk a kódot, ami a feladat nagyobb részéért felelős. A következő hónapban a PDF fájl kimeneti részén kezdünk majd dolgozni.

Elsőnek a menüelemekkel fogunk foglalkozni. A kód lent látható.

A globális ReadyToProcess változó biztosítja, hogy ha a felhasználó megnyomja a Process gombot akkor a rendszer nem próbál meg feldolgozni semmit ha nincs mit. A tkFileDialog askopenfilename beépített dialóg rutinját használjuk, hogy megkapjuk a fájl nevét a kiinduló képnek. Aztán a kiinduló képben lekérjük a színek számát valamint a szélességet és a magasságot. Ezeket az értékeket elmentjük és megjelenítjük őket a GUI-ban. Aztán megnyitjuk a képet és létrehozunk egy előnézeti képet, amit megjelenítünk az alsó keret bal oldali részében. Lásd a szövegdobozt a jobb oldalon.

Következőnek megírjuk a ShowHideGrid függvényt. Ez egyszerűen felcserél két képet a jobb oldali képben a ShowGrid globális változótól függően. Ha False akkor a megjelenítő/elrejtő gombon kicseréljük a szöveget, majd a ShowGrid változót true-ra állítjuk és beállítjuk a képet arra amelyik tartalmazza a rácsozást. Amúgy pedig a megjelenítő/elrejtő gombon „Show Grid”-re cseréljük a szöveget, False-ra állítjuk a ShowGrid változót és a rácsozatlan képet töltjük be. A kód a következő oldalon található (balra fent).

A StichSizeSelect függvény akkor hívódik meg, mikor megváltozik az öltésméret lenyíló lista értéke. Lekérdezzük az értéket a lenyíló listából és hozzárendeljük egy lokális változóhoz.

```
        OriginalFilename.set(fileName)
        OriginalColorCount.set(self.GetColorCount(fileName))
        OriginalSize.set(self.GetHW(fileName))
        masterimage=Image.open(fileName)
        masterimage.thumbnail((400,400))
        self.img = ImageTk.PhotoImage(masterimage)
        self.lblImageL['image'] = self.img
        ReadyToProcess = True
```

A FileSave menüelem egyszerűen meg fogja hívni a CreatePDF rutint amikor az elkészül.

```
    def FileSave(self):
        self.CreatePDF()
```

A ShowHelp és a ShowAbout routinokat most nem fejezzük be csak egy dialóg közli, ezek az opciók még nem elérhetők.

```
    def ShowHelp(self):
        tkMessageBox.showinfo(title="Help",message='Sorry,
but help is not yet available.')

    def ShowAbout(self):
        tkMessageBox.showinfo(title="About",message='Sorry,
but the About function is not yet available.')
```

Az OpenDB rutint már jóval ez előtt megírtuk, így tudnod kellene, hogy ez mit csinál.

```
    def OpenDB(self):
        global connection
        global cursor
        #---------------------------------
        connection = apsw.Connection("floss.db3")
        cursor = connection.cursor()
```

```
    def GetFileName(self):
        global ReadyToProcess
        #---------------------------------
        fileName = tkFileDialog.askopenfilename(parent=root,filetypes=self.picFormats ,title="Select File to open...")
```

```
    def ShowHideGrid(self):
        global ShowGrid
        #--------------------------------
        if ShowGrid == False:
            self.btnShowGrid['text'] = 'Hide Grid'
            ShowGrid = True
            self.im2=Image.open(self.GridImage)
            self.im2.thumbnail((400,400))
            self.img3 = ImageTk.PhotoImage(self.im2)
            self.lblImageR['image'] = self.img3
        else:
            self.btnShowGrid['text'] = 'Show Grid'
            ShowGrid = False
            self.im2=Image.open(self.ProcessedImage)
            self.im2.thumbnail((400,400))
            self.img3 = ImageTk.PhotoImage(self.im2)
            self.lblImageR['image'] = self.img3
```

```
def StitchSizeSelect(self,p):

selection = ComboStitch.get()
```

Az AidaSizeSelect függvény (jobbra fent) nagyon hasonló a StitchSizeSelect függvényhez. A lenyíló listában a kiválasztott értéktől függően beállítjuk a FabricWidth és a FabricHeight globálokat. Ha a 30-at választják ki akkor a 30x30 alapértelmezett értéket állítjuk be.

Van egy ReadyToProcess nevű változónk (lent) arra az esetre, ha a felhasználó futtatni próbálja a process függvényt mielőtt a kép be lenne töltve.

A kiinduló fájlból egy 5x5 pixel méretű mátrixot hozunk létre. Ez lehetővé teszi számunkra, hogy az 5x5-ös mátrixot egyetlen színné csoportosítsuk át. Aztán csökkentjük a színeket, lekérjük a feldolgozott kép szélességét és magasságát és beállítjuk a méretet, így a felhasználó láthatja, hogy milyen nagy lesz a keletkező kép.

```
    def AidaSizeSelect(self,p):
        selection = ComboSize.get()
        if selection != "30":
            pos = selection.find("x")
            width = int(selection[:pos])
            height=int(selection[pos+1:])
        else:
            width = 30
            height = 30
        FabricWidth.set(width)
        FabricHeight.set(height)
```

```
# Place image

self.im2=Image.open(Reduced)

self.im2.thumbnail((400,400))

self.img3 = ImageTk.PhotoImage(self.im2)

self.lblImageR['image'] = self.img3

self.ProcessedImage = 'im1.png'
```

A fenti kódrészlet belehelyezi a feldolgozott képet abba a képbe amelyik a módosított képet fogja tárolni. A következő kódrészlet létre fogja hozni a rácsozást így a felhasználónak meglesz a rácsa a keresztöltéshez.

```
self.MakeLines(Reduced,5)

self.MakeLines2('output.png',50)

self.im2 = Image.open('output2.png')

self.im2.thumbnail((400,400))

self.img3 = ImageTk.PhotoImage(self.im2)
```

```
    def Process(self):
        global ReadyToProcess
        #--------------------------------
        if ReadyToProcess == False:
            tkMessageBox.showinfo(title="ERROR...",message='You must load an original imaage first.')
        else:
            newimage = self.Pixelate(OriginalFilename.get(),5)
            Reduced = self.ReduceColors(newimage)
            W,H = self.GetHW2(Reduced)
            siz = "{0}x{1}".format(W/5,H/5)
            ProcessedSize.set(siz)
```

```
self.lblImageR['image'] =
self.img3

self.FillScrolledList('out-
put.png')

self.GridImage = 'out-
put2.png'
```

A CreatePDF függvényt kihagyjuk addig, amíg meg nem írjuk a PDF függvényt a következő hónapban.

```
def CreatePDF(self):

tkMessageBox.showinfo(title="
Create PDF",message='Sorry,
but the Create PDF function
is not yet available.')
```

Az OriginalInfo() rutin lekérdezi és beállítja a változókat az eredeti kép formátumától, méretétől és módjától függően.

```
def OriginalInfo(self,file):
  im = Image.open(file)
  imFormat = im.format
  imSize = im.size
  imMode = im.mode

  self.size = imSize
  self.imformat = imFormat
  self.immode = imMode
```

A GetColorCount függvény a .getcolors metódust használja, hogy lekérdezze a színek számát a képfájlban. Maxcolors paraméterként 1600000 kell használnunk mert ha a kép több mint 256 színt tartalmaz (vagy bármi más van a paraméterben) a metódus 'None'-al tér vissza. Ez a függvény hasonló a GetColors függvényhez kivéve, hogy a GetColors egy már megnyitott képfájlon dolgozik. Ha a GetColorCount-ot használod akkor egy megnyitatlan fájlt kell átadnod.

```
def GetColorCount(self,file):
  im = Image.open(file)
  numColors =
im.getcolors(1600000)
  self.colors =
len(numColors)
  return self.colors
```

A következő két függvény a képfájl magasságát és szélességét adja vissza pixelekben. A különbség a kettő között, hogy a GetHW egy sztringet ad vissza, például 1024x768, a GetHW2 pedig két egész számot.

```
def GetHW(self,file):
  im = Image.open(file)
  tmp =
"{0}x{1}".format(im.size[0],i
m.size[1])
  return tmp

def GetHW2(self,file):
  im = Image.open(file)
  return
im.size[0],im.size[1]
```

A GetColors lekérdezi a színek számát az átadott képfájlban. Paraméterként 1.6 millió színt használunk mert 256-nál több szín esetén az image.getcolors() rutin alapértelmezett értéke 0.

```
def GetColors(self,image):
  numColors =
image.getcolors(1600000)
  colors = len(numColors)
```

A Pixelate függvény (fent) két paramétert fogad, a képfájl nevét (im) és a kívánt pixelméretet. A feladatot az image.resize metódus hajtja végre. Ezt a rutint egy csomó helyen megtaláltam a weben. Ebben a példában 5 pixel méretet

```
def Pixelate(self,im,pixelSize):
    image = Image.open(im)
    self.GetColors(image)
    image = image.resize((image.size[0]/pixelSize, image.size[1]/pixelSize), Image.NEAREST)
    image = image.resize((image.size[0]*pixelSize, image.size[1]*pixelSize), Image.NEAREST)
    self.GetColors(image)
    #image.show()
    image.save('newimage.png')
    return 'newimage.png'
```

```
def ReduceColors(self,ImageName):
    #Reduce colors
    numcolors=MaxColors.get()
    image = Image.open(ImageName)
    output = image.convert('P', palette=Image.ADAPTIVE, colors=numcolors)
    x = output.convert("RGB")
    self.GetColors(x)
    numcolors = x.getcolors()
    ProcessedColors.set(len(numcolors))
    x.save('im1.png')
    return 'im1.png'
```

fogunk átadni, ami jól működik hímzőprojektek esetén. Azt is megmondjuk még a metódusnak, hogy a legközelebbi szomszéd színét is vegye figyelembe. Ez egy új képpel tér vissza, amit fájlba mentünk és visszaadjuk a fájl nevét.

A ReduceColors rutin (előző oldal lent) alapvetően az Image.ADAPTIVE palettát használja ezáltal egy sokkal kevesebb számú színt kaphatunk.

Van két MakeLines rutin (jobbra fent). Ezek létrehozzák a rácsozatot amiről korábban beszéltünk.

Az Rgb2Hex() visszaadja az átadott RGB érték hexa értékét. Ezt arra használjuk majd, hogy megpróbáljuk összehasonlítani az adatbázisban lévő színeket a képben lévő színekkel.

```
def Rgb2Hex(self,rgb):
  return '#%02x%02x%02x' %
rgb
```

A jobb oldalon a ScrollList (lent) azokat a színeket tárolja, amelyek arra lesznek használva, hogy megkapjuk a megfelelő fonalszíneket. Egyszerűen címkéket hozunk létre a színek (vizuálisan) és a szöveg tárolásához.

```
    def MakeLines(self,im,pixelSize):
        global backgroundColor1
        #--------------------------------
        image = Image.open(im)
        pixel = image.load()
        for i in range(0,image.size[0],pixelSize):
            for j in range(0,image.size[1],pixelSize):
                for r in range(pixelSize):
                    pixel[i+r,j] = backgroundColor1
                    pixel[i,j+r] = backgroundColor1
        image.save('output.png')

    def MakeLines2(self,im,pixelSize):
        global backgroundColor2
        #--------------------------------
        image = Image.open(im)
        pixel = image.load()
        for i in range(0,image.size[0],pixelSize):
            for j in range(0,image.size[1],pixelSize):
                for r in range(pixelSize):
                    try:
                        pixel[i+r,j] = backgroundColor2
                        pixel[i,j+r] = backgroundColor2
                    except:
                        pass
        image.save('output2.png')
```

```
    def FillScrolledList(self,filename):
        im = Image.open(filename)
        numColors = im.getcolors()
        colors = len(numColors)
        cntr = 1
        for c in numColors:
            hexcolor = self.Rgb2Hex(c[1])
            lblColor=Label(self.sfFrame,text="              ",bg=hexcolor,relief=GROOVE)
            lblColor.grid(row = cntr, column = 0, sticky = 'nsew',padx=10,pady=5)
            pkID = self.GetBestDistance(c[1][0],c[1][1],c[1][2])
            sql = "SELECT * FROM DMC WHERE pkID = {0}".format(pkID)
            rset = cursor.execute(sql)
            for r in rset:
                hexcolor2 = r[6]
                dmcnum = r[1]
                colorname = r[2]
            lblColor2=Label(self.sfFrame,text="              ",bg="#" + hexcolor2,relief=GROOVE)
            lblColor2.grid(row = cntr,column = 1,sticky = 'w',padx=5,pady=5)
            lblColor3=Label(self.sfFrame,text = str(dmcnum) + "-" + colorname,justify=LEFT)
            DmcColor.set(dmcnum)
            lblColor3.grid(row = cntr, column = 2,sticky = "w",padx=1,pady=5)
            cntr += 1
```

Ez a rutin (következő oldalon) amit arra használunk, hogy megpróbáljuk megtalálni a legkisebb eltérést a képben lévő szín és az adatbázisban lévő szín között. A weben sok különböző algoritmus található amit megnézhetsz és megpróbálhatod megérteni a logikát mögötte. Elég bonyolult.

Ok. Ennyit erre a hónapra. Következő alkalommal elkezdjük létrehozni a PDF kimeneti fájlt így a hímzőnek lesz valamije amivel dolgozhat.

Mint mindig, a kód elérhető a PasteBinen a http://pastebin.com/DmQ1GeUx címen. Folytatni fogjuk talán a következő hónapban. Néhány műtét vár rám, ezért nem tudom pontosan milyen hamar leszek képes valameddig is ülni egyáltalán. Addig is jó szórakozást.

```
    def GetBestDistance(self,r1,g1,b1):
        # dist = math.sqrt(((r1-r2)**2) + ((g1-g2)**2) + ((b1-b2)**2))
        sql = "SELECT * FROM DMC"
        rset = cursor.execute(sql)
        BestDist = 10000.0
        for r in rset:
            pkID = r[0]
            r2 = r[3]
            g2 = r[4]
            b2 = r[5]
            dist = math.sqrt(((r1-r2)**2) + ((g1-g2)**2) + ((b1-b2)**2))
            if dist < BestDist:
                BestDist = dist
                BestpkID = pkID
        return BestpkID
```

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

Keresztszemes mintagenerátor – 4. rész – a pyfPDF megértése

Elnézést a pár hónapos kimaradásért, de még mindig nem tudok tartósan egy helyben maradni. Úgyhogy ez a cikk is rövidebb lesz, mint amihez általában szoktál. Az eredeti tervem az volt, hogy egyből a PDF generálás témájára ugrok, de olyan sok mindent kell hozzá megérteni ebből a könyvtárból, hogy úgy döntöttem, ezt a részt arra fogom használni, hogy egy bevezetést adjak a pyfPDF-ről. Így aztán a következő alkalommal repülünk rá a PDF generálás témájára. Úgyhogy kezdjük is.

FPDF az ingyenes PDF-et ("Free PDF") jelenti. Egy nagyon minimális példa erre a következő kód:

```
from fpdf import FPDF

pdf = FPDF()

pdf.add_page()

pdf.set_font('Arial','B',16)

pdf.cell(40,10,'Hello From Python')

pdf.output('example1.pdf','F')
```

Az első sor beimportálja a könyvtárfájlt. A következő létrehoz egy FPDF objektumpéldányt. Itt az alapértelmezett értékeket fogjuk használni, amelyek a következők:
• Tájolás: Álló
• Mértékegység: milliméter
• Formátum: A4

Ha a „US" szabványra lenne szükségünk, akkor így adhatnánk meg:

```
pdf=FPDF('P','in','Letter)
```

Ügyeljünk a paraméterekre! FPDF(orientation, units, format):
• A tájolás lehetséges értékei: „P", az álló („Portrait"); „L" fekvő („Landscape")
• A mértékegységnél érvényes értékek: „pt" (képpont), „mm" (milliméter), „cm" (centiméter), „in" (hüvelyk)
• A formátumnál érvényes értékek: A3, A4, A5, Letter, Legal, vagy konkrét formátumméret a megadott mértékegységgel megadott hosszúság- és szélességadatokkal.

A harmadik sor létrehozza az oldalt, amibe az adatokat visszük be. Megjegyzendő, hogy az oldal nem automatikusan jön létre, amikor az objektumpéldányt létrehozzuk. Az oldal kiindulópontja a bal felső sarok. Az aktuális pozició alapértelmezetten 1 cm a margótól számítva. A margó változtatható a SetMargins függvénnyel.

Mielőtt bármilyen konkrét adatot tudnánk írni a dokumentumba, meg kell hívnunk a pdf.set_font() függvényt, amivel megadjuk a betűtípust. A felette levő sorban definiáltuk a formátumot (16 pontos félkövér Arial). Az érvényes szabványos betűtípusok az Arial, Times, Courier, Symbol és ZapfDingbats.

Most tehetünk bele cellákat a pdf.cell() függvény segítségével. A cella egy téglalap alakú terület, lehetőleg kerettel, mely szöveget tartalmaz. Oda teszi be a cellát, ahol az aktuális pozició van, amit a fenti példában meg is adunk (40, 10 cm). A paraméterek:

```
pdf.cell(Width, Height, text, border, line, align, fill, link)
```

Ahol:
• Width (szélesség): a cella szélességét jelenti. Ha 0, akkor a cella kiér a jobb oldali margóig.
• Height (magasság): a cella magasságát jelöli.
• Text: a szöveg, amit bele akarunk írni.
• Border (szegély): vagy 0 (azt jelenti nincs szegély – ez az alapértelmezett), 1 jelenti, hogy van keret, vagy a következő betűk valamelyike, vagy összessége: „L", „T", „B", „R"
• Line: ahova az aktuális pozició kerüljön, miután a szöveget kiírta. Az értékek: 0 (jobbra igazít), 1 (a következő sor elejére), 2 (alulra). Alapértelmezett érték 0, az 1 pedig egyenértékű azzal, ha 0 és rögtön utána az ln() függvényhívás.
• Align: segít középre, vagy valamelyik oldalra igazítani a cellán belül. Megadható értékek: „L" (bal), „C" (közép), „R" (jobb)
• Fill: kitölti a hátteret színnel (true), vagy áttetsző (false). Alapértelmezett érték: false.
• Link: egy URL vagy azonosító, melyet az addlink() függvény ad vissza.

Végezetül a dokumentumot lezárja, és fájlba írja az „Output”-tal. A paraméterek a következők: fdpdf.output(név, célmappa). Ha nincs megadva fájlnév, akkor az eredményt a böngészőnek küldi. A célhely értékei „I" (inline, azaz böngészőbe), „F" (lokális fájlnévvel megadva), „D" (a böngészőbe, de fájlletöltés kényszerítésével azzal a névvel, ami meg lett adva), és „S" (a dokumentumot sztring formájában adja vissza).

Mivel a keresztszemes képünket pdf fájlba küldjük, meg kell értsük a képfüggvényt.

A függvényt a következőképpen hívjuk meg:

```
pdf.image(name,x=None,y=None,
w=0,h=0,type="",link="")
```

Ez a függvény iileszti be a képet. A méretet, amit a kép fog elfoglalni az oldalon a következő módokon lehet megadni:
• Explicit: szélesség és magasság megadásával vagy
• Explicit dimenzió megadásával.

A támogatott formátumok JPEG, PNG, GIF. Ha GIF-et akarunk használni, akkor a GD kiegészítőt kell használnunk.

A JPEG minden formája engedélyezett:
• szürkeárnyalat
• true color (24 bit színmélység)
• CMYK (32 bit)

A PNG-nél a következők engedélyezettek:
• szürkeárnyalat legfeljebb 8 biten (256 szint)
• indexelt színek
• true color (24 bit színmélység)

Megjegyzés: interlacing nem engedélyezett, és az 1.7-nél korábbi FPDF-ben nem támogatott az alfa csatorna.

A következő példát a pyFPDF oktatóanyagából (jobbra) emeltem ki.

Most már eleget láttunk, hogy képesek legyünk nagyjából megérteni, mi történik a kódban. De ebben a példában nekünk igazából a negyedik sor a legérdekesebb:

```
this.image('img1.png',10,8,33
)
```

Ebben a példányban a képfüggvényt fájlnévvel hívjuk meg. Az x pozició azt a helyet jelöli, hogy az oldalon hova fog kerülni a kép, az y pozició pedig azt jelenti, hogy milyen széles lesz a kép.

```
from fpdf import FPDF

class PDF(FPDF):
    def header(this):
        # Logo - replace with a small png of your own
        this.image('img1.png',10,8,33)
        # Arial bold 15
        this.set_font('Arial','B',15)
        # Move to the right
        this.cell(80)
        # Title
        this.cell(30,10,'Title',1,0,'C')
        # Line break
        this.ln(20)

# Instantiation of inherited class
pdf=PDF()
pdf.alias_nb_pages()
pdf.add_page()
pdf.set_font('Times','',12)
for i in range(1,41):
    pdf.cell(0,10,'Printing line number '+str(i),0,1)
pdf.output('example2.pdf','F')
```

Most, hogy már van egy hozzávetőleges rálátásunk a könyvtárra, a következő alkalommal hozzákezdünk a PDF kódolásához.

Addig is minden jót kívánok a következő hónaphoz. Viszlát legközelebb!

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

Programozzunk Pythonban – 58. rész

Írta Greg D. Walters – Fordította: Meskó Balázs

Először is, hadd köszönjem meg minden olvasónak, aki e-mailt küldött, hogy reméli a mielőbbi felépülésemet. Nagyon kedvesek voltak és segítettek, szeretném megköszönni Ronnie-nak is, a csodás szerkesztőnknek, a támogatásáért és a türelmét ebben a fájdalmas időszakban. Még mindig gondjaim vannak a hosszan tartó üléssel, így ez több nap alatt készül el, remélem sikerül megtartani a folyamatosságot.

Most pedig folytatódjon „a műsor”…

Nem túl rég, épp a blokkolóórához sétáltam, amikor a vezérigazgató behívott az irodájába. Remélve, hogy ez csak egy „hogy megy” beszélgetés lesz, bementem és leültem. A találkozót azzal kezdte, hogy „van egy problémám a táblázatkezelőmmel, és remélem tudsz segíteni”.

A látásom elsötétedett és a jól ismert háromhangos zenekari hangok átfutottak az agyamon: „Da Da DAAAAAAAAA”, amit mind jól ismerünk a 70-es és 80-as évek horrorfilmjeiből, ahelyett hogy sikítva elfutottam volna a szobából, ártatlanul megkérdeztem, hogy mi a baj. Azzal válaszolt, hogy valami gond van az egyik makróval és „egyszerűen csak kilép számolás közben”. Előhúztam a fehér cowboy kalapomat, és azt mondtam a legjobb hőshangomon: „Ne aggódj polgártárs, semmi perc alatt megoldjuk”. Rövidesen felfedeztem az okot, hogy miért omlott össze minden előjel nélkül. Egy cellában, a 35 munkafüzet egyikén, volt egy „nullával osztás”-hiba, mert egy várt érték nem volt megadva egy másik munkafüzetben. Ez nem a főnököm hibája volt, ő egy egyszerű módot kért magasabb értékek kinyerésére az adatokból. (Az előző mondatnak semmi köze ahhoz, hogy a főnököm olvashatja ezt a cikket! Vagy lehet, hogy mégis.)

Ahogy visszasétáltam a helyemre, lesöpörve a hamis számítógépi kódot a fehér kalapomról, rájöttem, hogy ez egy nagyszerű tanítási pillanat lehet. Így, itt vagyunk. De először ugorjunk vissza 1979-be, amikor az Apple bemutatta a Visicalcot. Az volt az első „szabad űrlapos számolórendszer”, amely nagyot durrant a piacon. Ugyan nagyon sok hiba volt a szoftverben, a világ imádta az ötletet, és megjelentek a klónok (és velük a hibák) más rendszereken is, mint például a Commodore Peten és más Apple-versenytársaknál (beleértve a Microsoftot, a Multiplan nevű programmal). Végül, 1983-ban, a Lotus Development Corp. nevű cég bemutatta a Lotus 1-2-3-at. Ugyan nagyon közel állt néhány aspektusában a Visicalchoz, belértve a menüszerkezetet, teljesen újra lett írva x86 Assembly nyelven, amely nagyon gyorssá tette, és a Visicalc sok hibája javításra került. A Lotus 1-2-3 olyan népszerű lett, hogy a „PC kompatibilitás” egy közismert mutatója volt.

A szabad űrlapos számolórendszerek eljövetele elérhetővé tette az „egyszerű” embereknek, hogy oly módon kezeljék a számokat, amely addig csak a programozók világában volt lehetséges. Szinte bárki, néhány óra alatt képes lett számokat értelmezni, diagramokat és grafikonokat készíteni, és megosztani az információkat a kollégáival. Nem sokkal ezután, a táblázatok egyes részeinek makrókkal és Basic-szerű beágyazott nyelvekkel történő automatizálása még több hatalmat adott a nem-programozó felhasználók kezébe. Megkaphatták a válaszokat saját maguk, valamint szép diagramokat és grafikonokat készíthettek, anélkül, hogy IT segítségre kelljen várniuk. De ahogy Peter Parker Ben bácsikájától megtanultuk…

A nagy hatalom nagy felelősséggel jár.

Hamarosan a táblazatok olyan helyekre is bekerültek, amelyek jobban illettek az adatbázisokhoz. Munkafüzeteink voltak egymáson, amelyek más munkfüzetektől függtek, és ha egy kis szám nem frissült útközben… nos, a régi „kártyavár”-hatás.

Ugyan nem hiszem, hogy minden táblázat gonosz, van jó néhány (olvasd: „sok”) amelyeket évekkel ezelőtt adatbázisokra kellett volna cserélni. Túl nagyra és ormótlanra nőttek. Ha valaki leülne a progra-

mozókkal és azt mondaná: „Kérlek segíts”, akkor a világ egy kedvesebb hely lenne.

Most, hogy abbahagytam a szónoklatomat, elértünk az e havi cikk valódi okához. Minden jó Python programozónak meg kell tudnia bírkóznia a táblázatokkal az eszközarzenáljával. Sosem tudhatja az ember, hogy mikor hívnak, hogy szedj ki adatokat egy táblázatból és módosítsd azt. Ugyan sok módja van az adatok kinyerésének táblázatból, például CSV fájlok használata, amelynek megvan a maga hátránya, néha szükség van arra, hogy közvetlenül olvass és írj egy „élő” táblázatot. Szétnézés után, megmaradtam egy nagyon szép könyvtárnál, hogy hozzáférjek a főnököm problémás táblázatához.

Hozzáadjuk az XLRD nevű könyvtárat, amely, ahogy gondolná az ember, azt jelenti, hogy eXceL ReaD. A könyvtár lehetővé teszi az Excel fájlokból (.xls, .xlsx és .xlsm) történő adatolvasást a 2.0-s verziótól kezdve.

Hozzunk létre egy Excel táblázatot, amelyet a XLRD funkcióinak megtekintésére használunk. Nyisd meg az Excelt, OpenOffice vagy LibreOffice Calcot. Az első oszlopba (A) írd be a számokat 1-től 5-ig lefelé. A következő oszlopba (B), írd be 6-tól 10-ig. Így kellene kinéznie:

	A	B
1	1	6
2	2	7
3	3	8
4	4	9
5	5	10

Most mentsd el a táblázatot „example1.xls” néven, abba a mappába, ahová a tesztkódot fogod lementeni. Így nem kell az útvonallal törődni.

Most töltsd le és telepítsd fel az XLRD-t (https://pypi.python.org/pypi/xlrd). Így használhatjuk (jobbra látható):

Mentsd el a fájlt mint example1.py ugyanabba a mappába, mint a táblázatot. Mivel a kód ilyen rövid, egyszerűen itt beszéljük meg. Természetesen az első sor importálja a könyvtárat. Aztán létrehozzuk az OpenFile függvényt és átadjuk a táblázat nevét (és az útvonalát ha szükséges) a függvénynek.

Most meghívjuk az open_workbook metódust, és megkapjuk a „book” objektumot. Aztán az nsheet attribútumot használjuk, hogy visszakapjuk az AKTÍV munkafüzetek számát. A munkafüzetek nevét is lekérhetjük. Ebben az esetben ezek az alapértelmezettek. A sheet_by_index metódust használjuk a Munkafüzet1 lekéréséhez a first_sheet objektumba. Most elkezdhetjük az adatok kinyerését. Lekérjük az információt az (1,1) cellapozícióból, amely a B2-nek felel meg (0-tól kezdődik a számolás, tehát az A1 cella (0,0) lenne). Kiírjuk onnan az adatokat, mind azt, hogy mi van a cellában, mind az értékét, így használhatnánk számoláshoz is, ha szeretnénk.

Ez elég könnyű volt, nemde? Most csináljunk valami hasznosat. Írd be a kódot és mentsd el „example.py” néven. Ez a példa kiírja a munkafüzet tartalmát.

Mivel már használtuk az első négy sort az előző példában, ezért átugorjuk őket. A „sheet.nrows” és a „sheet.ncols” attribútumok használatával lekérjük a sorok és oszlopok számát. Ez hasznos lehet, és nem csak azért, hogy tudjuk mivel van dolgunk; így „általános” rutinokat írhatunk, amelyek használják ezeket az értékeket a számítások során. Valójában a „rows”-t használjuk egy for ciklusban

```
import xlrd
def OpenFile(path):
      # Open and read excel file
      book = xlrd.open_workbook(path)
      # Get number of active workbooks
      print "Number of workbooks: ",book.nsheets
      # Get the names of those workbooks
      print "Workbook names: ",book.sheet_names()
      first_sheet = book.sheet_by_index(0)
      cell = first_sheet.cell(1,1)
      print "Cell at 1,1: ",cell
      print "Cell Value at 1,1: ",cell.value

if __name__ == "__main__":
      path = "example1.xls"
      OpenFile(path)
```

minden egyes sor adatainak kinyeréséhez.

Figyeld meg a sort a „first_sheet.row_slice”-szal. Ez lekéri a cellablokkokat az adott sorból. A szintaktika a következő:

```
X =
first_sheet.row_slice(RowIn-
Question, Start_Column, End_-
Column)
```

Tehát használtuk a sorok és az oszlopok számát a számításokhoz. A program kimenete valahogy így kell hogy kinézzen…

```
There are 5 rows in this
workbook.
There are 2 cols in this
workbook.
[number:1.0, number:6.0]
[number:2.0, number:7.0]
[number:3.0, number:8.0]
[number:4.0, number:9.0]
[number:5.0, number:10.0]
Press any key to continue . .
.
```

Még egy példát csinálunk, mielőtt befejezzük az e havi cikket. Ugorj a munkafüzetre, a C oszlopba, és írj be néhány dátumot. Bármilyen dátumot használhatsz. Most futtassuk újra az example2.py programot. Az én táblázatom így néz ki most:

```
import xlrd
def OpenFile(path):
      book = xlrd.open_workbook(path)
      first_sheet = book.sheet_by_index(0)
# Get the number of rows in this workbook
rows = first_sheet.nrows
      # get the number of columns in this workbook
      cols = first_sheet.ncols
      print "There are %d rows in this workbook." % rows
      print "There are %d cols in this workbook." % cols
      for r in range(0,rows):
      cells = first_sheet.row_slice(rowx=r,start_colx=0,end_colx=cols)
      print cells

if __name__ == "__main__":
      path = "example1.xls"
      OpenFile(path)
```

1	6	1/10/2014
2	7	4/15/2015
3	8	6/24/1986
4	9	9/30/1963
5	10	3/3/2000

És itt az én kimenetem:

```
There are 5 rows in this
workbook.
There are 3 cols in this
workbook.
[number:1.0, number:6.0, xld-
ate:41649.0]
[number:2.0, number:7.0, xld-
ate:42109.0]
[number:3.0, number:8.0, xld-
ate:31587.0]
[number:4.0, number:9.0, xld-
ate:23284.0]
[number:5.0, number:10.0,
xldate:36588.0]
Press any key to continue . .
.
```

Nos, ez nem az, amit vártunk. Úgy néz ki, hogy az excel a dátumokat értékként tárolja, és úgy formázza, ahogyan kérjük. Ez rendezéshez és számításokhoz hasznos lehet, de az adatok megjelenítéséhez nem megfelelő. Szerencsére a könyvtár készítői már gondoltak erre. Töröld a „print cells” sort, és cseréld le a lent látható kódra.

Itt végigmegyünk minden cellán a listában, és leellenőrízzük, hogy a cella típusa XL_CELL_DATE. Ha az, akkor átalakítjuk tuple-nek. YYYY,MM,DD alakban tárolódik.

Egyszerűen MM/DD/YYYY formátumban írjuk ki. Ez az új pro-

```
      for c in cells:
          if c.ctype == xlrd.XL_CELL_DATE:
          date_value = xlrd.xldate_as_tuple(c.value,book.datemode)
          dt = str(date_value[1]) + "/" + str(date_value[2]) + "/" + str(date_value[0])
          print dt
          else:
          print c.value
```

gramunk kimenete…

```
There are 5 rows in this
workbook.
There are 3 cols in this
workbook.
1.0
6.0
1/10/2014
2.0
7.0
4/15/2015
3.0
8.0
6/24/1986
4.0
9.0
9/30/1963
5.0
10.0
3/3/2000
Press any key to continue . .
.
```

Csak hogy tudd, van egy könyvtár ugyanezektől a nagyszerű emberektől, az XLWT, amely lehetővé teszi az Excel fájlok írását. Egy nagyszerű oktatóanyag és a két könytárhoz tartozó dokumentáció megtalálható a http://www.python-excel.org/ oldalon.

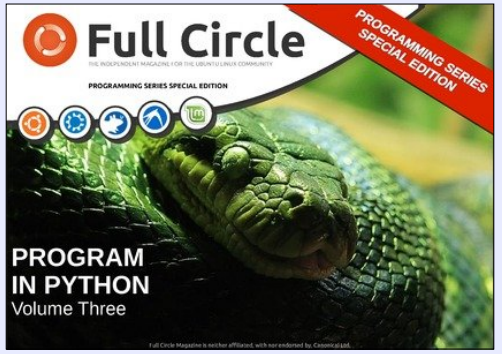
Az example3.py forráskódja megtalálható a pastebinen: http://pastebin.com/bWz7beBw. Remélhetőleg jövő hónapban újra látlak.

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

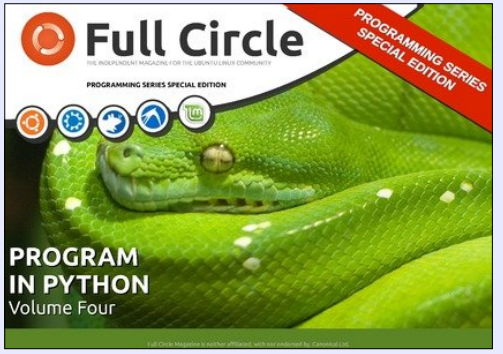
Pythonos Különkiadások:

http://fullcirclemagazine.org/issue-py01/

http://fullcirclemagazine.org/issue-py02/



http://fullcirclemagazine.org/python-special-edition-issue-three/



http://fullcirclemagazine.org/python-special-edition-volume-four/

http://fullcirclemagazine.org/python-special-edition-volume-five/

http://fullcirclemagazine.org/python-special-edition-volume-six/

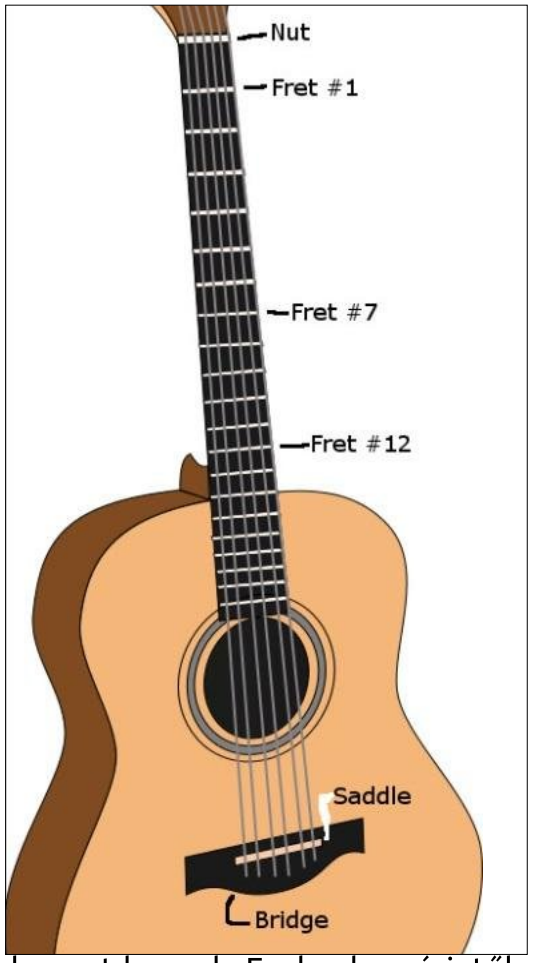
Programozzunk Pythonban – 59. rész

Írta: Greg D. Walters
Fordította: Tulipán Attila

Először is engedjétek meg, hogy gratuláljak Ronnienak és csapatának a 100. számhoz. Nagy kiváltság, hogy ennek a mérföldkőnek számító eseménynek a részese lehetek.

Ez alkalommal arra gondoltam, hogy a legújabb szenvedélyemmel kapcsolatban osztok meg néhány információt. Elkezdtem húros hangszerek (olyan, mint gitár és hegedű) javításával és építésével foglalkozni. Hiszed vagy sem, nagyon sok matek kapcsolódik a hangszerekhez. Ma a mateknak azzal a részével fogunk foglalkozni, amely a húr hosszára vonatkozik, és hogy hova kell helyezni az érintőket a fogólapon.

Nézd meg a mellékelt gitárábrát. Bejelöltem egy pár részét a képen. Az említésre méltó részek: a felső nyereg a lefogó lap felső végén (Nut), az érintők (Fret), a híd (Bridge) az alsó részen, és fehér vonal a hídon az úgynevezett alsó nyereg (Saddle). Az érintők célja, hogy jelöljék azt a pontos helyet, amellyel megváltoztatjuk a húr hosszát, amelyet lefogva egy tiszta



hangot kapunk. Ezeknek az érintőknek a helyzete nem tetszőleges, hanem pontosan, matematikailag meghatározott.

Nos, a rezgő húr fizikája a következő: ha megfelezed az elméletileg tökéletes rezgő húr hosszát, akkor a rezgés frekvenciáját megduplázod. A gitár esetében ez a húrhossz a felső és alsó nyereg közötti távolság. Ez a távolság utal a gitároknál a méretre (skálázott hossz). A fele távolság, ami a dupla frekvenciát adja a 12. érintő. Ha pontosan lett készítve, és csak puhán oda helyezed az ujjad a húrra ezen a helyen, akkor pengetésre egy kellemes hangot kapsz. Van még egy pár hely, ahol ezt kapjuk, de a 12. érintő tökéletes hely erre a duplázásra – ezzel egy oktávot lépünk a hangokban.

Másként skálázott távok különböző tapintás-érzést és hangszínt adnak. Például az olyan gitárok, mint a Fender Stratocaster®-nek a hossza 25 1/2”, mellyel egy gazdag, erős harangszerű hangzás jön létre. Másrészt a Gibson gitárok gyakran 24 ¾” hosszak. Ezzel „puhább húrozás” érhető el, melly könnyebb fogásérzetet ad, és melegebb a hangszín is, amit ad. Más gitárgyártók esküsznek a 25” hosszra, melylyel szerintük tisztább hang érhető el, mint a másik két „standard” gitár méretezése adhat.

Úgyhogy a gitárkészítés képességével bíró hangszerkészítőnek kell előállni a saját méretezésével: az érintők távolságának meghatározásával, amelyeket újra kell kalkulálni. Ez az a tevékenység, amely a hangszerkészítők foglalatossága évszázadok óta.

Korábban volt egy technika erre, az úgynevezett 18-as szabály. Ezzel a szabállyal osztották a lefogó lap teljes hosszát 18-al, úgyhogy a következő osztásnál az előző érintő távolságát kivonták. Bár ez a technika működött, de még így is volt, hogy a hangok hamisak voltak, és ez így volt akkor, amikor minél magasabb hangokat játszott a zenész. Jelenleg egy másik konstanst használnak az osztásra, ez pedig a 17,817. Ezzel az új konstanssal a 12. érintő vagy oktáv pontosan a teljes húrhossz felénél van.

Nos, ez a számítás könnyen elvégezhető papírral és ceruzával, vagy egy egyszerűbb számológéppel. De hasonlóan könnyű egy Python program elkészítése, amely elvégzi a kalkulációt számunkra egy másodperc alatt. Amint megvannak a pontos pozíciók, már fűrészelhe-

ted is az érintő helyét, aztán mehet is a helyére kalapáccsal.

Akkor nézzük is a programot.

Egy olyan programot akarunk készíteni, amely bekéri a gitár hosszát (vagy basszusgitárét), elvégzi a számítást, és kiadja a távolságokat. A számítások és a visszakapott méretek mind hüvelykben vannak, tehát minden barátunk, aki méterben számol, tegyen még bele egy átváltást. Majdnem öt év után, ennek már könnyen kellene mennie.

Nem kell semmilyen könyvtárat beimportálnunk, úgyhogy egyből kezdjük a változóink definiálását.

```
ScaleLength = 0
CumulativeLength = 0
```

Aztán egy függvényt készítünk, amit ismétlésszerűen meghívunk, ahogyan megyünk lejjebb a fogólapon. Ennek a függvénynek két paramétere lesz: az egyik a teljes hossz, a másik egy halmozódó érték, amely a felső nyereg és az előző érintő távolsága.

Ebben a függvényben, vesszük a teljes hosszt, kivonjuk a halmozott távolságot, és ezt értékül adjuk a BridgeToFret változónak. Aztán vesszük ezt az értéket, elosztjuk a konstansunkkal (17,817), hozzáadjuk ezt az értéket a halmozódó távolság értékéhez, és visszaadja ezt az értéket a meghívó függvénynek. Megjegyezném, hogy egyszerűen visszaadhattuk volna ezt az értéket anélkül, hogy változóhoz rendeltük volna, de ha valaha ellenőrizni akarjuk a kiszámolt értéket, akkor könnyebb, ha az értéket egy változóhoz rendeljük, mielőtt a hívó függvénynek visszaadnánk.

Most pedig a konkrét munkát végző függvényt készítjük el. Már korábban csináltunk ehhez hasonló dolgokat. Átadjuk paraméterként a húr hosszát, és egy ciklus ismétlődik 24-szer (a 24 érintő miatt; a bejárás 1-től 25-ig [range(1,25)]). Ha a munkád 24 érintőnél kevesebbet igényel, akkor is megkapod az összes érintő helyes pozícióját. Én a 24-et választottam, mert ez a maximum érintőszám, amit a legtöbb gitár használ. Amikor a ciklushoz érünk, akkor leellenőrizzük az érintők számát [number(x)], és ha az 1, akkor a halmozódó táv értékének 0-t adunk, mivel ez azt jelenti, hogy ez az első ciklus. Máskülönben átadjuk az utolsó halmozódó távolság értékét, és megkapja a függvényből az eredményt. Végezetül kiíratjuk az összes érintőszámot a halmozódó távolság értékével együtt, formázott szövegként.

```
def CalcSpacing(Length,NTF):
    BridgeToFret = Length-NTF
    NutToFret = (BridgeToFret/17.817) + NTF
    return NutToFret
```

Végül itt a kódunk, ami bekéri a húr hosszát. Biztos vagyok benne, hogy emlékszel a raw_input függvény formájára, amit oly sok alkalommal használtunk már. Amire talán nem emlékszel, hogy a raw_input függvény sztring változótípussal tér vissza, úgyhogy amikor a DoWork függvénynek át akarjuk adni az értéket, akkor lebegőpontos számként kell átadni, hogy a függvényünk helyesen működjön. Természetesen sztring változótípusként is átadhatnánk, de akkor a DoWork függvényen belül kellene lekezelnünk az átkonvertálást.

```
ScaleLength = raw_input("Please enter Scale Length of guitar -> ")
DoWork(float(ScaleLength))
```

Ennyi lenne az egész. Lehet, azon csodálkozol, hogy mi hasznod ebből a programból, ha nem fogsz soha gitárt a nulláról építeni. Ez a program akkor is hasznos lehet, ha használt gitárt tervezel vásárolni, vagy egy mozgatható híddal rendelkező gitárt akarsz hangolni (az nem olyan gitár, mint a képen). Ugyanakkor, ha gitározol, akkor lehet, hogy ez olyan infó volt, amit nem tudtál eddig a gitárokról.

Természetesen a kód az alábbi pastebin linken elérhető: http://pastebin.com/A2RNECt5.

Viszlát, a következő alkalomig! Addig is jó kódolást!

```
def DoWork(ScaleLength):
    CumulativeLength = 0
    for x in range(1,25):
        FretNumber = x
        if FretNumber == 1:
            CumulativeLength = CalcSpacing(ScaleLength,0)
        else:
            CumulativeLength = CalcSpacing(ScaleLength,CumulativeLength)
        print("Fret=%d,NutToFret=%.3f" % (FretNumber,CumulativeLength))
```

Közreműködnél?

A FULL CIRCLE-nek szüksége van rád!

Egy magazin, ahogy a Full Circle is, nem magazin cikkek nélkül. Szükségünk van játékok, programok és hardverek áttekintő leírására, ezenkívül bármire, amit elmondanátok a *buntu felhasználóknak. A cikkeiteket küldjétek a következő címre: articles@fullcirclemagazine.org

Folyamatosan keressük a cikkeket a magazinba. Segítségül nézzétek meg a **Hivatalos Full Circle Stílus Útmutatót**: http://url.fullcirclemagazine.org/75d471

Véleményed és Linuxos tapasztalataidat a letters@fullcirclemagazine.org címre,
Hardver és szoftver **elemzéseket** a reviews@fullcirclemagazine.org címre,
Kérdéseket a „Kávé” rovatba a questions@fullcirclemagazine.org címre,
Képernyőképeket a misc@fullcirclemagazine.org címre küldhetsz,
... vagy látogasd meg a **fórumunkat** a fullcirclemagazine.org címen.

Full Circle heti hírek:

A heti híreket elérheted az alábbi RRS-linken:
http://fullcirclemagazine.org/feed/podcast

Ha a szabadban vagy, akkor elérheted a Stitcher Radión (Android/iOS/web):
http://www.stitcher.com/s?fid=85347&refid=stpr

és a TuneIn-en keresztül, itt:
http://tunein.com/radio/Full-Circle-Weekly-News-p855064/

A Full Circle magazin beszerezhető:

EPUB – Az utóbbi kiadások megtalálhatók epub formátumban a letöltési oldalon. Ha bármi problémád lenne az epub fájllal, küldj e-mailt a mobile@fullcirclemagazine.org címre.

Issuu – Olvashatod a Full Circle magazint online az Issuu-n: http://issuu.com/fullcirclemagazine. Oszd meg és értékeld a magazint, hogy minél többen tudjanak a magazinról és az Ubuntu Linuxról.

Google Play – Már olvashatod a Full Circle magazint a Google Play/Books oldalán. Keresd a „full circle magazin”-t, vagy kattints ide: https://play.google.com/store/books/author?id=Ronnie+Tucker

A Full Circle Csapat

Szerkesztő – Ronnie Tucker
ronnie@fullcirclemagazine.org
Webmester – Lucas Westermann
admin@fullcirclemagazine.org

Szerkesztők és Korrektorok
Mike Kennedy, Gord Campbell, Robert Orsino, Josh Hertel, Bert Jerred, Jim Dyer és Emily Gonyer

Köszönet a Canonical-nek, a fordítócsapatoknak a világban és **Thorsten Wilms**-nek az FCM logóért.

Full Circle magazin
Magyar Fordítócsapat

Koordinátor:
Pércsy Kornél

Fordítók:
A fordítók csapata

Lektor:
A fordítócsapat lektorai

Szerkesztő/Korrektor:
Heim Tibor